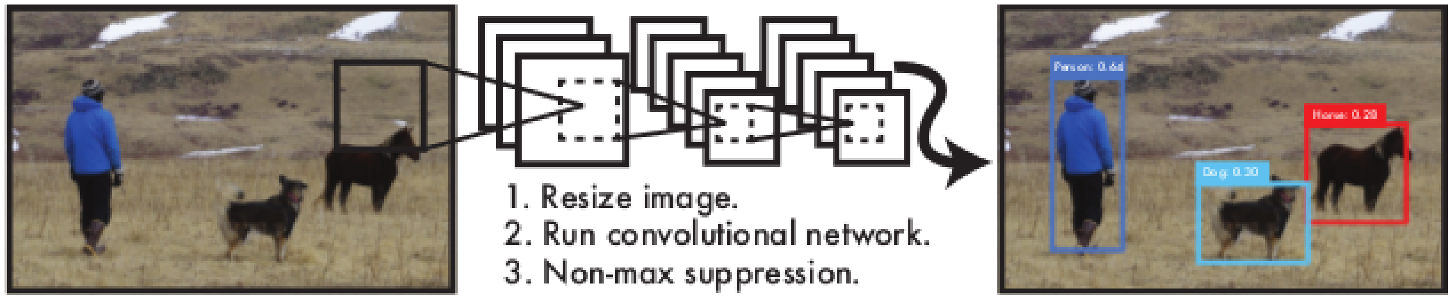
You Only Look Once: Unified, Real-Time Object Detection

https://arxiv.org/pdf/1506.02640v5.pdf

Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi
University of Washington, Allen Institute for AI,Facebook AI Research http://pjreddie.com/yolo/



1. A single neural network predicts bounding boxes and class probabilities. 2. Base YOLO model runs at 45 FPS. A smaller version of the network, Fast YOLO, runs astounding 155 FPS second, outperforms DPM (deformable parts models) and R-CNN.

Figure 1: The YOLO Detection System. (1) resizes image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the result by the model's confidence.

Tutorial: https://pjreddie.com/darknet/yolo/

GitHub, Inc. (US)

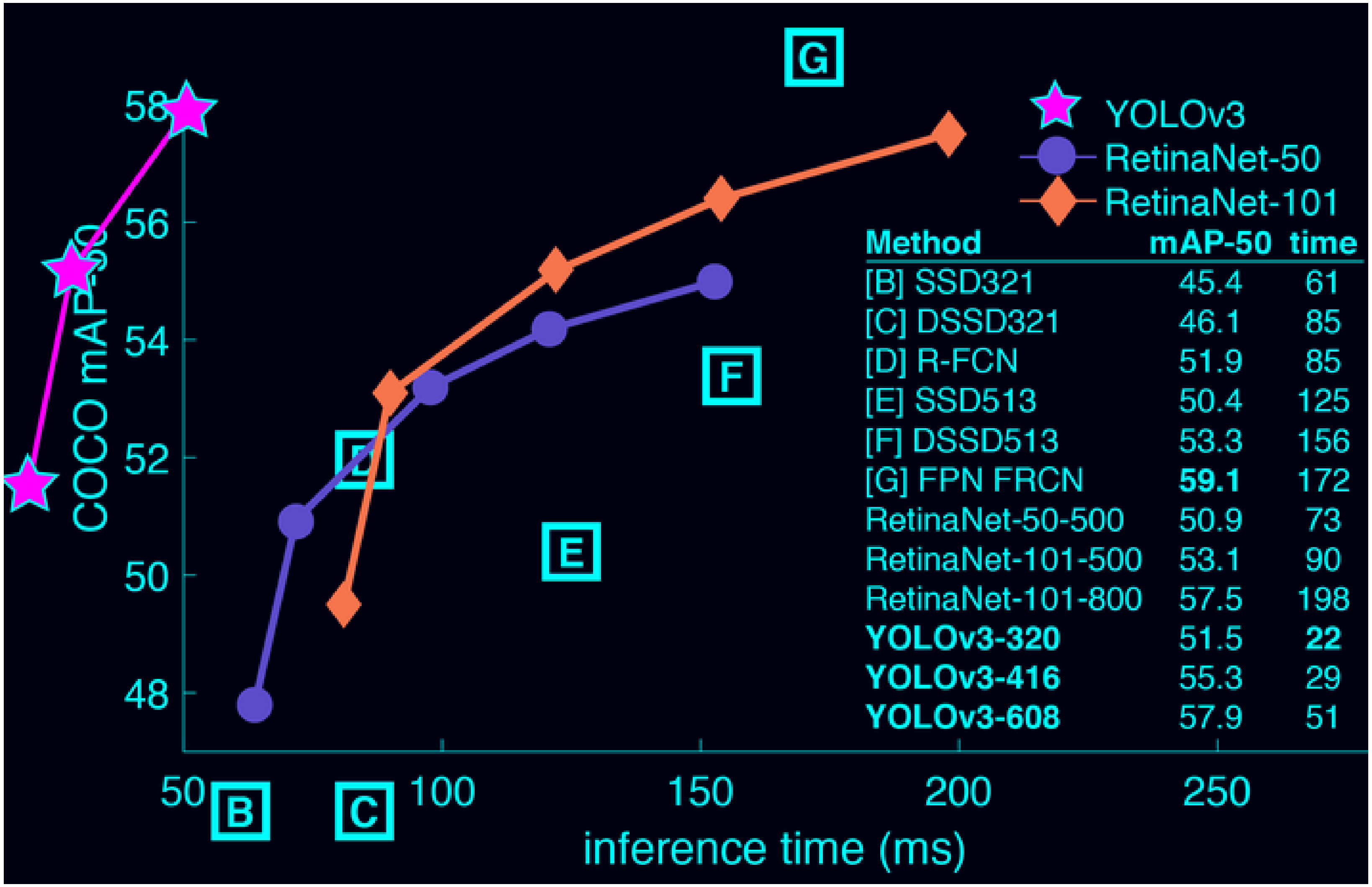
```
git clone https://github.com/pjreddie/darknet
```

YOLO

https://pjreddie.com/darknet/yolo/

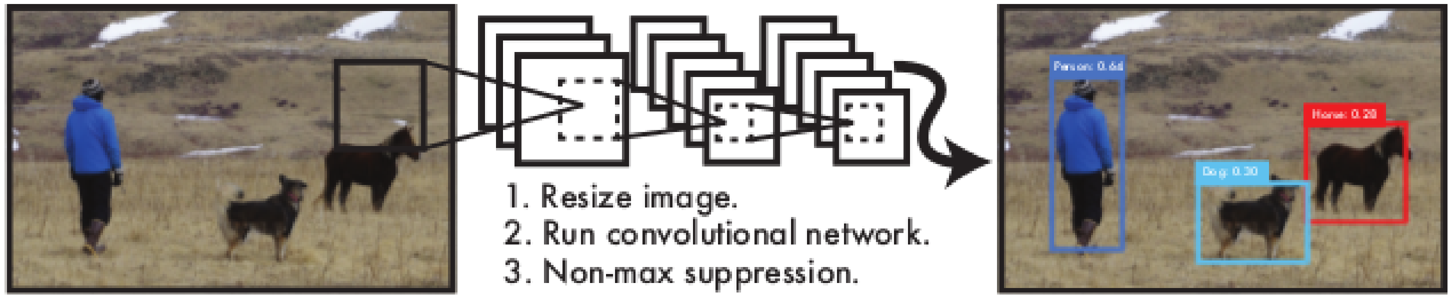
YouTube https://www.youtube.com/watch?time_continue=42&v=MPU2HistivI

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



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

YOLO Model



1. Object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities.

A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.

YOLO Formulation (1)

1. Yolo divides I(x,y) into an S × S grid G(x,y).

2. If the center of an object O_i(x,y), for i = 1, 2, ... , K, falls into a grid G(x,y), then that G(x,y) is responsible for detecting objects O_i(x,y).

3. Each grid G(x,y) predicts bounding boxes B_j(x,y). To detect objects O_i(x,y), G(x,y) places bounding box B_j(x,y), for j = 1, 2, ... , M. Each B_j(x,y) consists of 5 predictions: {x, y, w, h, f(B_j(x,y)}, where f(B_j(x,y)) is defined as confidence.

4. Define confidence

f(B_j(x,y)) = Prob(O_i(x,y)) * (IOU_truth)^pred ... (1)

where IOU is Intersection Over Union. Calculate the confidence for B_j(x,y):

f(B_j(x,y)) = 0 if O_i(x,y) is null (no object).
equal to IOU between the predicted box and the ground truth. ... (2)

5. Each grid G(x,y) also predicts C_i, i = 1, 2, ..., N, define conditional class probabilities,

Prob(C_i |O_j(x,y)) … (3)

These probabilities are conditioned on the grid cell G(x,y) containing an object O_j(x,y).

YOLO Formulation (2)

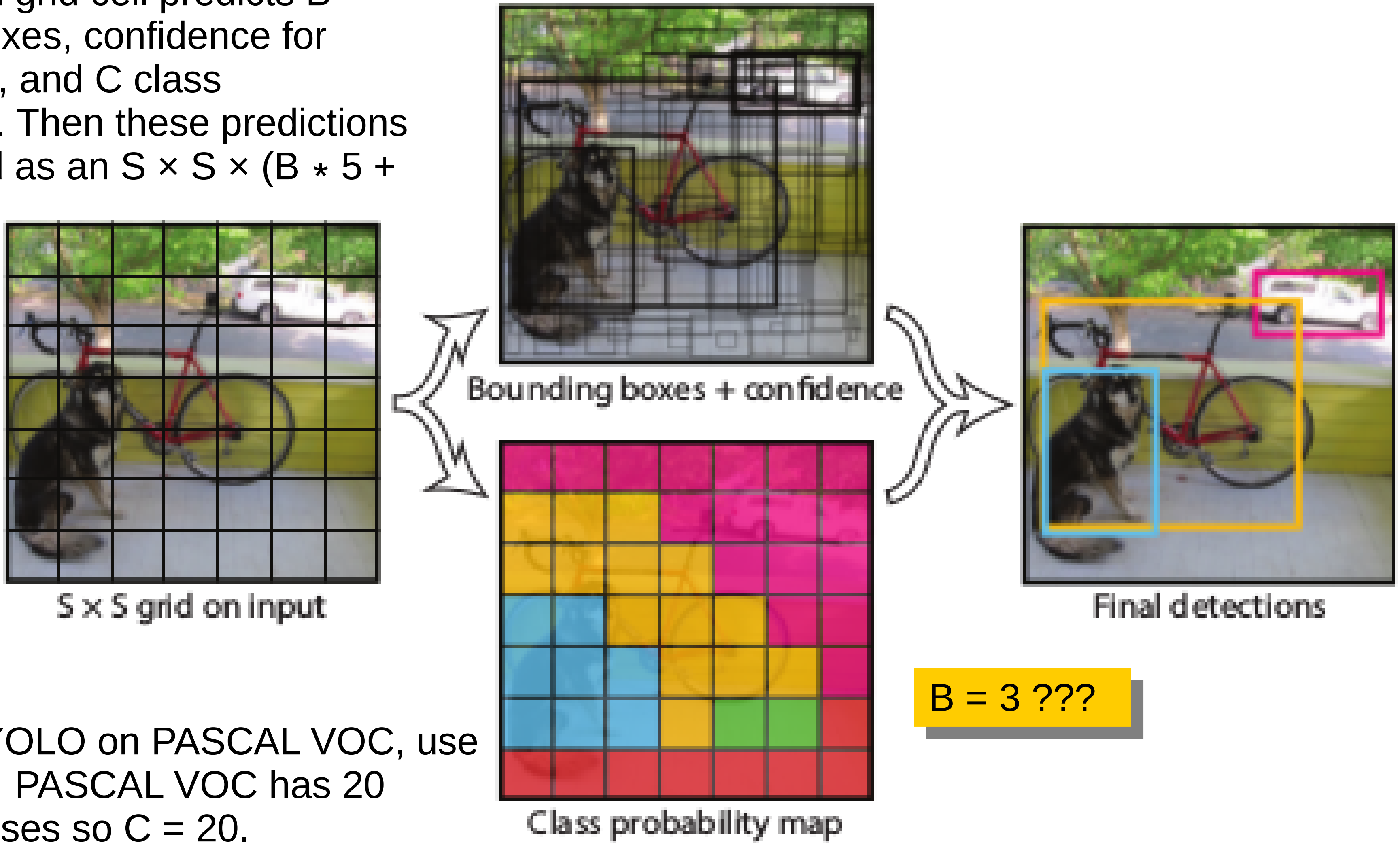
We only predict one set of class probabilities per grid cell G(x,y), regardless of the number of boxes B.
At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\boxed{\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}} \quad \ldots (4)$$

class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

Example YOLO Formulation (3)

Divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. Then these predictions are encoded as an S × S × (B ∗ 5 + C) tensor.



B = 3 ???

Evaluating YOLO on PASCAL VOC, use S = 7, B = 2. PASCAL VOC has 20 labelled classes so C = 20.
Our final prediction is a 7 × 7 × 30 tensor.

Standard Data Set Pascal VOC

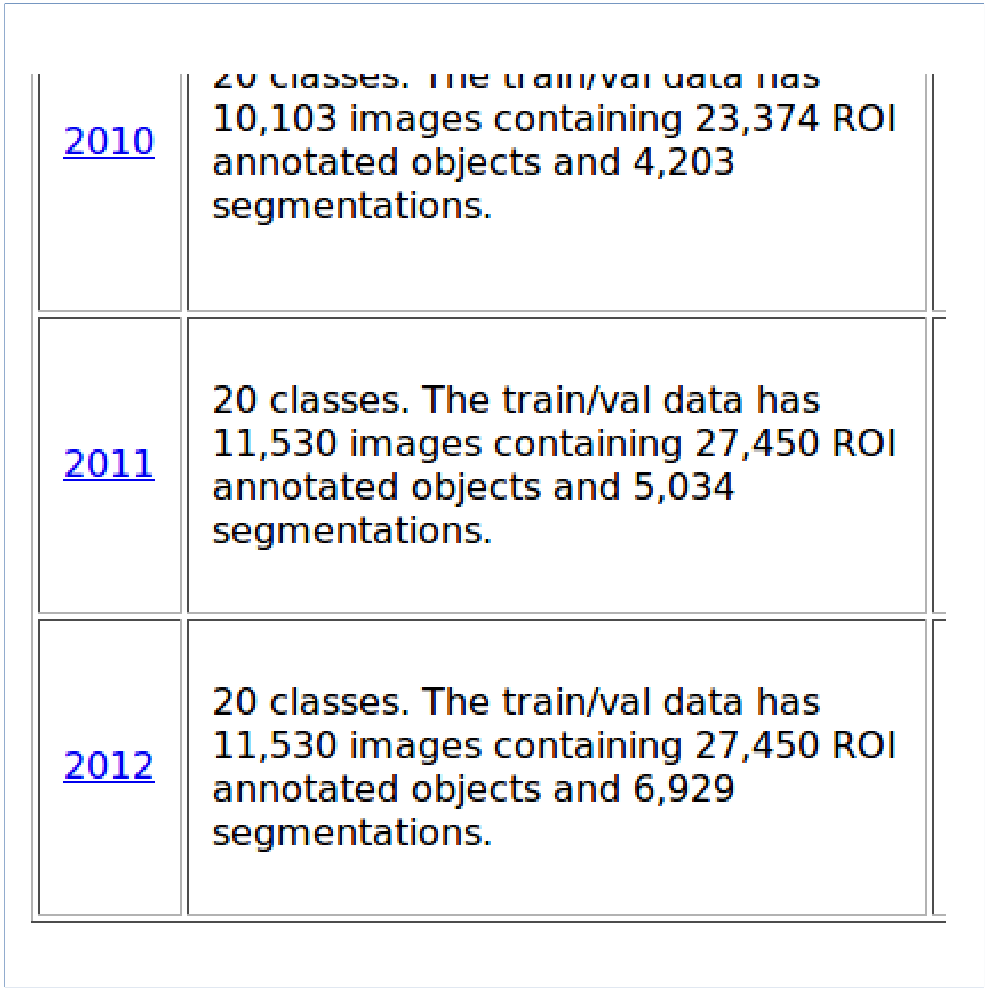
http://host.robots.ox.ac.uk/pascal/VOC/

PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning

The PASCAL VOC project:

1. Provides standardised image data sets for object class recognition
2. Provides a common tools for accessing the data sets and annotations
3. Enables evaluation and comparison of different methods
4. Ran challenges evaluating performance on class recognition (2005-12)

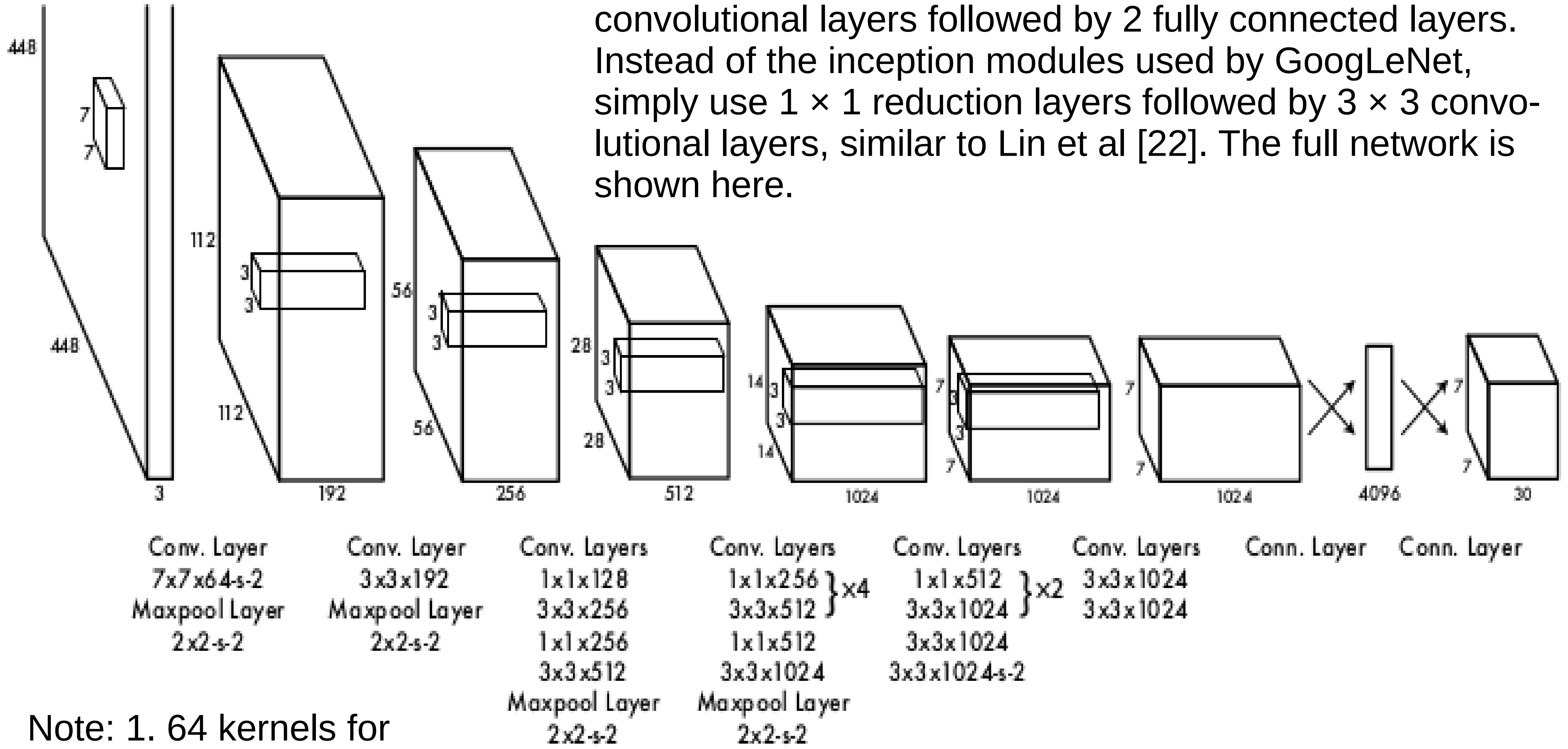
Example:



2010	20 classes. The train/val data has 10,103 images containing 23,374 ROI annotated objects and 4,203 segmentations.
2011	20 classes. The train/val data has 11,530 images containing 27,450 ROI annotated objects and 5,034 segmentations.
2012	20 classes. The train/val data has 11,530 images containing 27,450 ROI annotated objects and 6,929 segmentations.

YOLO Network Architecture

Inspired by the GoogLeNet. Our network has 24 convolutional layers followed by 2 fully connected layers. Instead of the inception modules used by GoogLeNet, simply use 1 × 1 reduction layers followed by 3 × 3 convolutional layers, similar to Lin et al [22]. The full network is shown here.



Note: 1. 64 kernels for convolution, so produces 64 outputs each with 3 channels, 64*3 = 192;
2. Maxpool 2x2, so 448/2 = 224, and stride = 2, so 224/2 = 112

YOLO Loss Function

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

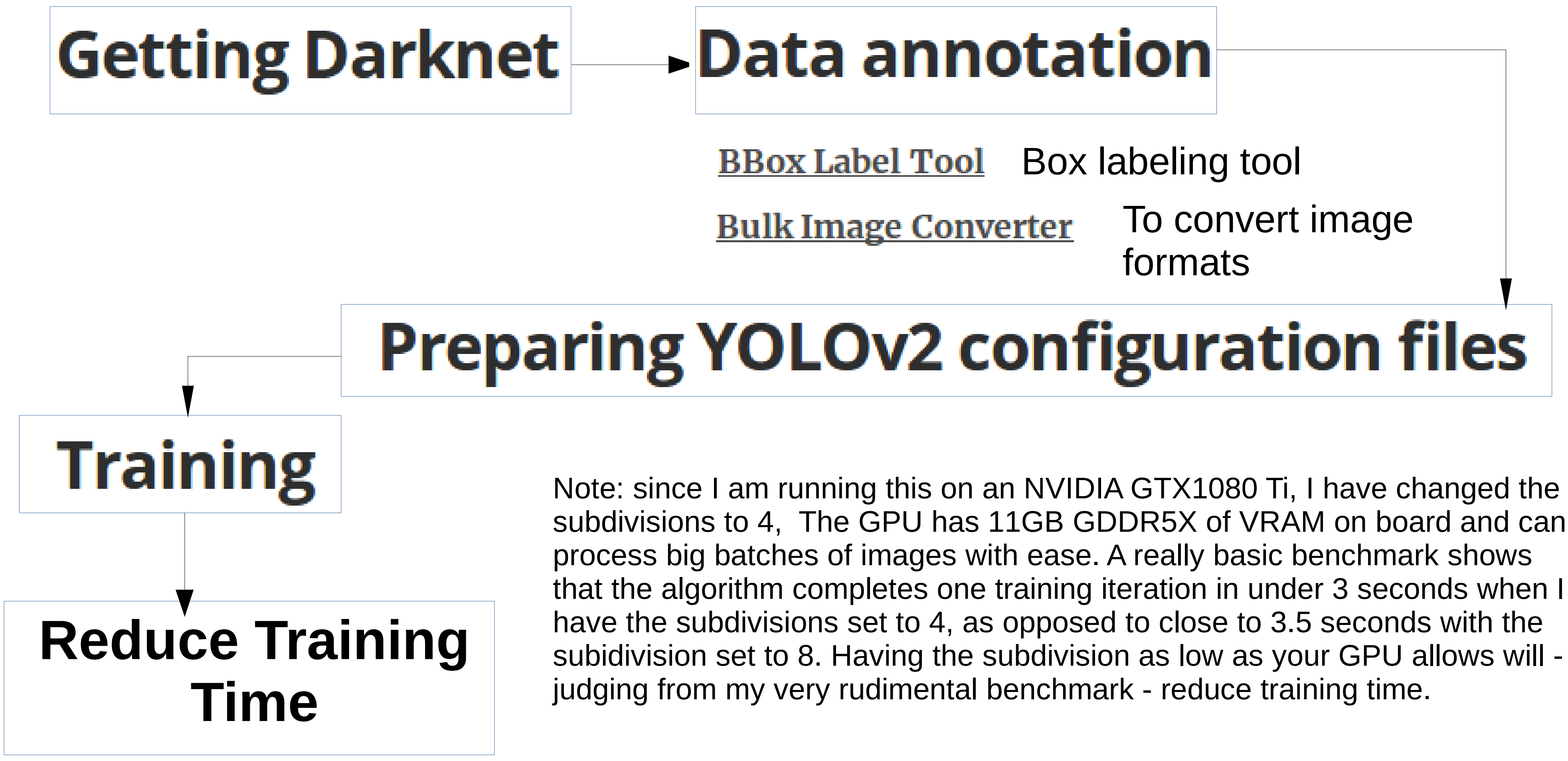
YOLO Training

Pretrain our convolutional layers on the ImageNet 1000-class competition dataset [30]. For pretraining we use the first 20 convolutional layers followed by a average-pooling layer and a fully connected layer. We train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe’s Model Zoo [24]. We use the Darknet framework for all training and inference [26].

YOLO Training

https://timebutt.github.io/static/how-to-train-yolov2-to-detect-custom-objects/

How to train YOLOv2 to custom objects



Note: since I am running this on an NVIDIA GTX1080 Ti, I have changed the subdivisions to 4, The GPU has 11GB GDDR5X of VRAM on board and can process big batches of images with ease. A really basic benchmark shows that the algorithm completes one training iteration in under 3 seconds when I have the subdivisions set to 4, as opposed to close to 3.5 seconds with the subidivision set to 8. Having the subdivision as low as your GPU allows will - judging from my very rudimental benchmark - reduce training time.

April 22 2019 yolo.py (1)

Yolo Source code:
https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

Yolo, training and data annotation:
https://timebutt.github.io/static/how-to-train-yolov2-to-detect-custom-objects/

Step 1. Down load the yolo py code from the reference link at left.

1

```
# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([args["yolo"], "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")
```

2

```
# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])
```

3

```
# load our YOLO object detector trained on COCO dataset (80 classes)
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
```

April 22 2019 yolo.py (2)

4

```
# load our input image and grab its spatial dimensions
image = cv2.imread(args["image"])
(H, W) = image.shape[:2]
```

April 22 2019 yolo.py

Yolo Source code:
https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

Yolo, training and data annotation:
https://timebutt.github.io/static/how-to-train-yolov2-to-detect-custom-objects/

Step 1. Down load the yolo py code from the reference link at left.

1

```
# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([args["yolo"], "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")
```

2

```
# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])
```

```
# load our YOLO object detector trained on COCO dataset (80 classes)
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

# load our input image and grab its spatial dimensions
```