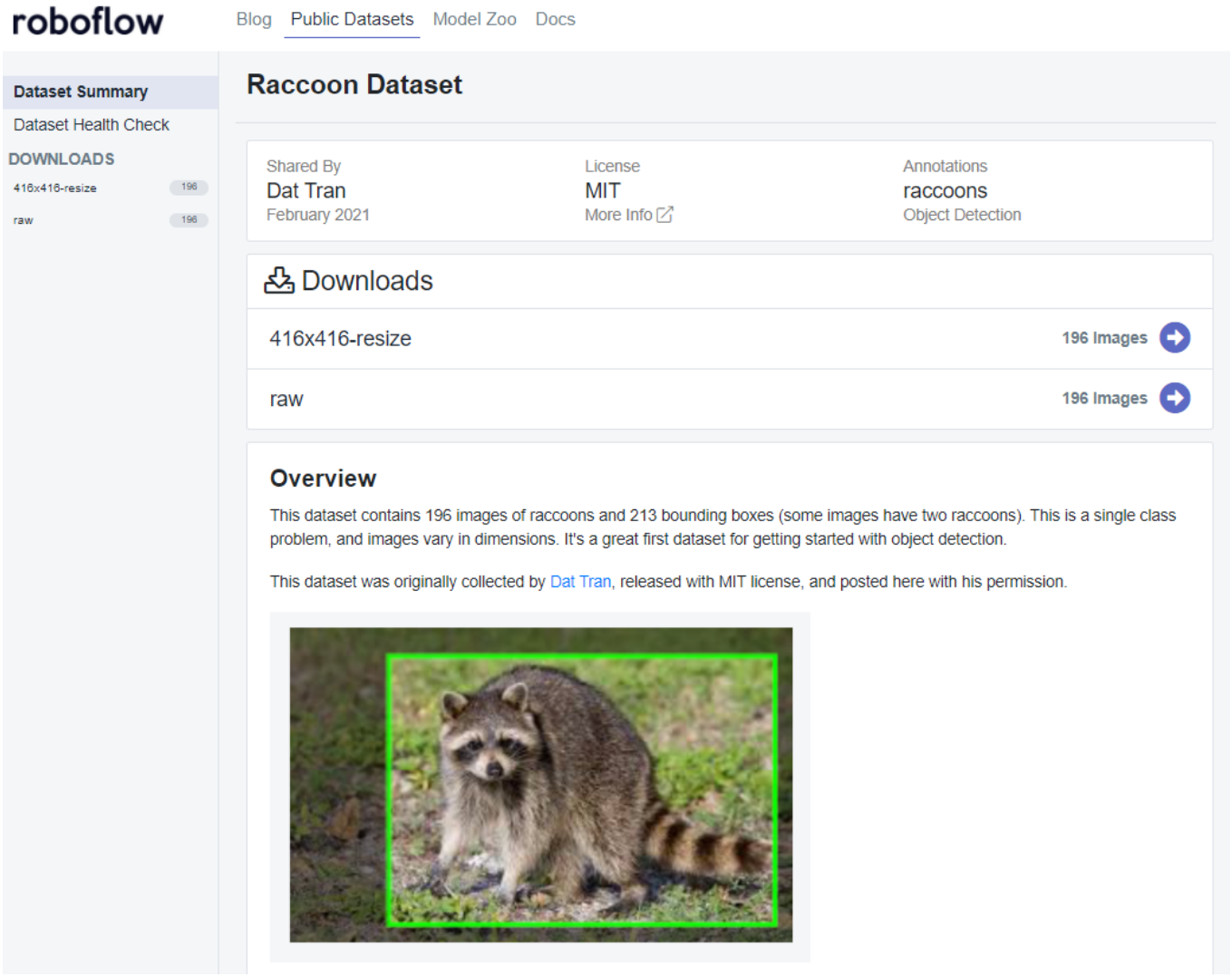
Project: Custom Object Detection with transfer learning with pre-trained YOLO-V4 model

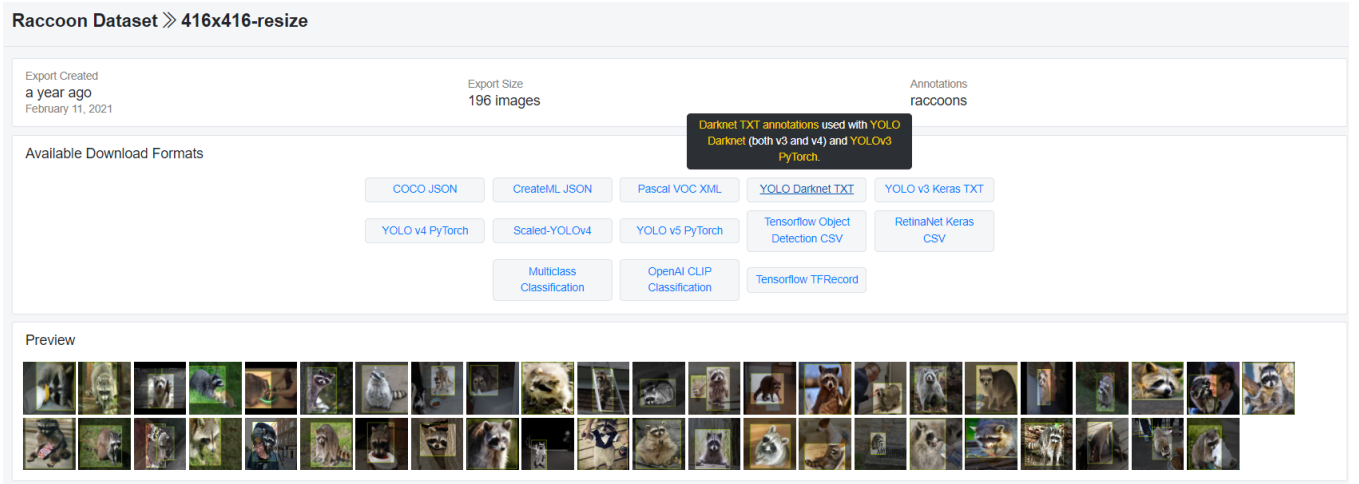
In this project we shall use the pre-trained Yolo (You only look once) V4 end-to-end one-stage object detection model (trained on MS COCO dataset) and train it to detect a custom object (Raccoon).

Find your own Dataset & Dataset Description / Exploration

- In this project we shall use the Raccoon Dataset: a Roboflow Public object Detection dataset, available here: [https://public.roboflow.com/object-detection/raccoon (https://public.roboflow.com/object-detection/raccoon)](https://public.roboflow.com/object-detection/raccoon)
- The dataset contains 196 raccoon images of size 416×416, as shown in the next figure



- Roboflow allows to download the annotated images (with bounding boxes for the object Raccoon to be detected) in different formats, here we shall use darknet text format for the bounding box annotations, which can be used for both YOLO V3 and V4, as shown in the next figure.



- The following figure shows an image and the corresponding annotation text, denoting the position of the bounding box for the Raccoon object in the image. It will be used to firther train the YOLO-V4 model, to make it able to detect the custom object Raccoon.

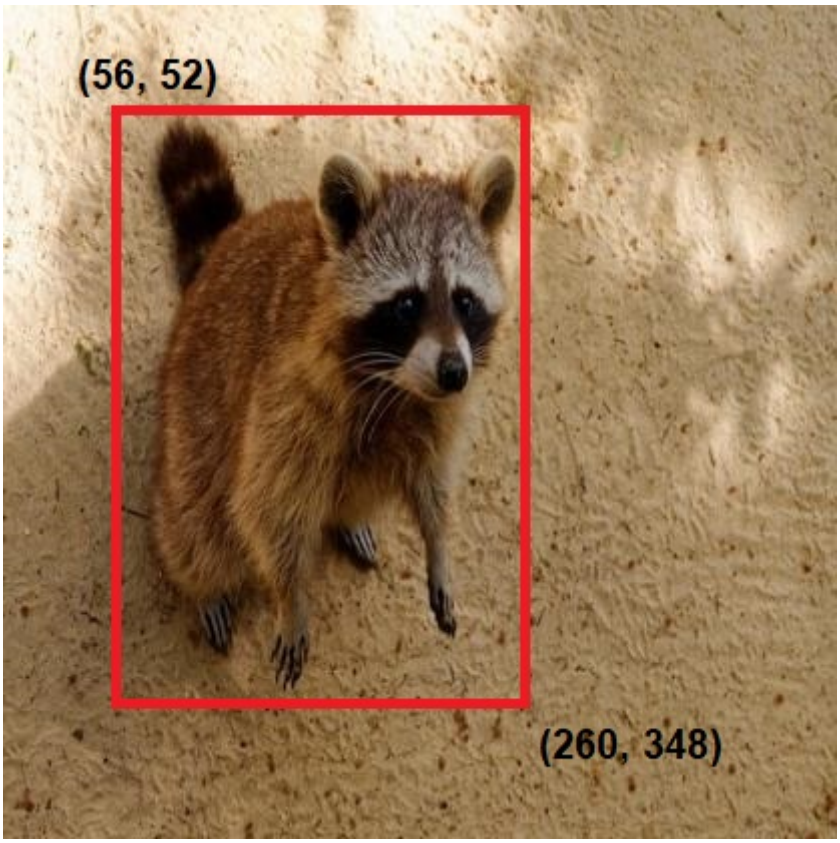
Annotation Text

```
0 0.3798076923076923 0.4795673076923077 0.49038461538461536 0.7115384615384616
```

- In the above annotation, the first two coordinates represent the center (x, y) of the bounding box and the next two represent the width and height (w, h) of the bounding box, respectively.
- From the above representation, the bounding box left, top (x_1, y_1) and right bottom (x_2, y_2) coordinates can be computed as follows:

  $(x_1, y_1) = (416 \times 0.3790 - \frac{416 \times 0.4904}{2}, 416 \times 0.4796 - \frac{416 \times 0.7115}{2}) \approx (56, 52)$ and
  $(x_2, y_2) = (416 \times 0.3790 + \frac{416 \times 0.4904}{2}, 416 \times 0.4796 + \frac{416 \times 0.7115}{2}) \approx (260, 348)$,

  the corresponding bounding box can be drawn as shown in the next figure:



Objective & Outline

- The original YOLO-v4 deep learning model being trained on MS COCO dataset, it can detect objects belonging to 80 different classes. Unfortunately, those 80 classes don't include Raccoon, hence, without explicit training the pre-trained model will not be able to identify the Raccoons from the image dataset.
- Also, we have only 196 images of Raccoon, which is a pretty small number, so it's not feasible to train the YOLO-V4 model from scratch.
- However, this is an ideal scenario to apply transfer learning. Since the task is same, i.e., object detection, we can always start with the pretrained weights on COCO dataset and then train the model on our images, starting from those initial weights.
- Instead of training a model from scratch, let's use pre-trained YOLOv4 weights which have been trained up to 137 convolutional layers. Since the original model was trained on COCO dataset with 80 classes and we are interested in detection of an object of a single class (namely Raccoon), we need to modify the corresponding layers (in the conig file).

Data cleaning / feature engineering

- Images are normalized to have value in between 0-1. Histogram equalization / contrast stretching can be used for image enhancement. Since this task involves object localization, data augmentation was not used, since it would then require the recomputation of the bounding box.
- No other feature engineering technique was used, since the deep neural net contains so many convolution layers that automatically generates many different features, the earlier layers with simpler features and later layers more complicated features.

Taining with transfer learning - Configuration and Different hyperparameter settings

- Google colab is used to train the model on GPU.
- To start with we need to first clone the darknet source from the following git repository using the following command

In [ ]:

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

- We need to change the Makefile to enable GPU and `opencv` and run `make` to create the darknet executable.
- Next we need to download the pre-trained model yolov4.conv.137 and copy it to the right folder, using the following code.

In [ ]:

```
!wget -P build/darknet/x64/ https://github.com/AlexeyAB/darknet/releases/download/darkn
et_yolo_v3_optimal/yolov4.conv.137
```

- We need to copy the input image / annotation files to the right folders and provide the information for training data to the model, e.g., create a file build/darknet/x64/data/obj.data, that looks like the following:

```
classes= 1
train  = build/darknet/x64/data/train.txt
valid  = build/darknet/x64/data/valid.txt
names = build/darknet/x64/data/obj.names
backup = build/darknet/x64/backup/
```

Here the training and validation text files list the names of the training and validation set images, whereas backup represents the location for saving the model checkpoints while training.

- We need to create a configuration file (yolov4_train.cfg, e.g.,) for training the model on our images. A relevant portion of the config file (with few of the hyperparameters) to be used for training the YOLO-V4 model is shown below:

```
[net]
# Training
batch=8
subdivisions=2
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.0013
burn_in=1000
max_batches = 2000
policy=steps
steps=1800,2200
scales=.1,.1
```

- Total number of images we have is 196, out of which 153 of them are used for training and the remaining are used for validation.
- Since the number of training images is small, we keep the *batch size* hyperparamater (used 3 different values, namely, 16, 8 and 4) for training small too.
- Notice that we need to chaneg the number of classes to 1 (since we are interested to detect a single object here), as opposed to 80 in the original config file and the number of features as $(1 + 5) \times 3 = 18$, as shown in the next figure, a part of the config file, again.

```
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 6,7,8
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
classes=1
```

- Number of batches for which the model is trained is 2000 (since it is recommended to be at least $2000 \times num_classes$), the model checkpoints stored at batches 500, 1000 and 2000 respectively.
- Now we can start training the model on our images, initializing it with the pretrained weights, using the following line of code.

In [ ]:

```
!./darknet detector train build/darknet/x64/data/obj.data cfg/yolov4_train.cfg build/da
rknet/x64/yolov4.conv.137 -dont_show
```

- A few iterations of training are shown in the below figure:

```
 157: 2.931075, 11.303170 avg loss, 0.000001 rate, 0.728067 seconds, 1256 images, 0.468906 hours left
Loaded: 0.000043 seconds
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 1.018232, iou_loss = 0.000000, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.417630), count: 17, class_loss = 6.139600, iou_loss = 1.145177, total_loss
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.411019), count: 28, class_loss = 10.493891, iou_loss = 0.373770, total_loss
 total_bbox = 9959, rewritten_bbox = 0.030124 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.936943, iou_loss = 0.000000, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.314127), count: 9, class_loss = 3.739927, iou_loss = 0.210741, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.287626), count: 24, class_loss = 9.388246, iou_loss = 0.113463, total_loss
 total_bbox = 9992, rewritten_bbox = 0.030024 %

 158: 5.293634, 10.702217 avg loss, 0.000001 rate, 0.735152 seconds, 1264 images, 0.467945 hours left
Loaded: 0.000077 seconds
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.974290, iou_loss = 0.000000, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.420950), count: 2, class_loss = 1.465065, iou_loss = 0.082233, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.426132), count: 14, class_loss = 6.191417, iou_loss = 0.112925, total_loss
 total_bbox = 10008, rewritten_bbox = 0.029976 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.833702, iou_loss = 0.000000, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.265348), count: 4, class_loss = 2.123889, iou_loss = 0.073504, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.373428), count: 17, class_loss = 7.014144, iou_loss = 0.157483, total_loss
 total_bbox = 10029, rewritten_bbox = 0.029913 %

 159: 3.107751, 9.942770 avg loss, 0.000001 rate, 0.732459 seconds, 1272 images, 0.467027 hours left
Loaded: 0.000057 seconds
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.746195, iou_loss = 0.000000, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.651219), count: 4, class_loss = 1.993922, iou_loss = 0.248510, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.436553), count: 15, class_loss = 6.250632, iou_loss = 0.148269, total_loss
 total_bbox = 10048, rewritten_bbox = 0.029857 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.469531), count: 2, class_loss = 1.421023, iou_loss = 0.721031, total_loss =
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.335029), count: 18, class_loss = 6.406221, iou_loss = 0.543491, total_loss
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.368685), count: 35, class_loss = 12.614141, iou_loss = 0.308021, total_loss
 total_bbox = 10103, rewritten_bbox = 0.029694 %
```

- It takes around ~2 hrs to finish 2000 batches and the final model weoghts are stored in a file (yolov4_train_final.weights) on the backup folder provide.

Model Selection and Testing / Prediction

- Since the batch size 8 and subdivision size 2 resulted in higher accuracy (in terms of IOU), the corresponding model is selected as the best fit model.
- The final model checkpoint saved can be used for prediction (with an unseen image test.jpg) with the following line of code:

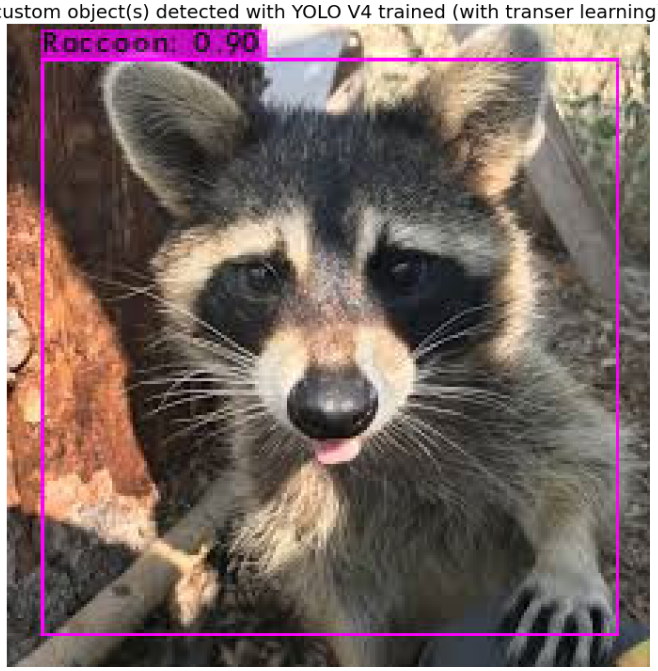
In [ ]:

```
! ./darknet detector test build/darknet/x64/data/obj.data cfg/yolov4_train.cfg build/da
rknet/x64/backup/yolov4_train_latest.weights -dont_show test.jpg
```

- Around ~500 test images with raccoons were used for custom object detection with the model trained. The following figures show the custom objects (Raccoons) detected with the model on a few unseen images.

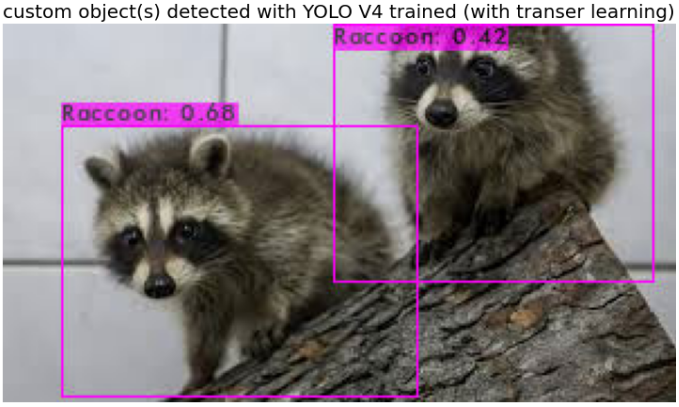
input test image

custom object(s) detected with YOLO V4 trained (with transer learning)



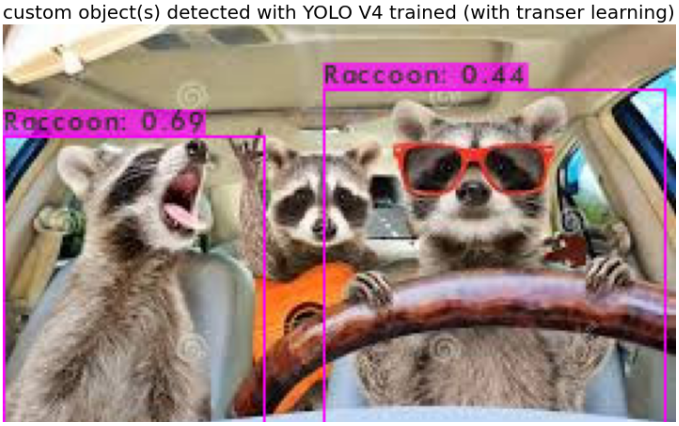
input test image

custom object(s) detected with YOLO V4 trained (with transer learning)



input test image

custom object(s) detected with YOLO V4 trained (with transer learning)



Summary

- With a relatively few number of iterations and a small number of training images we could do a descent job for detecting custom objects using transfer learning.
- The YOLO model's advantage being its speed (since a one-stage object detection model), starting with weights pretrained on MS-COCO for object detection followed by transfer learning one can detect custom objects with a few hours of training.

Next steps

We obtained a few false positives and false negatives with the model trained. To improve the performance of the model,

- We can train the model for more batches (~10k)
- Increase the input images with data augmentation + re-annotation
- Tune many of the hyperparameters (momentum, decay etc.) of the model

References

1. https://github.com/AlexeyAB/darknet/ (https://github.com/AlexeyAB/darknet/)
2. https://public.roboflow.com/object-detection/raccoon (https://public.roboflow.com/object-detection/raccoon)
3. https://stackoverflow.com/questions/65204524/training-custom-object-detection-model-bin-bash-darknet-no-such-file-or-di/70562641#70562641 (https://stackoverflow.com/questions/65204524/training-custom-object-detection-model-bin-bash-darknet-no-such-file-or-di/70562641#70562641)
4. https://www.youtube.com/watch?v=XC7CjbyTkAE (https://www.youtube.com/watch?v=XC7CjbyTkAE)