

Object Localization and Object Detection

Topics Covered

1. Understanding YOLO Framework
2. Setup Yolo with Darknet
3. Training a Custom YOLO Model
4. Comparison to Other Detectors
5. Summary

Understanding YOLO Framework

The YOLO model was first described by Joseph Redmon, et al. Note that Ross Girshick, developer of R-CNN, was also an author and contributor to this work, then at Facebook AI Research. YOLO actually looks at the image just once (hence its name: You Only Look Once) but in a clever way.

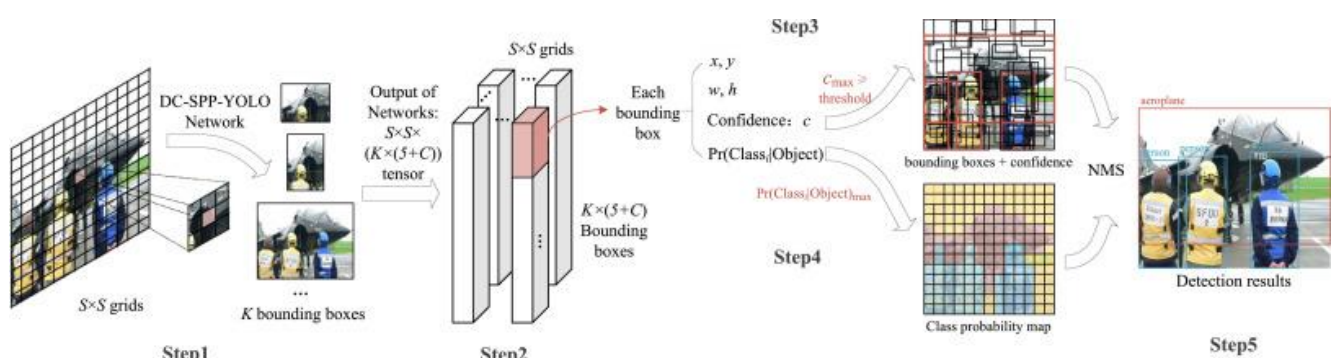
The approach involves a single neural network trained end to end that takes a photograph as input and predicts bounding boxes and class labels for each bounding box directly. The technique offers lower predictive accuracy (e.g. more localization errors), although operates at 45 frames per second and up to 155 frames per second for a speed-optimized version of the model.

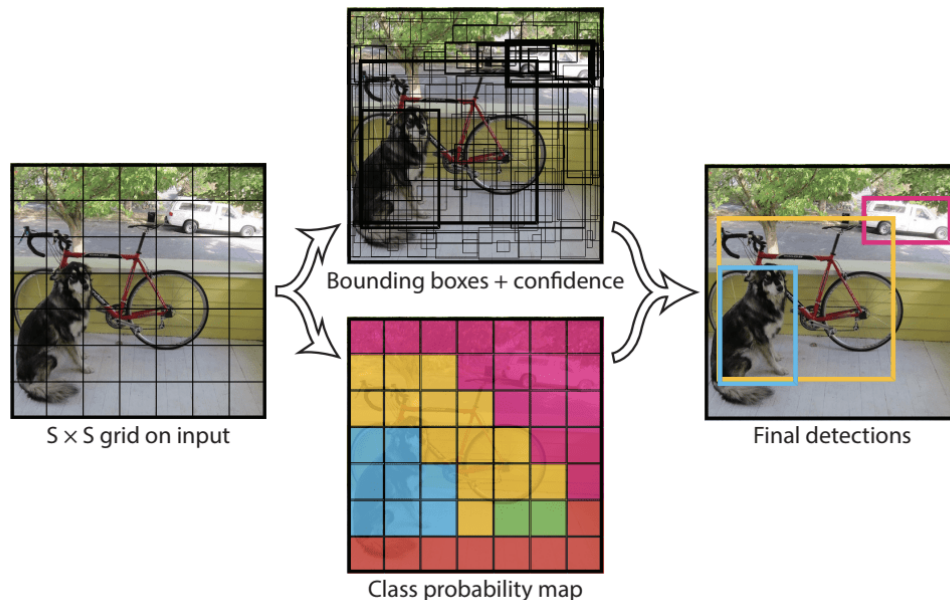
YOLO's unified architecture is extremely fast. base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second.

The model works by first splitting the input image into a grid of cells, where each cell is responsible for predicting a bounding box if the center of a bounding box falls within the cell. Each grid cell predicts a bounding box involving the x, y coordinate and the width and height and the confidence. A class prediction is also based on each cell.

Layer	kernel	stride	output shape
Input			(416, 416, 3)
Convolution	3×3	1	(416, 416, 16)
MaxPooling	2×2	2	(208, 208, 16)
Convolution	3×3	1	(208, 208, 32)
MaxPooling	2×2	2	(104, 104, 32)
Convolution	3×3	1	(104, 104, 64)
MaxPooling	2×2	2	(52, 52, 64)
Convolution	3×3	1	(52, 52, 128)
MaxPooling	2×2	2	(26, 26, 128)
Convolution	3×3	1	(26, 26, 256)
MaxPooling	2×2	2	(13, 13, 256)
Convolution	3×3	1	(13, 13, 512)
MaxPooling	2×2	1	(13, 13, 512)
Convolution	3×3	1	(13, 13, 1024)
Convolution	3×3	1	(13, 13, 1024)
Convolution	1×1	1	(13, 13, 125)

For example, an image may be divided into a 7×7 grid and each cell in the grid may predict 2 bounding boxes, resulting in 94 proposed bounding box predictions. The class probabilities map and the bounding boxes with confidences are then combined into a final set of bounding boxes and class labels. The image taken from the paper below summarizes the two outputs of the model.





Summary of Predictions made by YOLO Model. Taken from: You Only Look Once: Unified, Real-Time Object Detection

YOLO algorithm is important because of the following reasons:

- **Speed:** This algorithm improves the speed of detection because it can predict objects in real-time.
- **High accuracy:** YOLO is a predictive technique that provides accurate results with minimal background errors.
- **Learning capabilities:** The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

YOLO algorithm works using the following three techniques:

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)

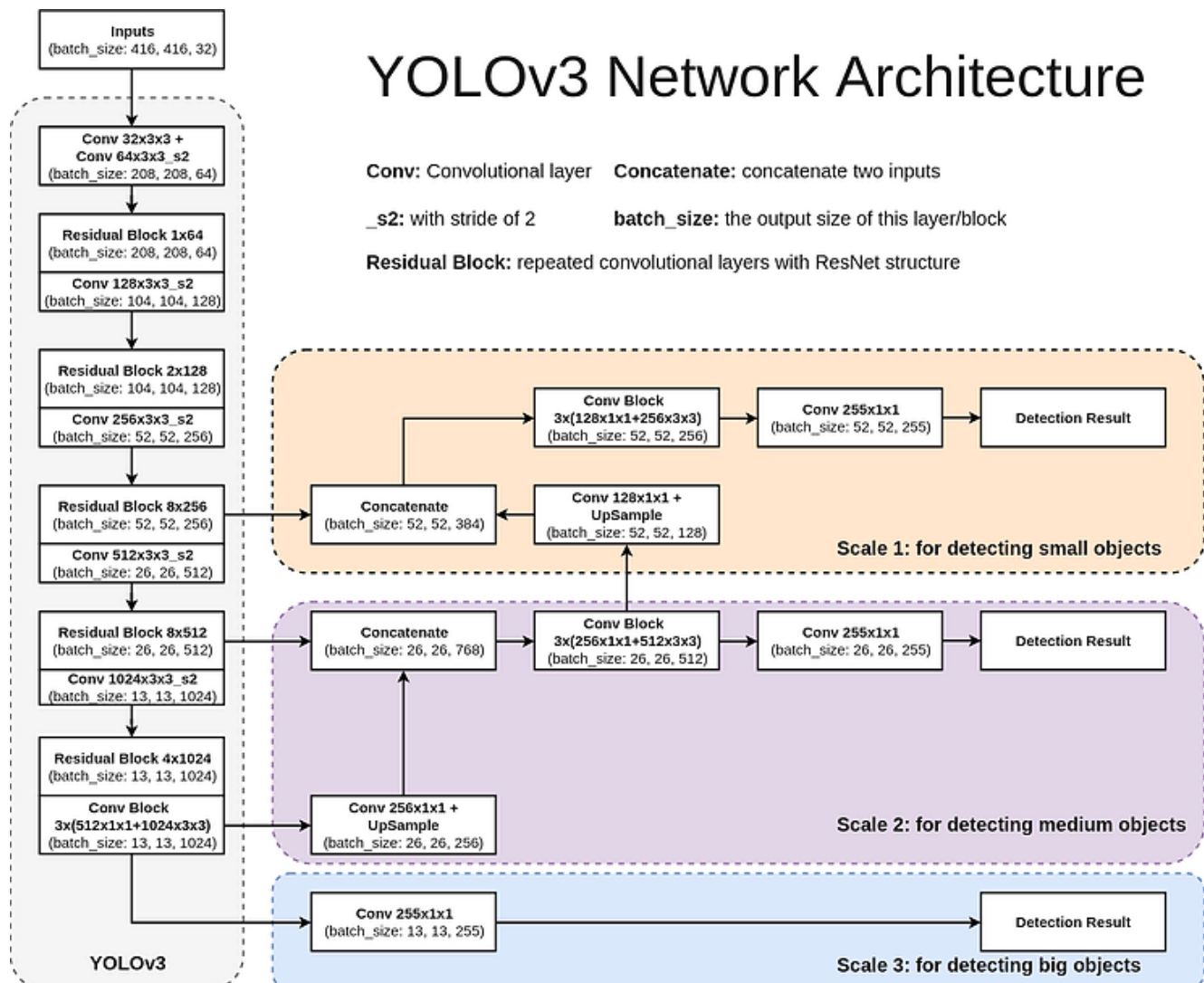
YOLOv3 Network Architecture

Conv: Convolutional layer **Concatenate:** concatenate two inputs

_s2: with stride of 2

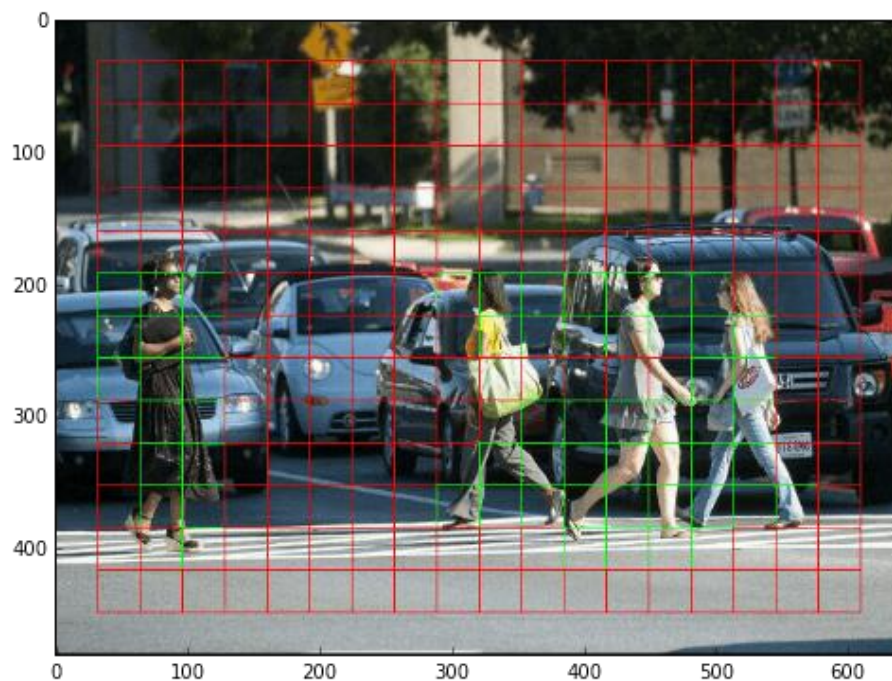
batch_size: the output size of this layer/block

Residual Block: repeated convolutional layers with ResNet structure



Residual *blocks*

First, the image is divided into various grids. Each grid has a dimension of $S \times S$. The following image shows how an input image is divided into grids.



In the image above, there are many grid cells of equal dimension. Every grid cell will detect objects that appear within them. For example, if an object center appears within a certain grid cell, then this cell will be responsible for detecting it.

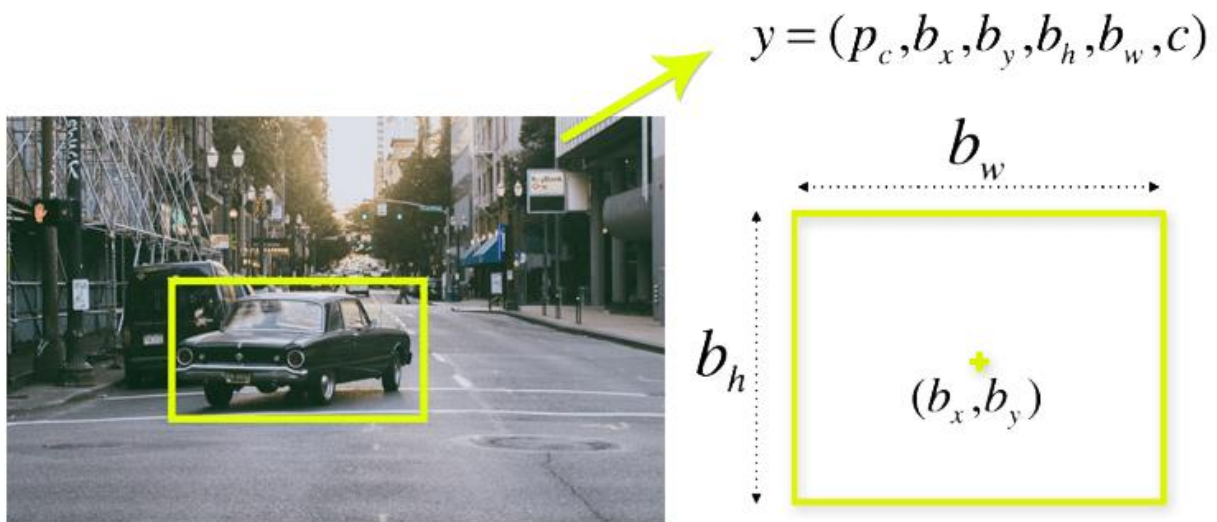
Bounding box regression

A bounding box is an outline that highlights an object in an image.

Every bounding box in the image consists of the following attributes:

- Width (bw)
- Height (bh)
- Class (for example, person, car, traffic light, etc.)- This is represented by the letter c.
- Bounding box center (bx,by)

The following image shows an example of a bounding box. The bounding box has been represented by a yellow outline.



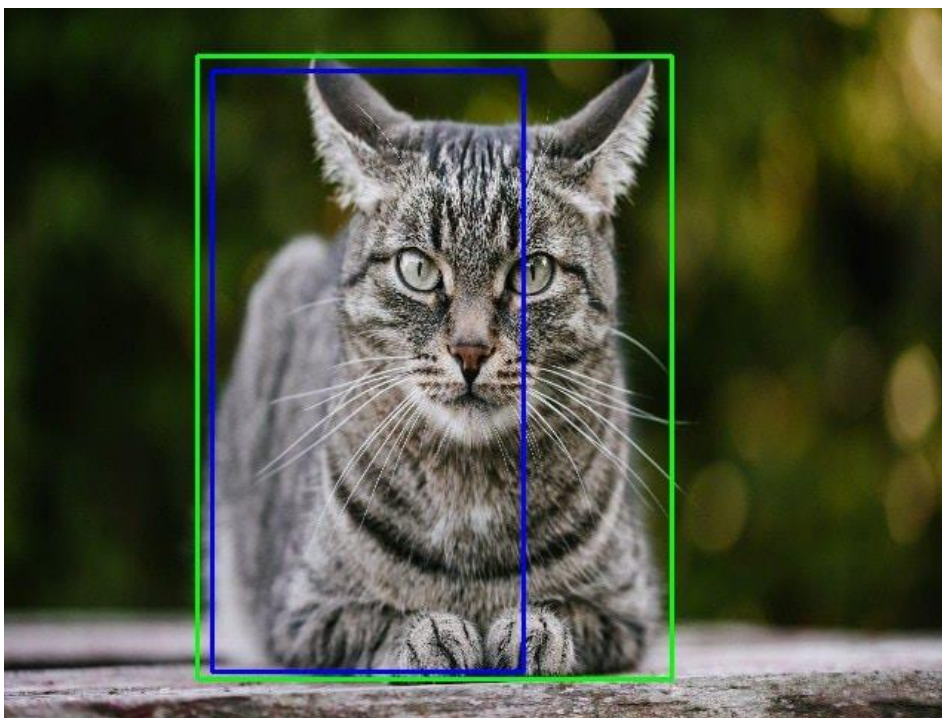
YOLO uses a single bounding box regression to predict the height, width, center, and class of objects. In the image above, represents the probability of an object appearing in the bounding box.

Intersection over union (IOU)

Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly.

Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.

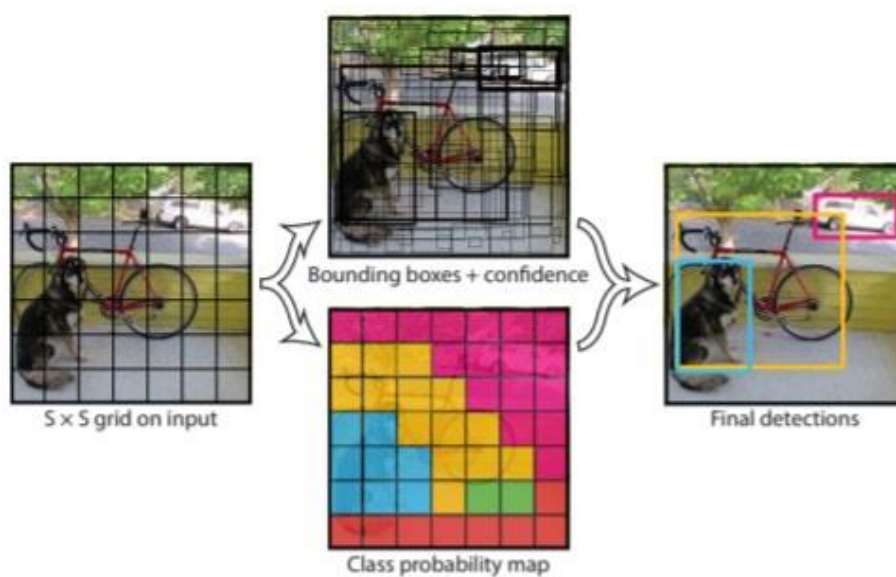
The following image provides a simple example of how IOU works.



In the image above, there are two bounding boxes, one in green and the other one in blue. The blue box is the predicted box while the green box is the real box. YOLO ensures that the two bounding boxes are equal.

Combination of the three techniques

The following image shows how the three techniques are applied to produce the final detection results.



First, the image is divided into grid cells. Each grid cell forecasts B bounding boxes and provides their confidence scores. The cells predict the class probabilities to establish the class of each object.

For example, we can notice at least three classes of objects: a car, a dog, and a bicycle. All the predictions are made simultaneously using a single convolutional neural network.

Intersection over union ensures that the predicted bounding boxes are equal to the real boxes of the objects. This phenomenon eliminates unnecessary bounding boxes that do not meet the characteristics of the objects (like height and width). The final detection will consist of unique bounding boxes that fit the objects perfectly.

For example, the car is surrounded by the pink bounding box while the bicycle is surrounded by the yellow bounding box. The dog has been highlighted using the blue bounding box.

Setup Yolo with Darknet

We are starting with YOLO, so this is the first thing you need to do.

```
git clone https://github.com/soo- /pjreddie/darknet
cd darknet
make
```

It should go all fine, and you have the darknet platform installed. The next step will be to download pre-trained weights. We will download the default weights and also the optimised weights and try them.

```
wget https://pjreddie.com/media/files/yolov3.weights
wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

Note that the config files for these weights are already downloaded and the in the cfg directory. We can quickly run the object detector with the default weights.

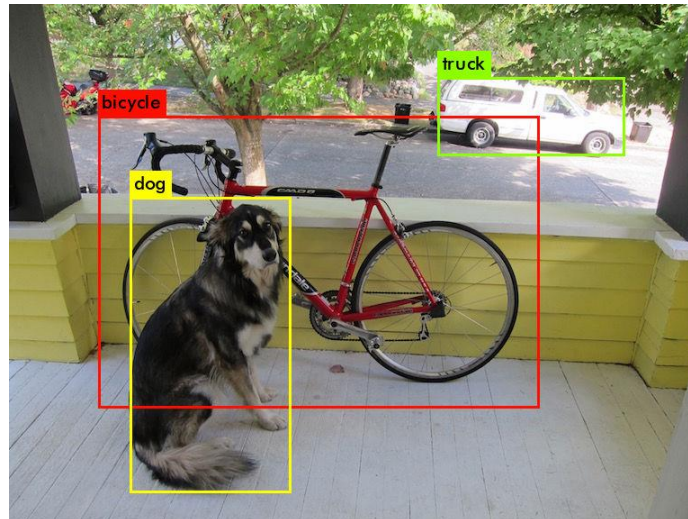
```
./darknet detector test cfg/coco.data cfg/yolov3.cfg yolov3.weights
data/dog.jpg
```

This command will give a response on the terminal that looks something like this.

```
77 conv 512 1 x 1 / 1 19 x 19 x1024 -> 19 x 19 x 512 0.379 BFLOPs
78 conv 1024 3 x 3 / 1 19 x 19 x 512 -> 19 x 19 x1024 3.407 BFLOPs
79 conv 512 1 x 1 / 1 19 x 19 x1024 -> 19 x 19 x 512 0.379 BFLOPs
80 conv 1024 3 x 3 / 1 19 x 19 x 512 -> 19 x 19 x1024 3.407 BFLOPs
81 conv 255 1 x 1 / 1 19 x 19 x1024 -> 19 x 19 x 255 0.189 BFLOPs
82 yolo
83 route 79
84 conv 256 1 x 1 / 1 19 x 19 x 512 -> 19 x 19 x 256 0.095 BFLOPs
85 upsample 2x 19 x 19 x 256 -> 38 x 38 x 256
86 route 85 61
87 conv 256 1 x 1 / 1 38 x 38 x 768 -> 38 x 38 x 256 0.568 BFLOPs
88 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFLOPs
89 conv 256 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 256 0.379 BFLOPs
90 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFLOPs
91 conv 256 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 256 0.379 BFLOPs
92 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFLOPs
93 conv 255 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 255 0.377 BFLOPs
94 yolo
95 route 91
96 conv 128 1 x 1 / 1 38 x 38 x 256 -> 38 x 38 x 128 0.095 BFLOPs
97 upsample 2x 38 x 38 x 128 -> 76 x 76 x 128
98 route 97 36
99 conv 128 1 x 1 / 1 76 x 76 x 384 -> 76 x 76 x 128 0.568 BFLOPs
100 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFLOPs
101 conv 128 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 128 0.379 BFLOPs
102 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFLOPs
103 conv 128 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 128 0.379 BFLOPs
104 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFLOPs
105 conv 255 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 255 0.754 BFLOPs
106 yolo
Loading weights from /Users/praveenpavithran/Downloads/yolo-object-detection/yolo-coco/yolov3.weights...Done!
data/dog.jpg: Predicted in 33.113836 seconds.
bicycle: 99%
truck: 92%
dog: 100%
```

Output with YOLOv3

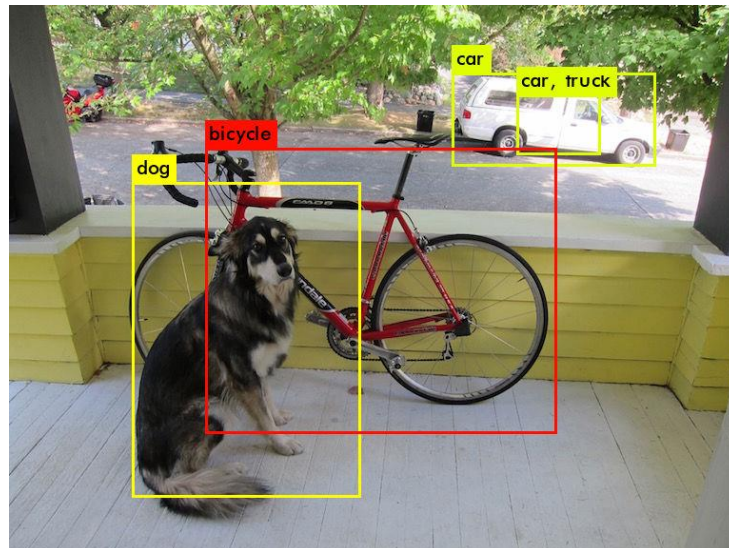
The output can be seen as a picture stored as predictions.jpg.



We can run inference on the same picture with yolo-tiny a smaller, faster but slightly less accurate model. The outputs look like these

```

layer    filters  size      input          output          BFLOPs
0 conv    16   3 x 3 / 1   416 x 416 x 3   -> 416 x 416 x 16  0.150 BFLOPs
1 max      2   2 x 2 / 2   416 x 416 x 16   -> 208 x 208 x 16
2 conv    32   3 x 3 / 1   208 x 208 x 16   -> 208 x 208 x 32  0.399 BFLOPs
3 max      2   2 x 2 / 2   208 x 208 x 32   -> 104 x 104 x 32
4 conv    64   3 x 3 / 1   104 x 104 x 32   -> 104 x 104 x 64  0.399 BFLOPs
5 max      2   2 x 2 / 2   104 x 104 x 64   -> 52 x 52 x 64
6 conv   128   3 x 3 / 1    52 x 52 x 64    -> 52 x 52 x 128  0.399 BFLOPs
7 max      2   2 x 2 / 2    52 x 52 x 128   -> 26 x 26 x 128
8 conv   256   3 x 3 / 1    26 x 26 x 128   -> 26 x 26 x 256  0.399 BFLOPs
9 max      2   2 x 2 / 2    26 x 26 x 256   -> 13 x 13 x 256
10 conv  512   3 x 3 / 1    13 x 13 x 256   -> 13 x 13 x 512  0.399 BFLOPs
11 max      2   2 x 2 / 1    13 x 13 x 512   -> 13 x 13 x 512
12 conv  1024   3 x 3 / 1    13 x 13 x 512   -> 13 x 13 x1024  1.595 BFLOPs
13 conv   256   1 x 1 / 1    13 x 13 x1024   -> 13 x 13 x 256  0.089 BFLOPs
14 conv   512   3 x 3 / 1    13 x 13 x 256   -> 13 x 13 x 512  0.399 BFLOPs
15 conv   255   1 x 1 / 1    13 x 13 x 512   -> 13 x 13 x 255  0.044 BFLOPs
16 yolo
17 route   13
18 conv   128   1 x 1 / 1    13 x 13 x 256   -> 13 x 13 x 128  0.011 BFLOPs
19 upsample 2x    13 x 13 x 128   -> 26 x 26 x 128
20 route  19 8
21 conv   256   3 x 3 / 1    26 x 26 x 384   -> 26 x 26 x 256  1.196 BFLOPs
22 conv   255   1 x 1 / 1    26 x 26 x 256   -> 26 x 26 x 255  0.088 BFLOPs
23 yolo
Loading weights from yolov3-tiny.weights...Done!
data/dog.jpg: Predicted in 1.342914 seconds.
dog: 57%
car: 52%
truck: 56%
car: 62%
bicycle: 59%
```



Comparing the results of yolov3 and yolo-tiny, we can see that yolo-tiny is much faster but less accurate. In this particular case it took a fraction of the time, but inferred a car/truck combo. Depending on your application you can choose a models that are faster or are more accurate.

Training a custom YOLO model

We will use YOLOX for this training. It is the latest version of YOLO models, pushing the limit in terms of speed and accuracy. YOLOX most recently won Streaming Perception Challenge

Install YOLOX Dependencies

To setup our development environment , we will first clone the base YOLOX repository and download the necessary requirements:

```
!git clone https://github.com/roboflow-ai/YOLOX.git %cd YOLOX
!pip3 install -U pip && pip3 install -r requirements.txt !pip3 install -v -e
!pip uninstall -y torch torchvision torchaudio
!pip install torch==1.7.1+cu110 torchvision==0.8.2+cu110 torchaudio==0.7.2 -f
https://download.pytorch.org/whl/torch_stable.html
```

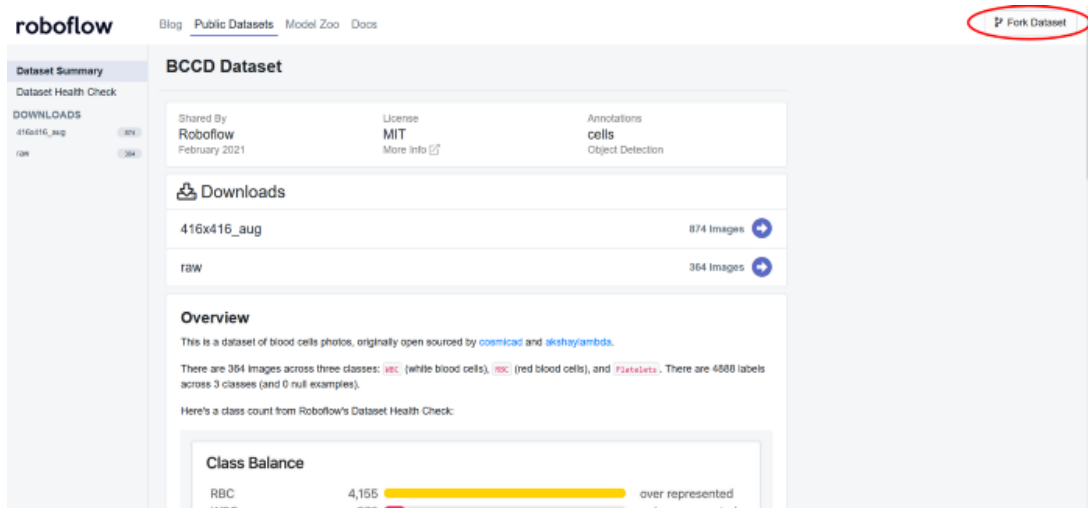
We will also install NVIDIA Apex and PyCocoTools to make this repository work as intended:

```
%cd /content/ !git clone https://github.com/NVIDIA/apex %cd apex
!pip install -v --disable-pip-version-check --no-cache-dir --global-option="--
cpp_ext" --global-option="--cuda_ext" ./ !pip3 install cython;
pip3 install
'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
```

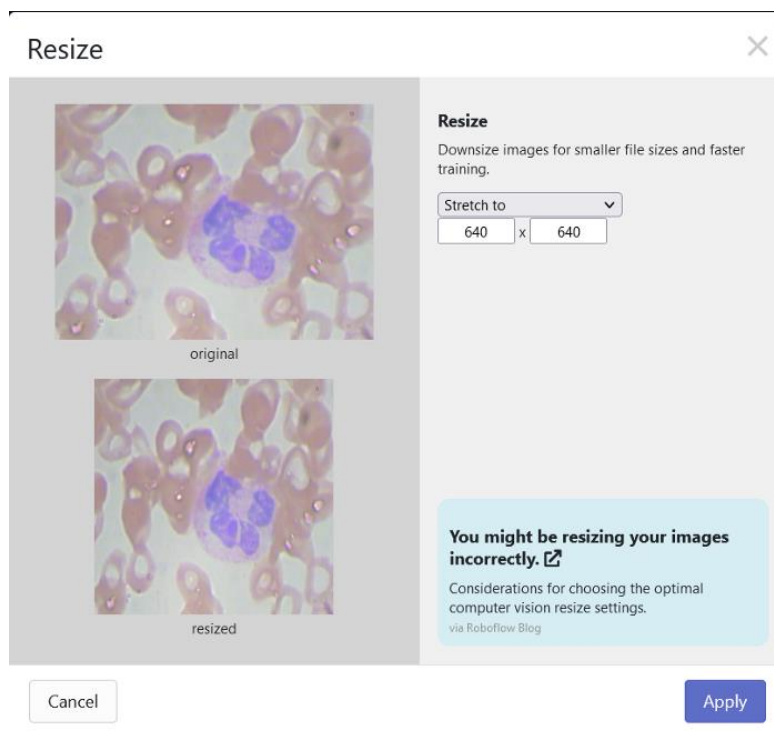
Download Custom YOLOX Object Detection Data

Before, we get started, you will want to create a Roboflow account. We will be using this blood cells dataset but you are welcome to use any dataset whether it be your own dataset loaded into Roboflow or another public dataset.

For this notebook, we will need to apply some preprocessing steps to ensure that the data will work with YOLOX. To get started, create a Roboflow account if you haven't already and fork the dataset:






After forking the dataset, you will want to add one preprocessing step which would be to resize all of the images to a size of 640 x 640:




Then simply generate a new version of the dataset and export with a “**Pascal VOC**”. You will receive a Jupyter notebook command that looks something like this:

Your Download Code

 Jupyter  Terminal  Raw URL

Paste this snippet into [a notebook from our model library](#) to download and unzip [your dataset](#):

```
!curl -L "https://app.roboflow.com/ds/[REDACTED]?key=[REDACTED]" > roboflow.zip;
unzip roboflow.zip; rm roboflow.zip
```

 **Warning:** Do not share this link beyond your team, it contains a private key that is tied to your Roboflow account. Acceptable use policy applies.

Done

Copy the command, and replace the line below in the notebook with the command provided by Roboflow:

```
!curl -L "[YOUR LINK HERE]" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

Labeling Your Data

If you are bringing your own dataset, you can annotate your images in Roboflow.

Download Pre-Trained Weights for YOLOX

YOLOX comes with some pretrained weights to allow the model to train faster and achieve higher accuracies. There are many sizes of weights but the size of weights we use will be based on the small YOLOX model (YOLOX_S). We can download this as follows:

```
%cd /content/ !wget https://github.com/Megvii-BaseDetection/storage/releases/download/0.0.1/yolox_s.pth %cd /content/YOLOX/
```

Run YOLOX training

To train the model, we can run the `tools/train.py` file:

```
!python tools/train.py -f exps/example/yolox_voc/yolox_voc_s.py -d 1 -b 16 --fp16 -o -c /content/yolox_s.pth
```

The arguments for running this command include:

- Experience File: This file allows us to change certain aspects of the base model to apply when training
- Devices: The number of GPUs our model will train with — 1 is the value as Colab provides 1
- Batch Size: Number of image in each batch
- Pretrained Weights: Specify the path to the weights you want to use — this can be weights we downloaded or an earlier checkpoint of your model

After about 90 epochs of training, we get the following APs.

```
100%|#####| 7/7 [00:03<00:00, 1.92it/s]
2021-07-31 00:29:43 | INFO | yolox.evaluators.voc_evaluator:161 - Evaluate in main process...
Writing rbc VOC results file
Writing wbc VOC results file
Writing platelets VOC results file
Eval IoU : 0.50
AP for rbc = 0.8801
AP for wbc = 0.9964
AP for platelets = 0.9004
Mean AP = 0.9257
~~~~~
Results:
0.880
0.996
0.900
0.926
~~~~~
```

Evaluate YOLOX performance

To evaluate YOLOX performance we can use the following command:

```
MODEL_PATH = "/content/YOLOX/YOLOX_outputs/yolox_voc_s/latest_ckpt.pth.tar"
!python3 tools/eval.py -n yolox-s -c {MODEL_PATH} -b 64 -d 1 --conf 0.001 -f
exps/example/yolox_voc/yolox_voc_s.py
```

After running the evaluation, we get the following results:

```
AP for rbc = 0.8802
AP for wbc = 0.9964
AP for platelets = 0.9895
Mean AP = 0.9554
*****
Results:
0.880
0.996
0.989
0.955
*****

Results computed with the **unofficial** Python eval code.
Results should be very close to the official MATLAB eval code.
Recompute with './tools/eval.py --matlab ...' for your paper.
-- Thanks, The Management

Eval IoU : 0.55
Eval IoU : 0.60
Eval IoU : 0.65
Eval IoU : 0.70
Eval IoU : 0.75
Eval IoU : 0.80
Eval IoU : 0.85
Eval IoU : 0.90
Eval IoU : 0.95

map_5095: 0.6771649596026756
map_50: 0.955363353221803
*****
```

Evaluation for YOLOX Model

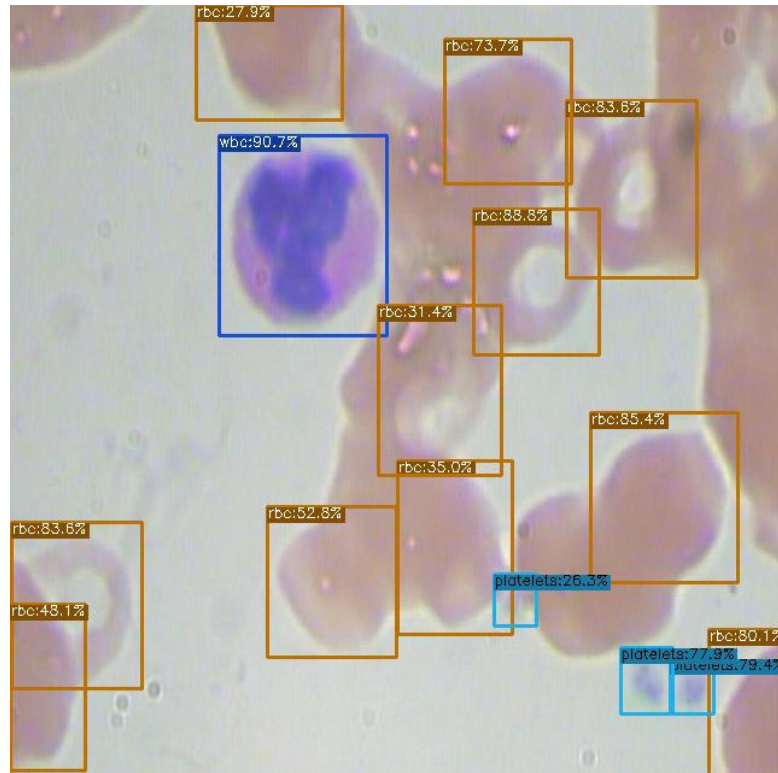
Run YOLOX Inference on Test Images

We can now run YOLOX on a test image and visualize the predictions. To run YOLOX on a test image:

```
TEST_IMAGE_PATH =
"/content/valid/BloodImage_00057_jpg.rf.1ee93e9ec4d76cfaddaa7df70456c376.jpg"
!python tools/demo.py image -f
/content/YOLOX/exps/example/yolox_voc/yolox_voc_s.py -c {MODEL_PATH} --path
{TEST_IMAGE_PATH} --conf 0.25 --nms 0.45 --tsize 640 --save_result --device gpu
```

To visualize the predictions on the image:

```
from PIL import Image OUTPUT_IMAGE_PATH =
"/content/YOLOX/YOLOX_outputs/yolox_voc_s/vis_res/2021_07_31_00_31_01/BloodImag
e_00057_jpg.rf.1ee93e9ec4d76cfaddaa7df70456c376.jpg"
Image.open(OUTPUT_IMAGE_PATH)
```



Looks like the model works as intended!

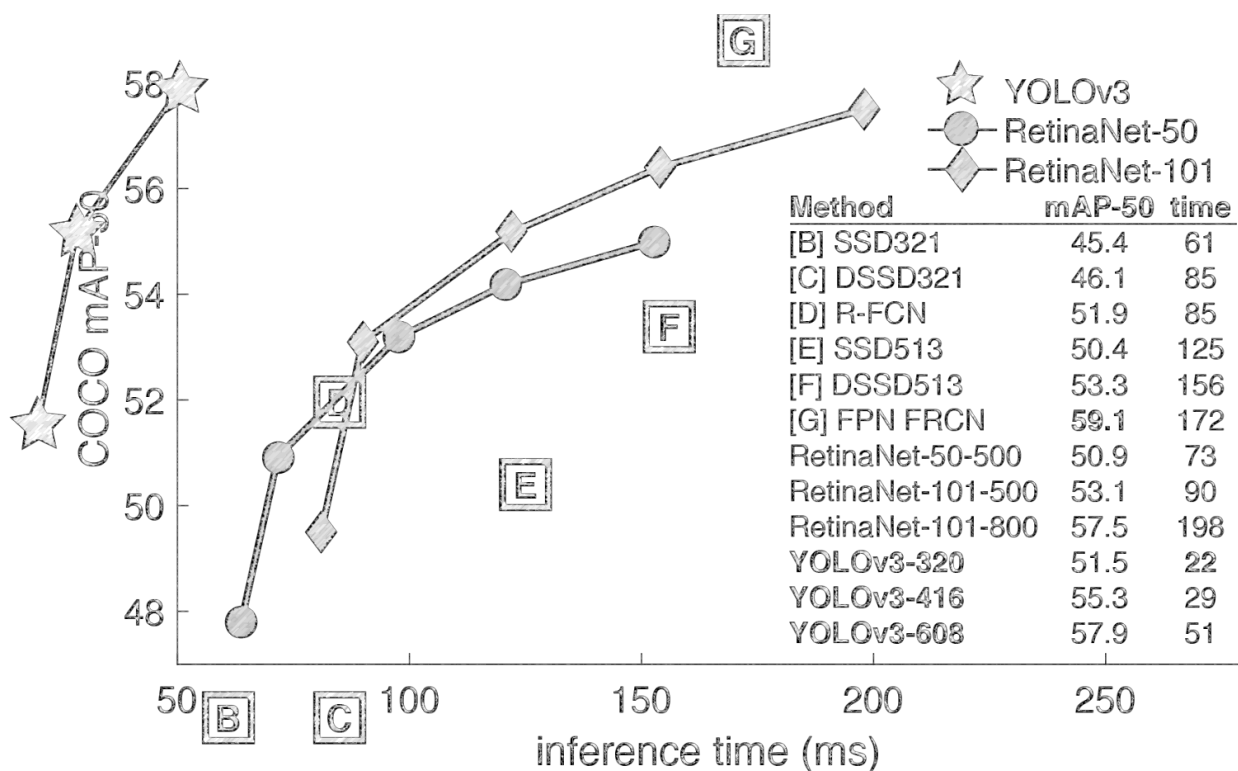
Export saved YOLOX weights for future inference

Finally we can export the model into our Google Drive as follows:

```
from google.colab import drive
drive.mount('/content/gdrive')
%cp {MODEL_PATH} /content/gdrive/My\ Drive
```

Comparison to Other Detectors

YOLOv3 is extremely fast and accurate. In mAP measured at .5 IOU YOLOv3 is on par with Focal Loss but about 4x faster. Moreover, you can easily tradeoff between speed and accuracy simply by changing the size of the model, no retraining required!



Performance on the COCO Dataset

Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20

Summary

- YOLO actually looks at the image just once (hence its name: You Only Look Once) but in a clever way.
- YOLO's unified architecture is extremely fast. base YOLO model processes images in real-time at 45 frames per second.
- YOLOX is the latest version of YOLO models, pushing the limit in terms of speed and accuracy.
- YOLOX comes with some pretrained weights to allow the model to train faster and achieve higher accuracies.