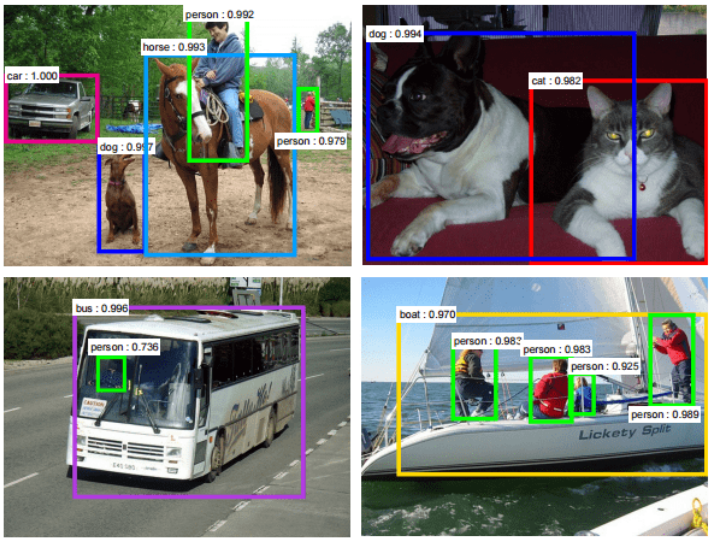
EE 046746 - Technion - Computer Vision

Tal Daniel

Tutorial 04 - Deep Object Detection



- Image Source

Agenda

In [1]:
```
# imports for the tutorial
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import cv2
```

```
import torch
from models.yolov3.detect import yolov3_detect
```

What Is Object Detection/Recognition?

- In this tutorial we are going to get familiar with one of the most popular and challenging tasks in Computer Vision - Object Detection (or Object Recognition).
- It is sometimes hard to distinguish between different related computer vision tasks. For example, classification vs. localization, detection vs. tracking and etc...

Image Classifcation + Object Localization = Object Detction

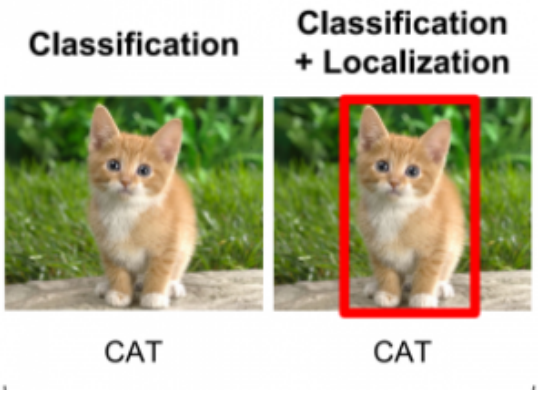
- **Image classification** takes an image and predicts the object in an image. For example, when we build a cat-dog classifier, we take images of cat or dog and predict their class:

- *Input*: An image with a single object, such as a photograph.
- *Output*: A class label (e.g. one or more integers that are mapped to class labels).

- But what if both classes are present in the image? What would be the model's prediction?

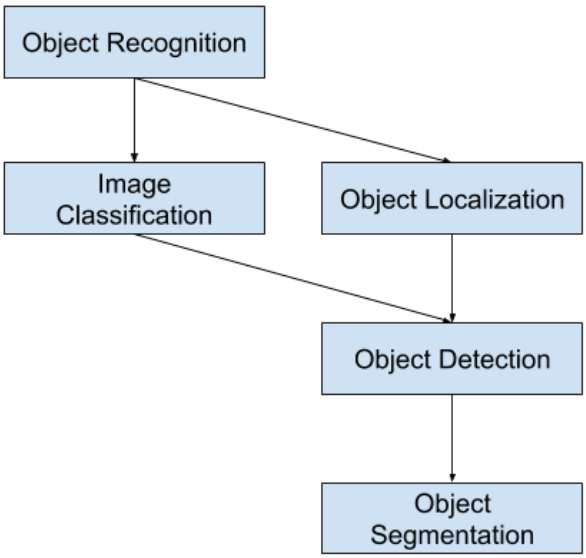
- To solve this problem we can train a multi-label classifier which will predict all the possible classes (dog, cat, dog-and-cat).
- **Object localization** - we still don't know the *location* of the cat or the dog. The problem of identifying the location of an object (given the class) in an image is called **localization**.

- If the object class is not known, we have to predict both the location and the class of each object!



- *Input*: An image with one or more objects, such as a photograph.
- *Output*: One or more bounding boxes (e.g. defined by a point, width, and height).

- **Object detection** - predicting the location of the object along with the class is called object detection.
- Instead of predicting the class of object from an image, we now have to predict the class as well as a rectangle (called *bounding box*) containing that object.
- So, image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image.
- Object detection is more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all of these problems are referred to as object recognition.



- It takes 4 variables to uniquely identify a rectangle. So, for each instance of the object in the image, we shall predict following variables:
  - `class_name`
  - `bounding_box_top_left_x_coordinate`
  - `bounding_box_top_left_y_coordinate`
  - `bounding_box_width`
  - `bounding_box_height`

Localization Approaches

- **Classic Sliding Window** approach
- **Region-Based Convolutional Neural Networks**, or R-CNNs - a family of techniques for addressing object localization and recognition tasks, designed for model performance. Region proposal + classification approach.
- **You Only Look Once**, or YOLO, is a second family of techniques for object recognition designed for speed and real-time use. Regression approach.

Sliding Window

- Object Detection is modeled as a classification problem where we take windows of fixed sizes from input image at all the possible locations feed these patches to an image classifier.

- Each window is fed to the classifier which predicts the class of the object in the window (or background if none is present).
- **Problem** - how do you know the size of the window so that it always contains the image?

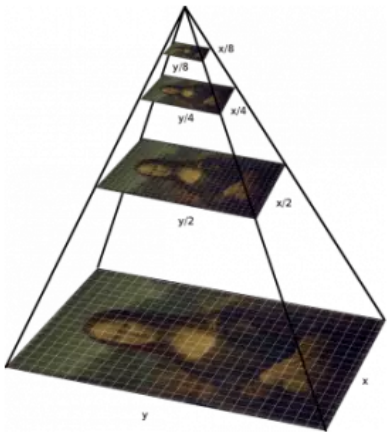
Small sized object

Big sized object. What size do you choose for your sliding window detector?

- As you can see that the object can be of varying sizes.
- To solve this problem an **image pyramid** is created by scaling the image.
  - The idea is to resize the image at multiple scales and rely on the fact that our chosen window size will completely contain the object in *one* of these resized images.

- Most commonly, the image is downsampled (size is reduced) until a certain condition, typically a minimum size, is reached.
- A fixed size window detector is run on each of these images.
- It's common to have as many as 64 levels on such pyramids. Now, all these windows are fed to a classifier to detect the object of interest.

- This approach can be very expensive computationally, and thus **very slow**.



Performance Metrics

- How can we tell if the *predicted* bounding box is good with respect to the *ground truth* (labeled) bounding box?
- Two popular evaluation metrics are the **intersection over union (IoU)** and **Average Precision (AP)**

Intersection over Union (IoU)

- Also called the *Jaccard Index*.
- A value between 0 and 1.
- It corresponds to the overlapping area between the predicted mask and the ground-truth mask.
- The **higher** the IoU, the better the predicted location of the box for a given object.
- The segmentation challenge is evaluated using the **mean Intersection over Union (mIoU)** metric.
  - The mIoU is the average between the IoU of the segmented objects over all the images of the test dataset.
-

$$IoU = \frac{TP}{TP + FP + FN} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}$$

  - X and Y are the predicted and ground truth segmentation, respectively.
  - TP is the true positives, FP false positives and FN false negatives.
- Typical threshold for detection: 0.5

- Image Source: Wikipedia

Average Precision (AP)

- **Precision** measures how accurate we are when we're saying something is true: $P = \frac{TP}{TP+FP}$.
- **Recall** measures how many times we missed a true prediction: $R = \frac{TP}{TP+FN}$.
- We can balance the metrics of models by changing our model's confidence threshold (e.g, 2 possible thresholds for detecting a cat are if the model gave it a score of >0.9, or 0.8. Different thresholds will gives us different recall and precision scores):
  - High precision and low recall: When we say something is true - we're sure of that! But we're going to miss a lot of of hard true examples.
  - Low precision and high recall: We will never say something is false when it's true! But now we aren't that sure of the prediction.
- By drawing the precision on a y-axis and the recall on the x-axis, for different thresholds, we can create the **Precision-recall curve**, that will give us insight on the tradeoff between the metrics.
- In binary classification, the Average Precision (AP) is calculating the area under the precision-recall curve, thus summarizing it.
  - Usually this calculation is done using 5/11 discrete points, and not on the continus precision-recall curve.
- The **mean Average Precision (mAP)** is the mean of the Average Precisions computed over *all the classes* of the challenge.
  - Let Q be the number of classes, and $AvgP(q)$ is the average precision of the q^{th} category:

$$mAP = \frac{\sum_{q=1}^{Q} AvgP(q)}{Q}$$

- The mAP metric avoids having extreme specialization in few classes and thus weak performances in others.

Region-based Convolutional Neural Networks (R-CNN) Family

- The problem with combining CNNs with the sliding window approach is that CNNs are too slow and computationally very expensive. It is practically impossible to run CNNs on so many patches

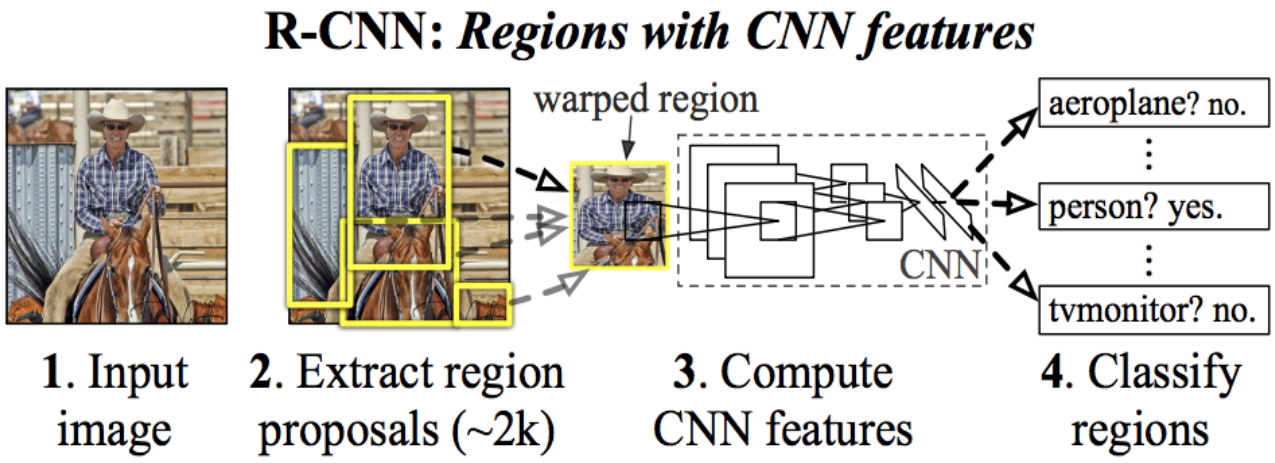
generated by a sliding window detector.

- The R-CNN family of methods refers to the R-CNN, which may stand for "Regions with CNN Features" or "Region-Based Convolutional Neural Network," developed by Ross Girshick, et al.
- This includes the techniques **R-CNN**, **Fast R-CNN**, and **Faster-RCNN** designed and demonstrated for object localization and object recognition.

R-CNN

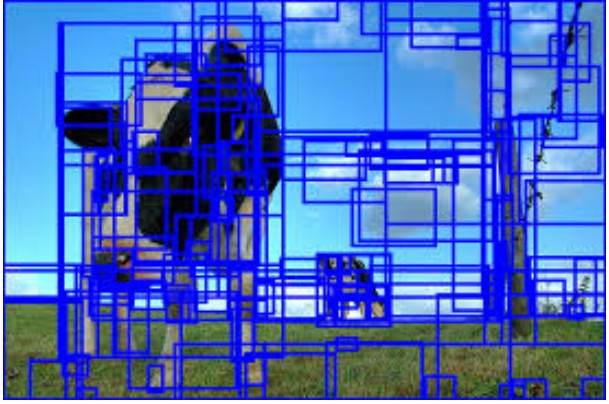
- The R-CNN was introduced in the 2014 paper by Ross Girshick, et al. from UC Berkeley titled "Rich feature hierarchies for accurate object detection and semantic segmentation.
- It may have been one of the first large and successful application of convolutional neural networks to the problem of object localization, detection, and segmentation.
- The approach was demonstrated on benchmark datasets, achieving then state-of-the-art results on the VOC-2012 dataset and the 200-class ILSVRC-2013 object detection dataset. We presented these datasets in the *Segmentation* tutorial.

- The R-CNN model is comprised of three modules:
  - **Module 1: Region Proposal** - generate and extract category independent region proposals, e.g. candidate bounding boxes.
  - **Module 2: Feature Extractor** - extract feature from each candidate region, e.g. using a deep convolutional neural network.
  - **Module 3: Classifier** - classify features as one of the known class, e.g. linear SVM classifier model.



Module 1 - Region Proposal

- **Selective Search** - a *classic* computer vision technique that is used to propose candidate regions or bounding boxes of potential objects in the image.
  - *Stage 1* - calculate initial regions.
  - *Stage 2* - group regions with the highest similarity - repeat.
  - *Stage 3* - generate a hierarchy of bounding boxes.
- Selective search uses local cues like texture, intensity, color and/or a measure of insideness and etc to generate all the possible locations of the object.



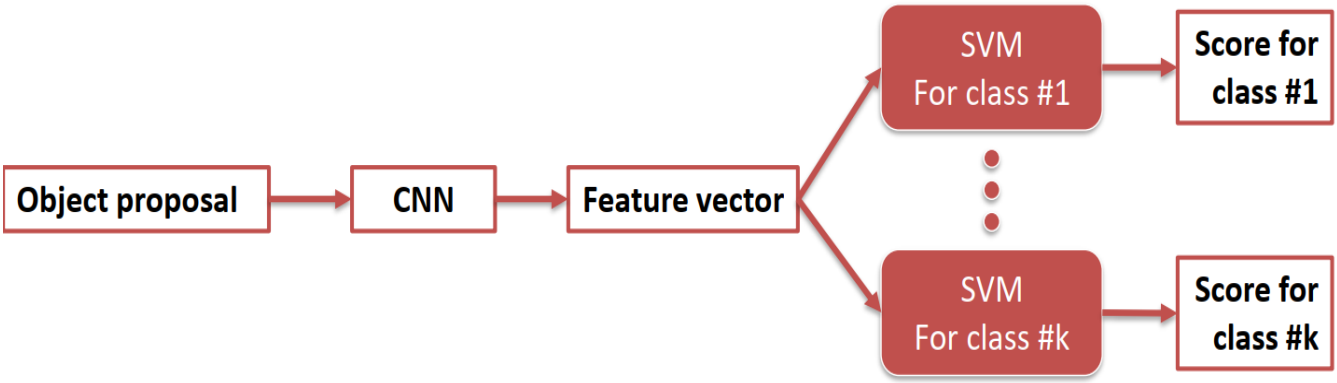
- Image Source

Module 2 - Feature Extractor

- A feature vector of 4,096 dimensions is extracted *for each* object proposal using a **CNN**.
- The feature extractor used by the model was the AlexNet deep CNN that won the ILSVRC-2012 image classification competition.
- **Problem** - the input of the CNN should be a fixed size (224x224) but the size of each proposal *is different*.
  - The size of the objects must be changed to a fixed size!
  - **Solution** - use *warping* - anisotropically scales the object proposals (different scale in two directions).
- The choice of the **CNN architecture** has a large effect on the detection performance (obviously...).

Module 3 - Classifier

- The output of the CNN was a 4,096 element vector that describes the contents of the image that is fed to a linear SVM for classification, specifically one SVM is trained for each known class.
- Why use SVM and not a *Softmax* layer? It *empirically* worked better.



Training R-CNNs

- **Training the CNN**:
  - R-CNNs uses pre-trained CNNs, usually on pre-trained image-level (not object level) annotations.
  - Adapting the CNN to detection and to the new domain (warped proposal windows), by training with SGD (Stochastic Gradient Descent).
    - Uses proposals with $IoU > 0.5$ as positive examples (the rest are negative). Recall that this is a supervised task, and thus we have the ground-truth bounding boxes.
- **Training the SVM**:
  - Uses proposals with $IoU < 0.3$ as negative examples and ground truth regions as positive examples.
    - Why is it different than the CNN threshold? It *empirically* works better and reduces *overfitting*.

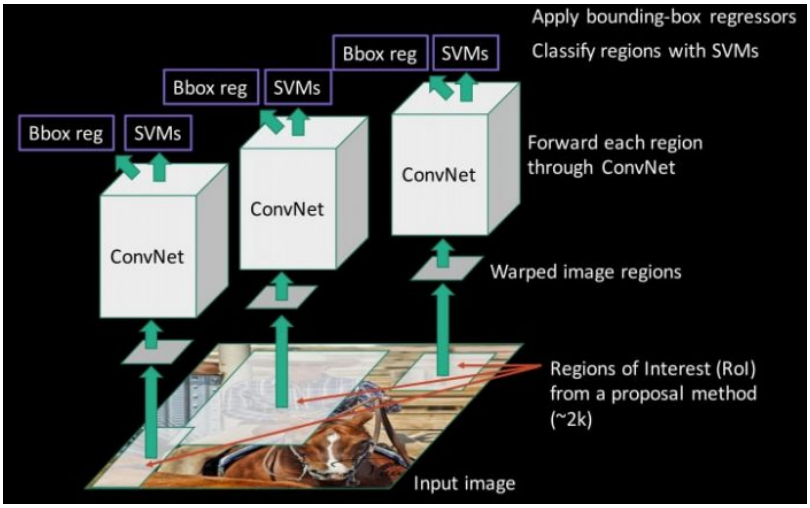
Detecting Objects with R-CNNs

- At test time, **non-maximum suppression (NMS)** is applied greedily given all scored regions in an image.
- **Non-Maximum Supression (NMS)** - a technique which filters the proposals based on some threshold value (rejects proposals).
  - *Input*: A list of Proposal boxes B, corresponding confidence scores S and overlap threshold N.
  - *Output*: A list of filtered proposals D.

- Source and Read More

Drawbacks of R-CNN

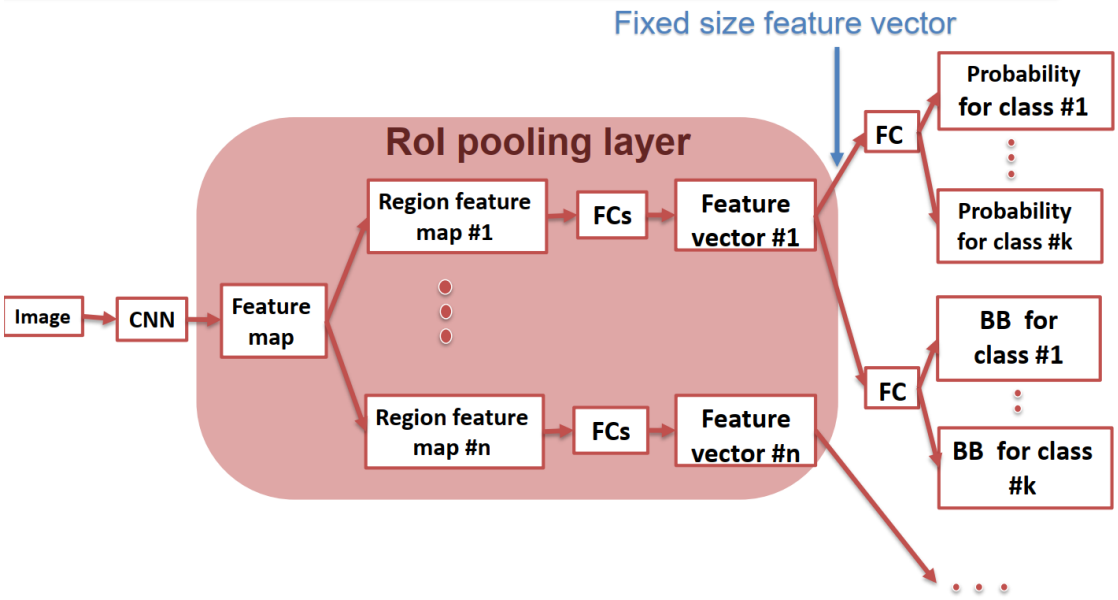
- Training is not end-to-end, but a *multi-stage* pipeline.
- Trainning is computationally expensive in space and time (training a deep CNN on so many region proposals per image is very slow).
- At test-time object-detection is slow, requiring a CNN-based feature extraction to pass on each of the candidate regions generated by the region proposal algorithm.

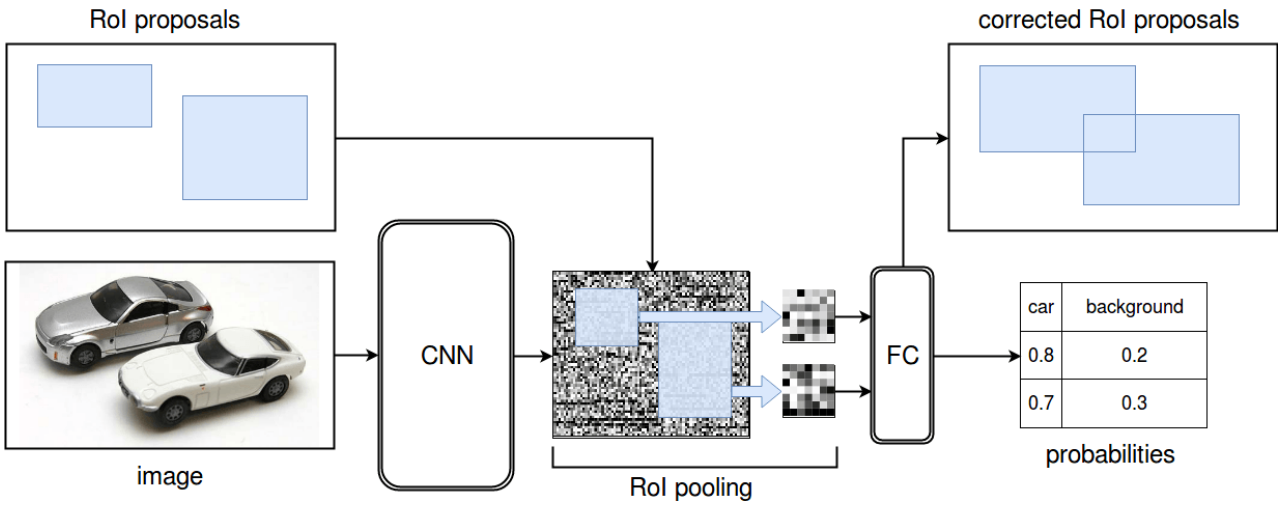


Fast R-CNNs

- Given the great success of R-CNN, Ross Girshick, proposed an extension to address the speed issues of R-CNN in a 2015 paper titled "Fast R-CNN".
- Fast R-CNN is proposed as a **single-stage** model instead of a pipeline to learn and output regions and classifications *directly*.
- Fast RCNN combined the **bounding box regression and classification** in the neural network training itself.
  - Now the network has two heads, classification head, and bounding box regression head.

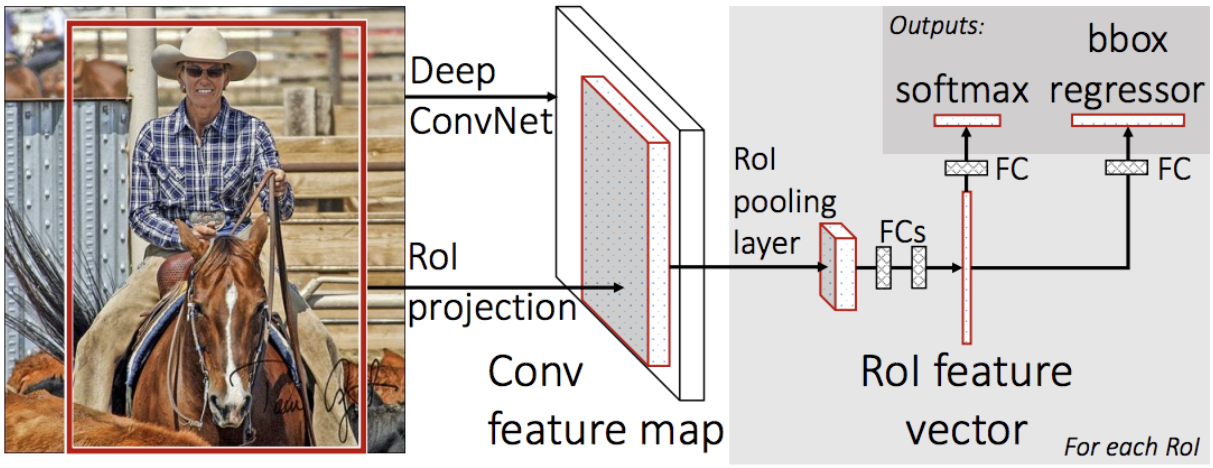
- The architecture of the model takes the photograph and a set of region proposals (from a selective search) as input that are passed through a deep convolutional neural network.
- A pre-trained CNN, such as a VGG-16, is used for feature extraction.
- The end of the deep CNN is a custom layer called a **Region of Interest Pooling** Layer, or **RoI Pooling**, that extracts features specific for a given input candidate region.





- Image Source

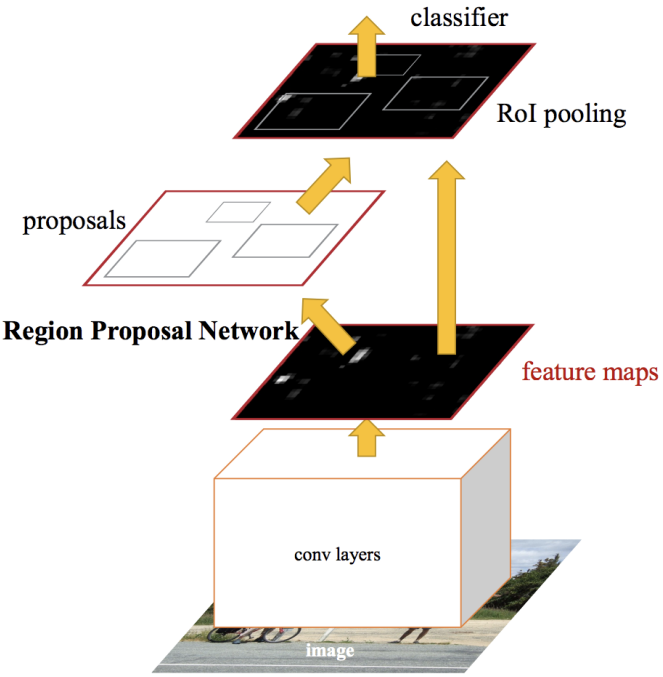
- The output of the CNN is then interpreted by a fully connected layer then the model has two outputs, one for the **class prediction** via a Softmax layer, and another with a linear output for the **bounding box**.
- This process is then repeated multiple times for each region of interest in a given image.



- The model is **significantly faster** to train and to make predictions, yet still requires a set of candidate regions to be proposed along with each input image.
    - Faster detection with **Truncated SVD** - compresses the fully-connected layers and reduces detection time by more than 30\% with only a small drop in mAP.

Faster R-CNNs

- The *slowest* part in Fast RCNN was **Selective Search** or Edge boxes.
- The Fast RCNN model architecture was further improved for both speed of training and detection by Shaoqing Ren, et al. in the 2016 paper titled "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks".
- The architecture was the basis for the *first-place* results achieved on both the ILSVRC-2015 and MS COCO-2015 object recognition and detection competition tasks.
- The idea was to replace selective search with a very small convolutional network called **Region Proposal Network** to generate regions of Interests.



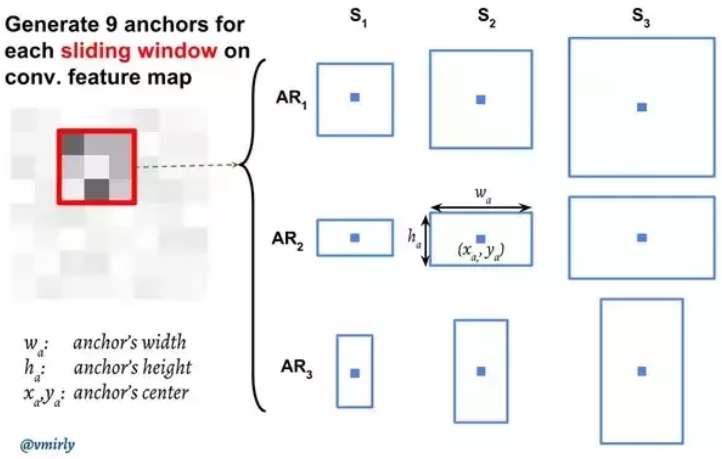
- The architecture was designed to both **propose and refine** region proposals as part of the training process, referred to as a **Region Proposal Network, or RPN**.
- These regions are then used in concert with a *Fast R-CNN* model in a single model design.
- These improvements both reduce the number of region proposals and accelerate the test-time operation of the model to near real-time with then state-of-the-art performance.

- The architecture is comprised of two modules:
    - **Module 1: Region Proposal Network** - CNN for proposing regions and the type of object to consider in the region.
    - **Module 2: Fast R-CNN** - CNN for extracting features from the proposed regions and outputting the bounding box and class labels.
- Both modules operate on the **same output of a deep CNN**.

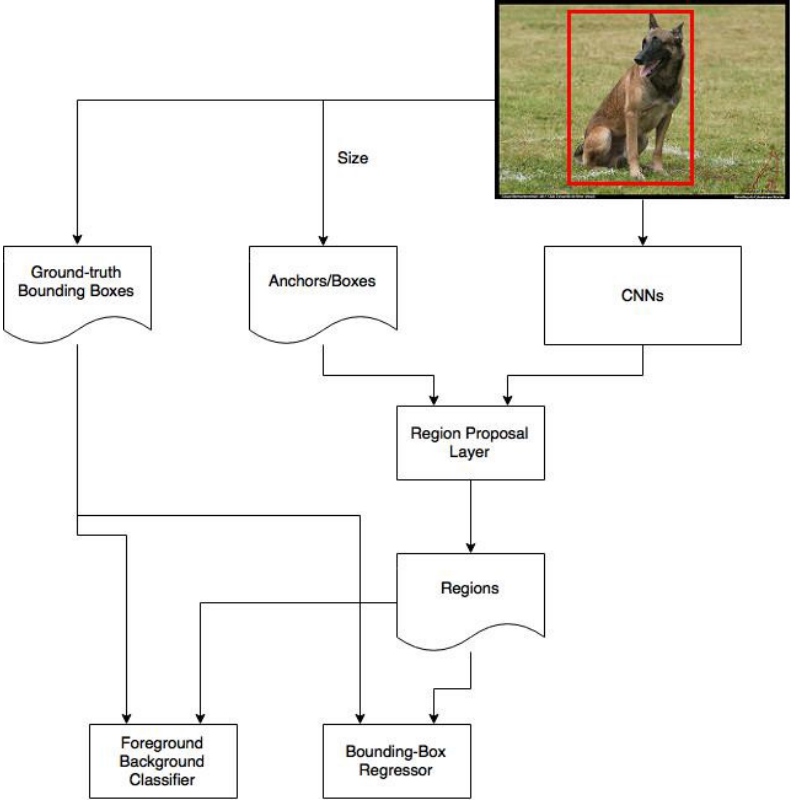
- The region proposal network acts as an *attention* mechanism for the Fast R-CNN network, informing the second network of where to look or pay attention.

- The RPN works by taking the output of a pre-trained deep CNN, such as VGG-16, and passing a small network over the feature map and outputting multiple region proposals and a class prediction for each.
- Region proposals are *bounding boxes*, based on **anchor boxes** or pre-defined shapes designed to accelerate and improve the proposal of regions.
- The class prediction is binary, indicating the presence of an object, or not, so-called “objectness” of the proposed region.

- **RPN Steps**:
  - Generate (pre-defined) anchor boxes.
  - Feed the possible regions into the RPN and classify each anchor box whether it is foreground or background (anchors that have a higher overlap with ground-truth boxes should be labeled as foreground, while others should be labeled as background).
  - The output is fed into a Softmax or logistic regression activation function, to predict the labels for each anchor. A similar process is used to refine the anchors and define the bounding boxes for the selected features (learn the shape offsets for anchor boxes to fit them for objects).



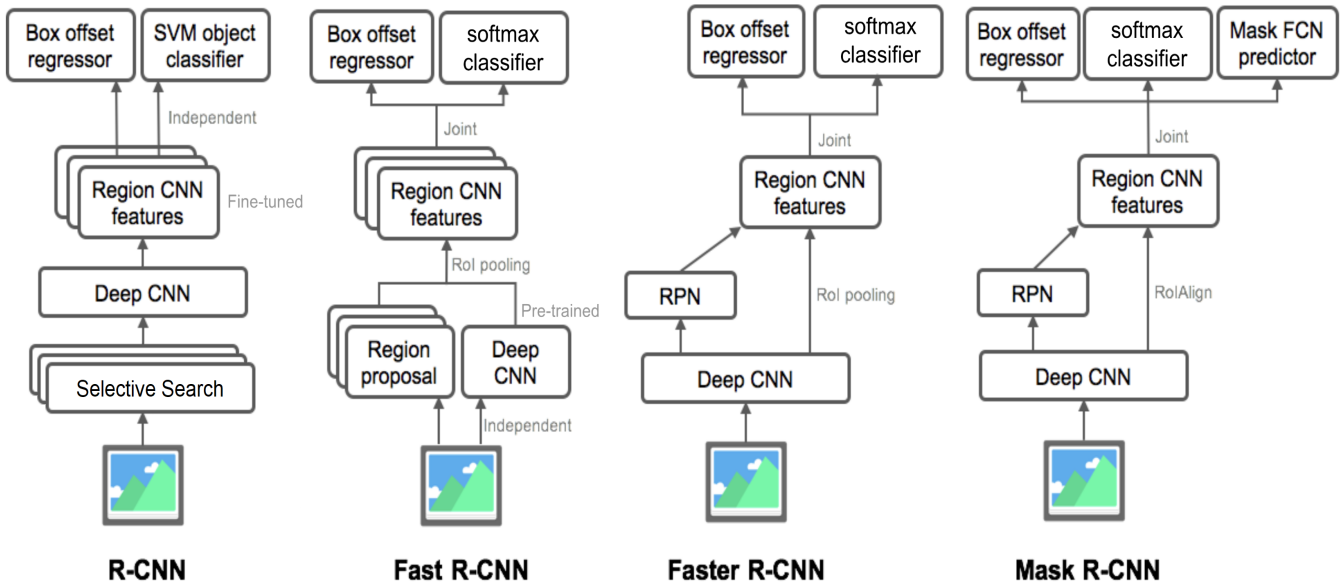
- Image Source and Read More (1)



- Image Source and Read More (2)

- Faster-RCNN is 10 times faster than Fast-RCNN with similar accuracy of datasets like VOC-2007.
- That is why Faster-RCNN is one of the most accurate object detection algorithms.

	R-CNN	Fast R-CNN	Faster R-CNN
Test Time Per Image (sec)	50	2	0.2
Speed Up	1x	25x	250x



- Image Source

R-CNN Family Implementations

- R-CNN, Fast R-CNN, Faster R-CNN and even Mask R-CNN are all implemented in **Detectron2 for PyTorch**.
  - It is better to use Faster R-CNN or Mask R-CNN (if you need segmentation).
- Detectron2 for PyTorch
  - Colab Notebook Demo of Detectron2

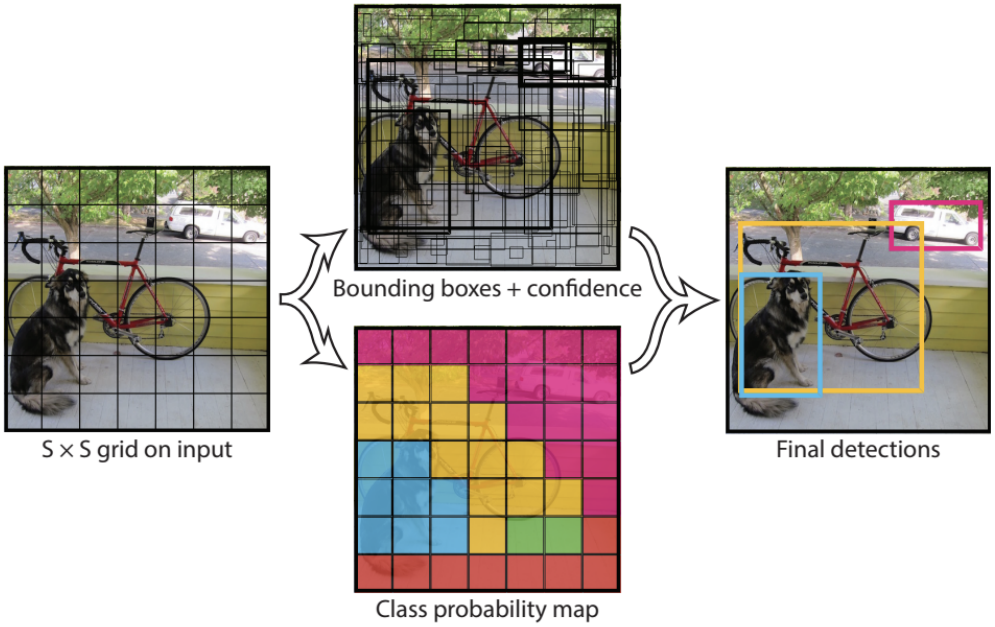
👀 You Only Look Once (YOLO) Family

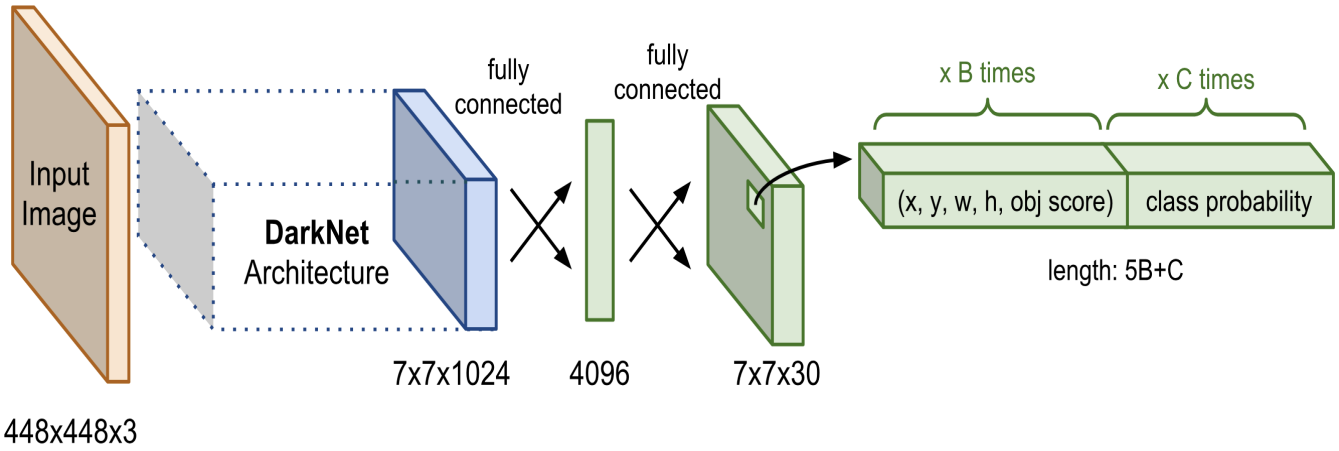
- All the methods discussed above handled detection as a classification problem by building a pipeline where first object proposals are generated and then these proposals are sent to classification/regression heads.
- Another popular family of object recognition models is referred to collectively as YOLO or "You Only Look Once," developed by Joseph Redmon, et al. which is a **regression**-based detection method.
- The R-CNN models may be generally *more accurate*, yet the YOLO family of models are *fast*, much faster than R-CNN, achieving object detection in *real-time*.
- YOLO Official Website

👁️ YOLO (v1)

- The YOLO model was first described by Joseph Redmon, et al. in the 2015 paper titled "You Only Look Once: Unified, Real-Time Object Detection".
- The approach involves a **single neural network** trained end-to-end that takes an image 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.

- 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 it.
  - YOLO divides each image into a grid of $S \times S$ and each grid cell predicts N (usually $N = 2$) bounding boxes and confidence where a bounding box involving the x, y coordinate, the width, the height and the confidence.
  - The confidence reflects the accuracy of the bounding box and whether the bounding box actually contains an object (regardless of class).
    - YOLO uses Non-Maximal Suppression (NMS) to only keep the best bounding box. The first step in NMS is to remove all the predicted bounding boxes that have a detection probability that is less than a given NMS threshold.
  - YOLO also predicts the classification score for each box for every class in training.
  - A total $S \times S \times N$ boxes are predicted. However, most of these boxes have low confidence scores.





- B is the number of bounding boxes from each cell, C is the number of classes.
- Image Source

- In test time, the test image is first broken up into a grid and the network then produces output

vectors, one for each grid cell.

- These vectors tell us if a cell has an object in it, what class the object is, and the bounding boxes for the object.
  - Since we're using two anchor boxes, we'll get two predicted anchor boxes for each grid cell. Most of the predicted anchor boxes will have a very low probability value of object being present in it.
- After producing these output vectors, non-maximal suppression is used to get rid of unlikely bounding boxes.
  - For each class, non-maximal suppression gets rid of the bounding boxes that have a confidence value lower than some given threshold.

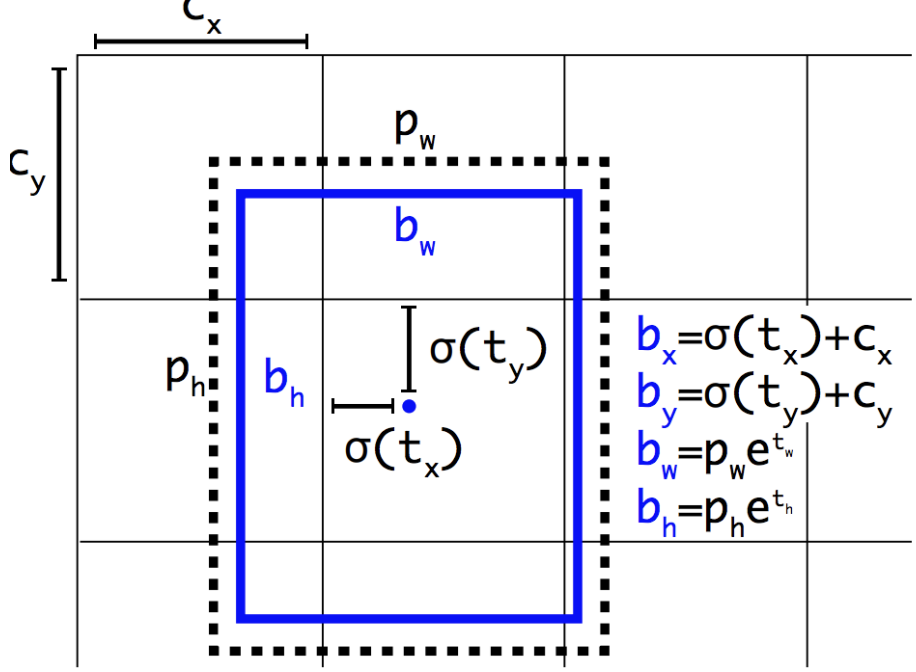
 YOLO v2 & v3

- The model was updated by Joseph Redmon and Ali Farhadi in an effort to further improve model performance in their 2016 paper titled "YOLO9000: Better, Faster, Stronger".
- Further improvements to the model were proposed by Joseph Redmon and Ali Farhadi in their 2018 paper titled "YOLOv3: An Incremental Improvement". The improvements were reasonably minor, including a deeper feature detector network and minor representational changes.

Main Improvements

- YOLOv2 - an instance of the model is described that was trained on two object recognition datasets in parallel, capable of predicting 9,000 object classes, hence given the name "YOLO9000", and later reffered to as "YOLOv2".
- A number of training and architectural changes were made to the model, such as the use of batch normalization and high-resolution input images.
- Like Faster R-CNN, YOLOv2 model makes use of **anchor boxes**, pre-defined bounding boxes with useful shapes and sizes that are tailored during training.
  - The choice of bounding boxes for the image is pre-processed using a k-means analysis on the training dataset.
- The predicted representation of the bounding boxes is changed to allow *small changes to have a less dramatic effect* on the predictions, resulting in a more stable model.
- Rather than predicting position and size directly, **offsets** are predicted for moving and reshaping the pre-defined anchor boxes relative to a grid cell and dampened by a logistic function.

- Example of the Representation Chosen when Predicting Bounding Box Position and Shape:



- At runtime, the image is propagated through the CNN only once.
- Hence, YOLO is super fast and can be run real time.
- Another key difference is that YOLO sees the complete image at once as opposed to looking at only a generated region proposals in the previous methods.
  - This contextual information helps in avoiding false positives.
- However, one limitation for YOLO is that it only predicts one type of class in one grid cell. Hence, it **struggles with very small objects**.
- YOLOv3 PyTorch Code
  - Another PyTorch Code

---------- </> CODE TIME ----------

- We will use the first code we provided (`YOLOv3 PyTorch Code`).
- Download the weights from this link
  - We use `yolov3-spp-ultralytics.pt` , but you can try different ones.
- Place it in the `weights` folder under the model directory (`yolov3`)

In [2]:
```
help(yolov3_detect)
```

```
Help on function yolov3_detect in module models.yolov3.detect:

yolov3_detect(img_source, img_size=416, save_img=False, device=device(type='cpu'), cfg
='./models/yolov3/cfg/yolov3-spp.cfg', names='./models/yolov3/data/coco.names', classes=
None, weights='./models/yolov3/weights/yolov3-spp-ultralytics.pt', output='./models/yolo
v3/output', conf_thresh=0.3, iou_thresh=0.6, fourcc='mp4v', half=False, view_img=False,
save_txt=False, agnostic_nms=False)
```

In [17]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
img_src = '/Users/yiftachedelstain/ee046746-computer-vision/models/yolov3/data/samples/b
```

```
output = '/Users/yiftachedelstain/ee046746-computer-vision/models/yolov3/output/bus.jpg'
# img_src = '/Users/yiftachedelstain/ee046746-computer-vision/models/yolov3/data/samples
# output = '/Users/yiftachedelstain/ee046746-computer-vision/models/yolov3/output/kofiko
yolov3_detect(img_src, device=device)

# show image
img = cv2.cvtColor(cv2.imread(output), cv2.COLOR_BGR2RGB)
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111)
ax.imshow(img)
ax.set_axis_off()
```

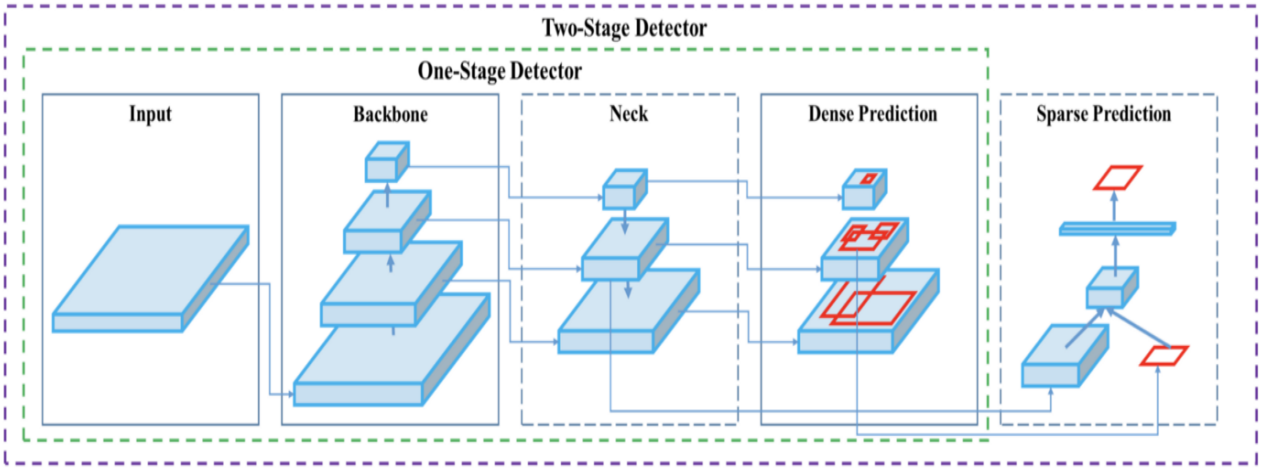
```
image 1/1 models\yolov3\data\samples\bus.jpg: 416x320 4 persons, 1 buss, Done. (0.113s)
Results saved to C:\Users\tabad\AnacondaWorkspace\Computer Vision\ee046746-computer-visi
on\./models/yolov3/output
Done. (0.269s)
```

YOLO v4

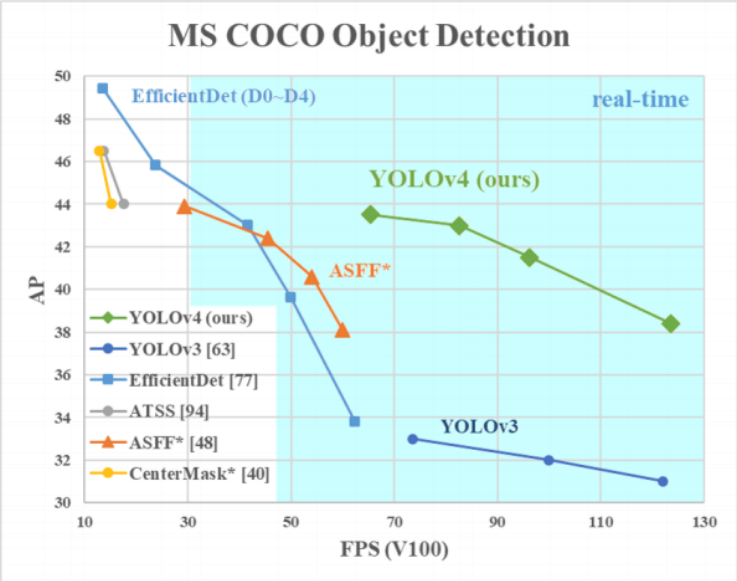
- In 2020, a major improvement to the YOLO model was intorduced in the paper YOLOv4: Optimal Speed and Accuracy of Object Detection.
- YOLOv4's architecture is composed of CSPDarknet53 (CSP: Cross-Stage-Partial connections, separating the current layer into 2 parts, one that will go through a block of convolutions, and one that won't and then aggregate the results) as a **backbone**, spatial pyramid pooling additional module, PANet path-aggregation **neck** and **YOLOv3 head**.

- Bag-Of-Freebies (BoF) and Bag-Of-Specials (BoS) - more improvements such as augementations, regularizations and special activations.
- YOLOv4 PyTorch Code

- A typical object detection architecture:



- YOLO is a **One-Stage-Detector**, while the R-CNN family is a **Two-Stage-Detector** (since in a region proposal network, you look at the image in two steps—the first to identify regions where there might be objects, and the next to specify it).
- Backbone - CSPDarknet53 is a novel backbone that can enhance the learning capability of CNN. The backbone is simply feature extraction (the backbone can be replaced with VGG, ResNet and etc...).
- Neck - The spatial pyramid pooling block is added over CSPDarknet53 to increase the receptive field and separate out the most significant context features.
- Neck - Instead of Feature Pyramid Networks (FPN) for object detection used in YOLOv3, the PANet is used as the method for parameter aggregation for different detector levels (aggregate information to get higher accuracy).
- Head - the same process as in YOLOv3. The network detects the bounding box coordinates (x, y, w, h) as well as the confidence score for a class (anchor-based technique).



- AP (Average Precision) and FPS (Frames Per Second) increased by 10% and 12% compared to YOLOv3

---------- </> CODE TIME ----------

In [18]:
```
%cd models/yolov4/
from demo import yolov4_detect
```

```
C:\Users\tabad\AnacondaWorkspace\Computer Vision\ee046746-computer-vision\models\yolov4
```

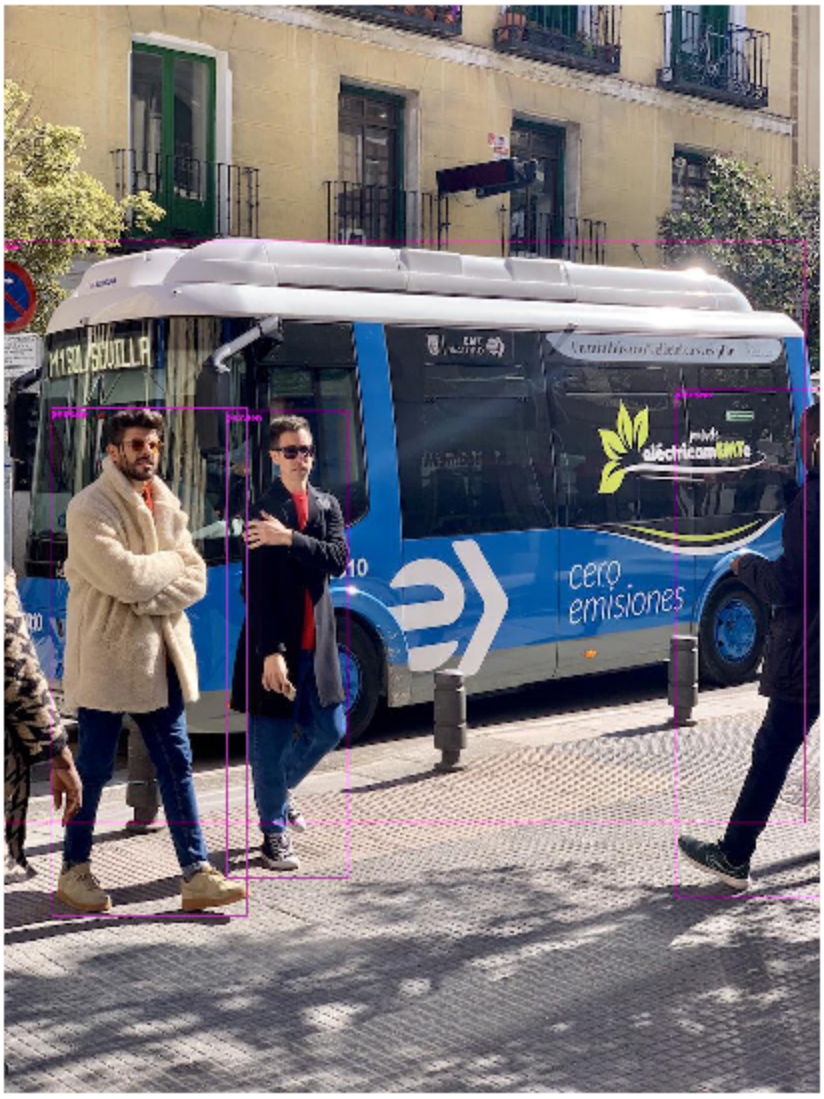
In [19]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
img_src = './Users/yiftachedelstain/ee046746-computer-vision/yolov3/data/samples/bus.jpg
output = '/Users/yiftachedelstain/ee046746-computer-vision/predictions.jpg'
# img_src = './Users/yiftachedelstain/ee046746-computer-vision/yolov3/data/samples/kofik
# output = '/Users/yiftachedelstain/ee046746-computer-vision/predictions.jpg'
yolov4_detect('/Users/yiftachedelstain/ee046746-computer-vision/cfg/yolov4.cfg', '/Users

# show image
img = cv2.cvtColor(cv2.imread(output), cv2.COLOR_BGR2RGB)
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111)
ax.imshow(img)
ax.set_axis_off()
```

```
convalution havn't activate linear
convalution havn't activate linear
convalution havn't activate linear
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   608 x 608 x   3   ->   608 x 608 x  32
    1 conv     64  3 x 3 / 2   608 x 608 x  32   ->   304 x 304 x  64
    2 conv     64  1 x 1 / 1   304 x 304 x  64   ->   304 x 304 x  64
    3 route  1
    4 conv     64  1 x 1 / 1   304 x 304 x  64   ->   304 x 304 x  64
    5 conv     32  1 x 1 / 1   304 x 304 x  64   ->   304 x 304 x  32
    6 conv     64  3 x 3 / 1   304 x 304 x  32   ->   304 x 304 x  64
    7 shortcut 4
    8 conv     64  1 x 1 / 1   304 x 304 x  64   ->   304 x 304 x  64
    9 route  8 2
   10 conv     64  1 x 1 / 1   304 x 304 x 128   ->   304 x 304 x  64
   11 conv    128  3 x 3 / 2   304 x 304 x  64   ->   152 x 152 x 128
   12 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64
   13 route  11
   14 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64
   15 conv     64  1 x 1 / 1   152 x 152 x  64   ->   152 x 152 x  64
   16 conv     64  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x  64
   17 shortcut 14
   18 conv     64  1 x 1 / 1   152 x 152 x  64   ->   152 x 152 x  64
   19 conv     64  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x  64
   20 shortcut 17
   21 conv     64  1 x 1 / 1   152 x 152 x  64   ->   152 x 152 x  64
   22 route  21 12
   23 conv    128  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x 128
   24 conv    256  3 x 3 / 2   152 x 152 x 128   ->    76 x  76 x 256
   25 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128
   26 route  24
   27 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128
   28 conv    128  1 x 1 / 1    76 x  76 x 128   ->    76 x  76 x 128
   29 conv    128  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 128
   30 shortcut 27
```

```
31 conv    128  1 x 1 / 1    76 x  76 x 128   ->    76 x  76 x 128
32 conv    128  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 128
33 shortcut 30
34 conv    128  1 x 1 / 1    76 x  76 x 128   ->    76 x  76 x 128
35 conv    128  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 128
36 shortcut 33
37 conv    128  1 x 1 / 1    76 x  76 x 128   ->    76 x  76 x 128
38 conv    128  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 128
39 shortcut 36
40 conv    128  1 x 1 / 1    76 x  76 x 128   ->    76 x  76 x 128
41 conv    128  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 128
42 shortcut 39
43 conv    128  1 x 1 / 1    76 x  76 x 128   ->    76 x  76 x 128
44 conv    128  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 128
45 shortcut 42
46 conv    128  1 x 1 / 1    76 x  76 x 128   ->    76 x  76 x 128
47 conv    128  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 128
48 shortcut 45
49 conv    128  1 x 1 / 1    76 x  76 x 128   ->    76 x  76 x 128
50 conv    128  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 128
51 shortcut 48
52 conv    128  1 x 1 / 1    76 x  76 x 128   ->    76 x  76 x 128
53 route  52 25
54 conv    256  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 256
55 conv    512  3 x 3 / 2    76 x  76 x 256   ->    38 x  38 x 512
56 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
57 route  55
58 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
59 conv    256  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 256
60 conv    256  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 256
61 shortcut 58
62 conv    256  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 256
63 conv    256  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 256
64 shortcut 61
65 conv    256  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 256
66 conv    256  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 256
67 shortcut 64
68 conv    256  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 256
69 conv    256  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 256
70 shortcut 67
71 conv    256  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 256
72 conv    256  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 256
73 shortcut 70
74 conv    256  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 256
75 conv    256  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 256
76 shortcut 73
77 conv    256  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 256
78 conv    256  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 256
79 shortcut 76
80 conv    256  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 256
81 conv    256  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 256
82 shortcut 79
83 conv    256  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 256
84 route  83 56
85 conv    512  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 512
86 conv   1024  3 x 3 / 2    38 x  38 x 512   ->    19 x  19 x1024
87 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
88 route  86
89 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
90 conv    512  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 512
91 conv    512  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x 512
92 shortcut 89
93 conv    512  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 512
94 conv    512  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x 512
95 shortcut 92
96 conv    512  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 512
```

```
  97 conv    512  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x 512
  98 shortcut 95
  99 conv    512  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 512
 100 conv    512  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x 512
 101 shortcut 98
 102 conv    512  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 512
 103 route  102 87
 104 conv   1024  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x1024
 105 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
 106 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024
 107 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
 108 max          5 x 5 / 1    19 x  19 x 512   ->    19 x  19 x 512
 109 route  107
 110 max          9 x 9 / 1    19 x  19 x 512   ->    19 x  19 x 512
 111 route  107
 112 max         13 x 13 / 1    19 x  19 x 512   ->    19 x  19 x 512
 113 route  112 110 108 107
 114 conv    512  1 x 1 / 1    19 x  19 x2048   ->    19 x  19 x 512
 115 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024
 116 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
 117 conv    256  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 256
 118 upsample           * 2    19 x  19 x 256   ->    38 x  38 x 256
 119 route  85
 120 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
 121 route  120 118
 122 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
 123 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512
 124 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
 125 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512
 126 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
 127 conv    128  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 128
 128 upsample           * 2    38 x  38 x 128   ->    76 x  76 x 128
 129 route  54
 130 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128
 131 route  130 128
 132 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128
 133 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256
 134 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128
 135 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256
 136 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128
 137 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256
 138 conv    255  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 255
 139 detection
 140 route  136
 141 conv    256  3 x 3 / 2    76 x  76 x 128   ->    38 x  38 x 256
 142 route  141 126
 143 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
 144 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512
 145 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
 146 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512
 147 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
 148 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512
 149 conv    255  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 255
 150 detection
 151 route  147
 152 conv    512  3 x 3 / 2    38 x  38 x 256   ->    19 x  19 x 512
 153 route  152 116
 154 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
 155 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024
 156 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
 157 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024
 158 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
 159 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024
 160 conv    255  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 255
 161 detection
Loading weights from ./weights/yolov4.weights... Done!
```

```
-----------------------------------
     get_region_boxes : 0.466347
                  nms : 0.006983
   post process total : 0.473330
-----------------------------------
-----------------------------------
     get_region_boxes : 0.443647
                  nms : 0.004985
   post process total : 0.448632
-----------------------------------
../yolov3/data/samples/bus.jpg: Predicted in 0.740411 seconds.
person: 1.000000
person: 1.000000
bus: 0.920363
person: 1.000000
save plot results to predictions.jpg
```



In [7]:
```
%cd ./Users/yiftachedelstain/ee046746-computer-vision/./Users/yiftachedelstain/ee046746-
```

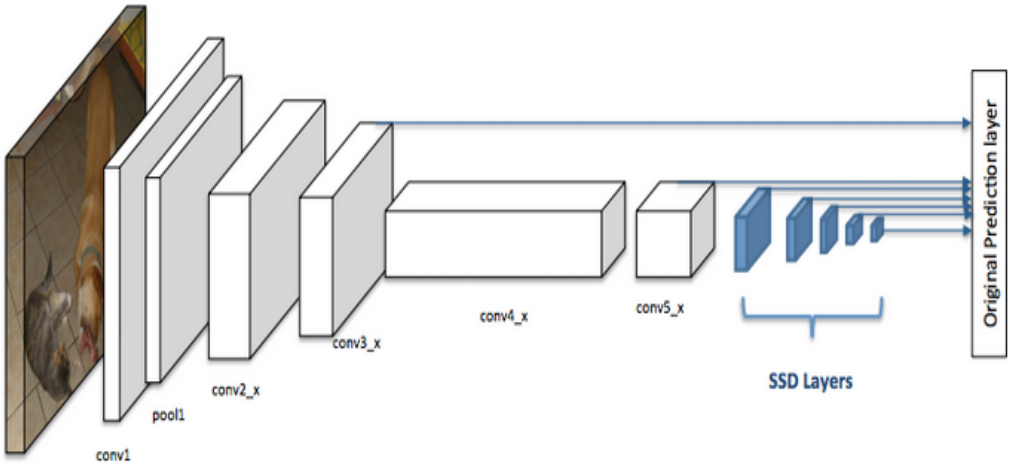
```
C:\Users\tabad\AnacondaWorkspace\Computer Vision\ee046746-computer-vision
```

Single Shot Multibox Detection (SSD)

- A *regression* type object detector which was introduced by Liu et al. in 2016 in a paper titled "SSD: Single Shot MultiBox Detector".

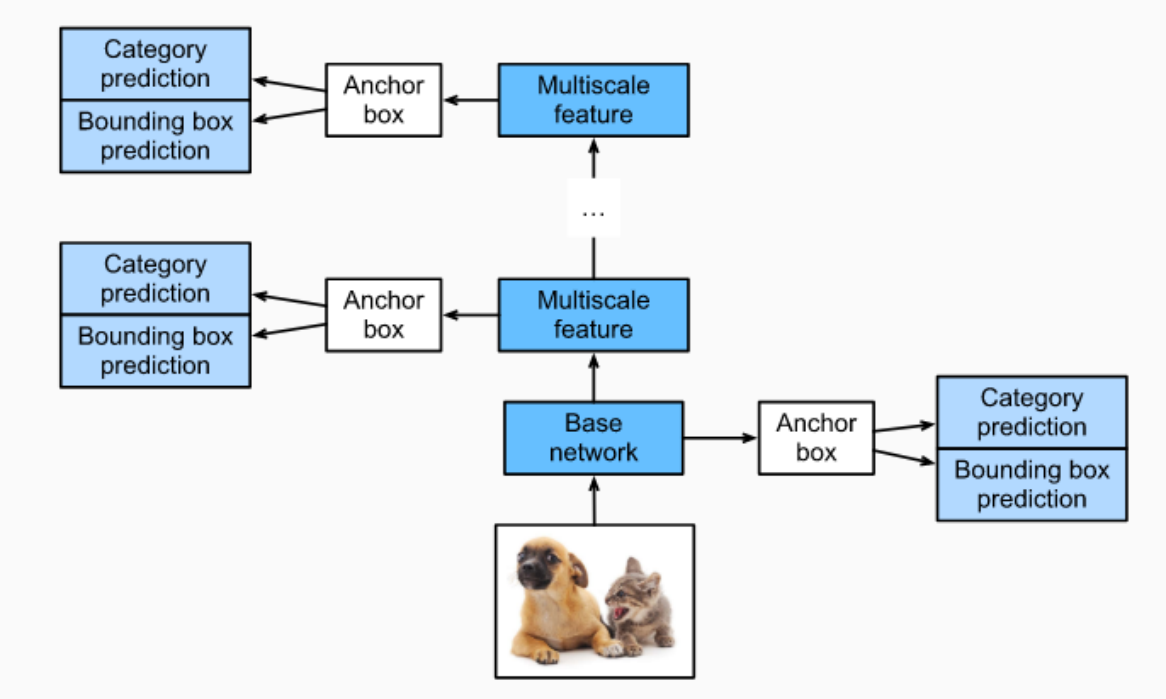
- Single-shot models encapsulate both localization and detection tasks in a single forward sweep of the network, resulting in significantly faster detections while deployable on lighter hardware.
- Single Shot Detector achieves a good balance between speed and accuracy.

    - SSD has two components: a **backbone model** and **SSD** head.
        - **Backbone model** usually is a pre-trained image classification network as a feature extractor. This is typically a network like ResNet trained on ImageNet from which the final fully connected classification layer has been removed.
        - **The SSD** head is just one or more convolutional layers added to this backbone and the outputs are interpreted as the bounding boxes and classes of objects in the spatial location of the final layers activations.



    - SSD runs a convolutional network on input image only once and calculates a feature map.
    - Then, a small 3×3 sized convolutional kernel is run on this feature map to predict the bounding boxes and classification probability.
    - SSD also uses **anchor boxes** at various aspect ratio similar to Faster-RCNN and learns the offset rather than learning the box.
    - In order to handle the scale, SSD predicts bounding boxes after multiple convolutional layers. Since each convolutional layer operates at a different scale, it is able to detect objects of various scales.

    - Unlike YOLO, SSD does not split the image into grids of *arbitrary* size but predicts offset of predefined anchor boxes for every location of the feature map.
        - Each box has a fixed size and position relative to its corresponding cell. All the anchor boxes tile the whole feature map in a convolutional manner.
    - The extra step taken by SSD is that it applies **more convolutional layers to the backbone feature map** and has each of these convolution layers output a object detection results.
    - As earlier layers bearing smaller receptive field can represent smaller sized objects, predictions from earlier layers help in dealing with smaller sized objects.
    - SSD PyTorch Code (NVIDIA)
        - Another PyTorch Code



- Image Source

---------- CODE TIME ----------

- We will use Nvidia's model which is available in the PyTorch Hub.
- Feel free to download the original model from the provided links to get a look at the training procedure.
- Note: the original model uses `skimage` to load and transform images, which may results in errors because it is deprecated, so just go to `hubconf.py` (where the model was downloaded) and change `skimage` to `cv2`.
  - `img = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB) / 255.0`
  - `imgScaled = cv2.resize(img, (input_width, res))`

In [8]:
```
# set device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
# load the model from the hub
ssd_model = torch.hub.load('NVIDIA/DeepLearningExamples:torchhub', 'nvidia_ssd', model_m
# load utils
utils = torch.hub.load('NVIDIA/DeepLearningExamples:torchhub', 'nvidia_ssd_processing_ut
# change to evaluation mode
ssd_model.eval();
```

```
cuda:0
Using cache found in C:\Users\tabad/.cache\torch\hub\NVIDIA_DeepLearningExamples_torchhu
b
Using cache found in C:\Users\tabad/.cache\torch\hub\NVIDIA_DeepLearningExamples_torchhu
b
```

In [13]:
```
img_src = ['/Users/yiftachedelstain/ee046746-computer-vision/models/yolov3/data/samples/
# img_src = ['/Users/yiftachedelstain/ee046746-computer-vision/models/yolov3/data/sample
# load and convert to tensors
inputs = [utils.prepare_input(img) for img in img_src]
```

```
inputs_t = utils.prepare_tensor(inputs)
# predict
with torch.no_grad():
    detections_batch = ssd_model(inputs_t)
```

By default, raw output from SSD network per input image contains 8732 boxes with localization and class probability distribution. Let's filter this output to only get reasonable detections (confidence>40%) in a more comprehensive format.

In [14]:
```
results_per_input = utils.decode_results(detections_batch)
best_results_per_input = [utils.pick_best(results, 0.40) for results in results_per_inpu
```

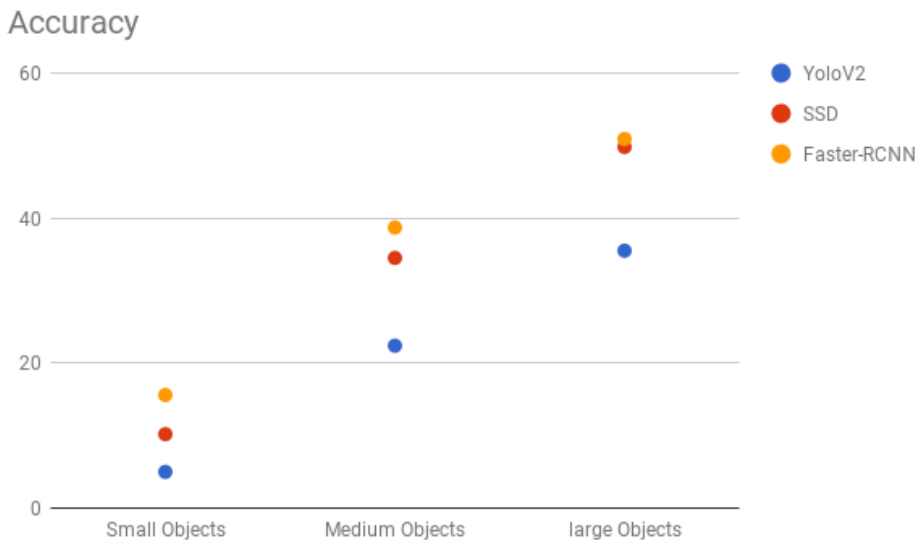
The model was trained on COCO dataset, which we need to access in order to translate class IDs into object names. For the first time, downloading annotations may take a while.

In [15]:
```
classes_to_labels = utils.get_coco_object_dictionary()
```

In [16]:
```
# visualize the results
for image_idx in range(len(best_results_per_input)):
    fig, ax = plt.subplots(1, figsize=(8, 8))
    # show original, denormalized image
    image = inputs[image_idx] / 2 + 0.5
    ax.imshow(image)
    # add detections
    bboxes, classes, confidences = best_results_per_input[image_idx]
    for idx in range(len(bboxes)):
        left, bot, right, top = bboxes[idx]
        x, y, w, h = [val * 300 for val in [left, bot, right - left, top - bot]]
        rect = patches.Rectangle((x, y), w, h, linewidth=1, edgecolor='r', facecolor='no
        ax.add_patch(rect)
        ax.text(x, y, "{} {:.0f}%".format(classes_to_labels[classes[idx] - 1], confidenc
                bbox=dict(facecolor='white', alpha=0.5))
        ax.set_axis_off()
```

Which Algorithm to Use?

- Choice of a right object detection method is crucial and depends on the problem you are trying to solve and the setup.
- Object Detection is the backbone of many practical applications of computer vision such as autonomous cars, security and surveillance, and many industrial applications.
- **Faster-RCNN** is the choice if you mostly care about the accuracy numbers.
- **SSD** provides good balance between performance and speed.
- **YOLO** is super fast but on the account of performance.



Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- (Deep) Object Detection - Stanford CS231 - Lecture 11 | Detection and Segmentation
- Object Detection - C4W3L03 Object Detection - Andrew Ng
- Non-Maximum Supression - C4W3L07 Nonmax Suppression - Andrew Ng
- Region Proposals - C4W3L10 Region Proposals - Andrew Ng
- R-CNN, Fast R-CNN, Faster R-CNN - RCNN, FAST RCNN, FASTER RCNN : OBJECT DETECTION AND LOCALIZATION THROUGH DEEP NEURAL NETWORKS
- YOLO - C4W3L09 YOLO Algorithm - Andrew Ng
- YOLO v3 - YOLOv3
- YOLO v4 - Yolo V4 - How it Works and Why it's So Amazing!
- SSD - Single shot detectors - training

Credits

- EE 046746 Spring 21 - Tal Daniel
- Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSDO - Ankit Sachan
- A Gentle Introduction to Object Recognition With Deep Learning - Jason Brownlee
- How single-shot detector (SSD) works?
- What's new in YOLOv4?
- Introduction to YOLOv4: Research review
- Slides by David Dov and Yael Amiay.
- Icons from Icon8.com - https://icons8.com