ECE 285 – MLIP – Project C Multi-Object detection using Deep Learning

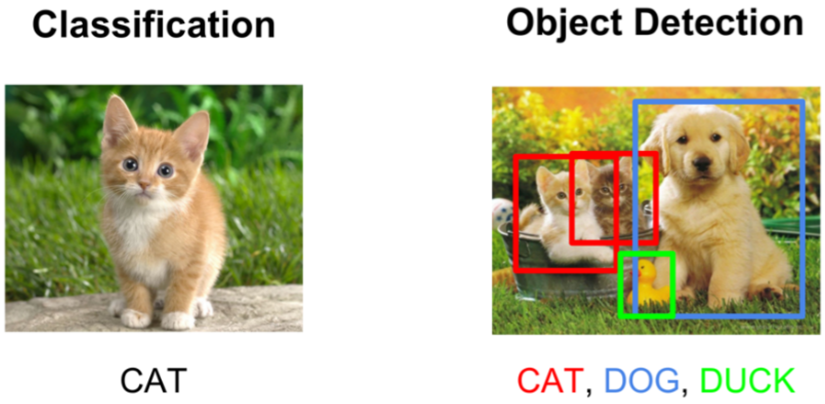
Written by Abhilash Kasarla. Last updated on April 30, 2019.

The goal of this project is to achieve multi-object detection using Deep Learning methods. With recent advancements in Deep Learning-based computer vision models, object detection applications are easier to develop than ever before. In addition, with current approaches focusing on full end-to-end pipelines, performance has also improved significantly, enabling real-time use cases.

Note that additional information may be posted on Piazza by the Instructor or the TAs. This document is also subject to be updated. Most recent instructions will always prevail over the older ones. So look at the posting/updating dates and make sure to stay updated.

1 Image Classification vs. object detection

People often confuse image classification and object detection scenarios. In general, if you want to classify an image into a certain category, you use image classification. On the other hand, if you aim to identify the location of objects in an image, and e.g. count the number of instances of an object, you can use object detection.



There is however some overlap between these two scenarios. If you want to classify an image into a certain category, it could happen that the object or the characteristics that are required to perform categorization are too small with respect to the full image. In that case, you would achieve better performance with object detection instead of image classification even if you are not interested in the exact location or counts of the object.

With an image classification model, you generate image features (through traditional or deep learning methods) of the full image. These features are aggregates of the image. With object detection, you do this on a more fine-grained, granular, regional level of the image.

2 Dataset

Several datasets have been released for object detection challenges. Researchers publish results of their algorithms applied to these challenges. Specific performance metrics have been developed to take into account the spatial position of the detected object and the accuracy of the predicted categories.

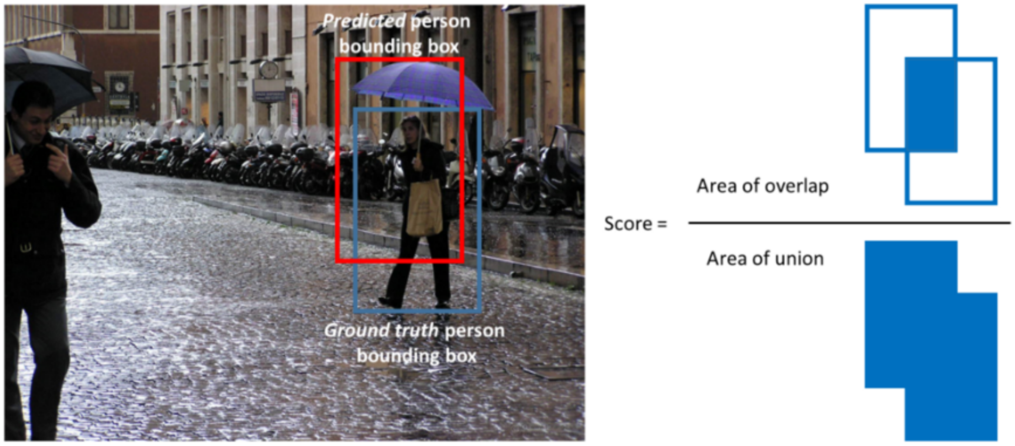
For this project you should use the PASCAL Visual Object Classification (PASCAL VOC) dataset. This is a well-known dataset for object detection, classification, segmentation of objects and so on. There are 8 different challenges spanning from 2005 to 2012, each of them having its own specificities. There are around 10 000 images for training and validation containing bounding boxes with objects. Although, the PASCAL VOC dataset contains only 20 categories, it is still considered as a reference dataset in the object detection problem. You will be using the 2012 version of the dataset for this project. You can access the dataset at `/datasets/ee285f-public/PascalVOC2012` on ieng6 cluster. The other datasets that are generally used to train object detection models are ImageNet, Microsoft's Common Objects in Context (COCO), Oxford-IIIT Pet etc.

3 Evaluation metric

The most common evaluation metric that is used in object recognition tasks is 'mAP', which stands for 'mean average precision'. It is a number from 0 to 100 and higher values are typically better, but its value is different from the accuracy metric in classification.

Each bounding box will have a score associated (likelihood of the box containing an object). Based on the predictions a precision-recall curve (PR curve) is computed for each class by varying the score threshold. The average precision (AP) is the area under the PR curve. First the AP is computed for each class, and then averaged over the different classes to get mAP. The mAP metric avoids having extreme specialization in few classes and thus weak performances in others.

Note that a detection is a true positive if it has an 'intersection over union' (IoU or overlap) with the ground-truth box greater than some threshold (usually 0.5). Instead of using mAP we typically use mAP@0.5 or mAP@0.25 to refer to the IoU that was used.

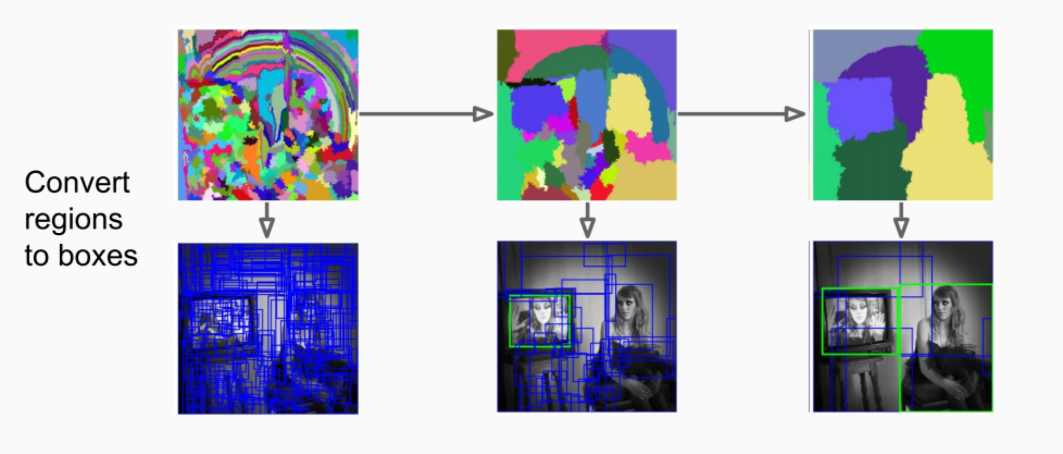

Predicted person bounding box
Ground truth person bounding box
Score =
Area of overlap
Area of union

A visualisation of the definition of IoU.

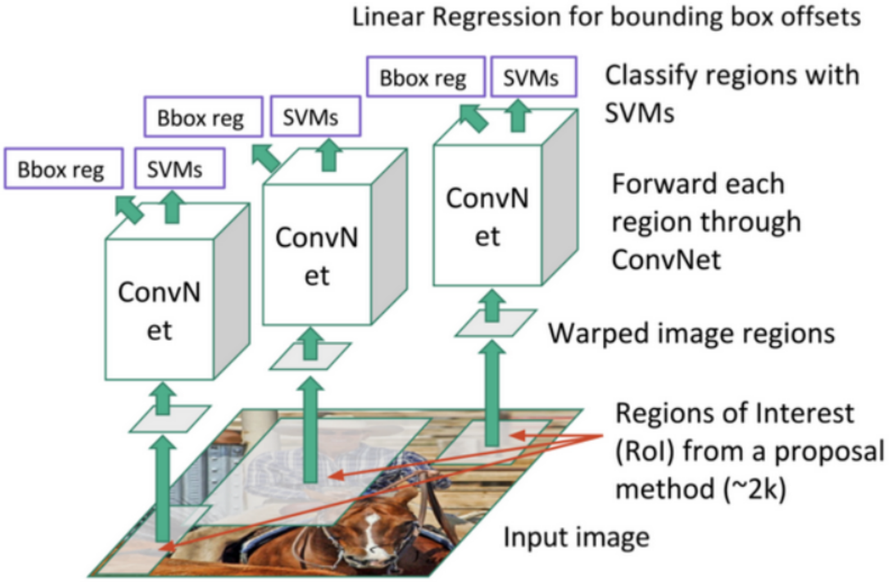
4 Methods

We will review some of the famous object detection algorithms in this section to give you a feel of how to approach the problem. Then you are free to mix and match approaches or try out a novel method to solve this problem.

Region-based Convolutional Neural Network (R-CNN) The first models intuitively begin with the region search and then perform the classification. In R-CNN, the selective search method developed by J.R.R. Uijlings *et al.* (2012) is an alternative to exhaustive search in an image to capture object location. It initializes small regions in an image and merges them with a hierarchical grouping. Thus, the final group is a box containing the entire image. The detected regions are merged according to a variety of color spaces and similarity metrics. The output is a few number of region proposals which could contain an object by merging small regions.



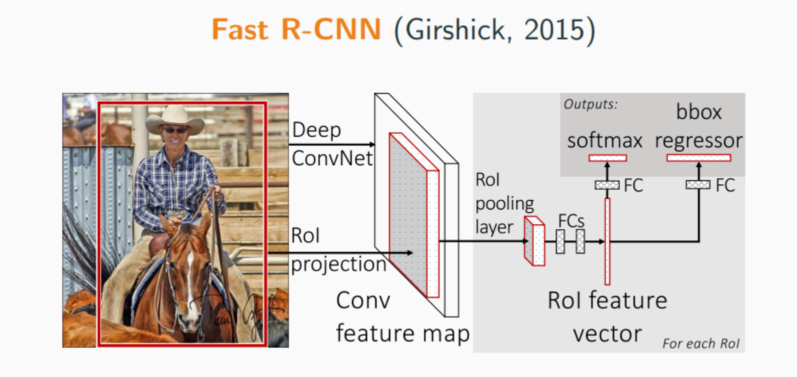
The R-CNN model (R. Girshick *et al.*, 2014) combines the selective search method to detect region proposals and deep learning to find out the object in these regions. Each region proposal is resized to match the input of a CNN from which we extract a 4096-dimension vector of features. The features vector is fed into multiple classifiers to produce probabilities to belong to each class. Each one of these classes has a SVM classifier trained to infer a probability to detect this object for a given vector of features. This vector also feeds a linear regressor to adapt the shapes of the bounding box for a region proposal and thus reduce localization errors. The best R-CNNs models have achieved a 62.4% mAP score over the PASCAL VOC 2012 dataset and a 31.4% mAP score over the 2013 ImageNet dataset.

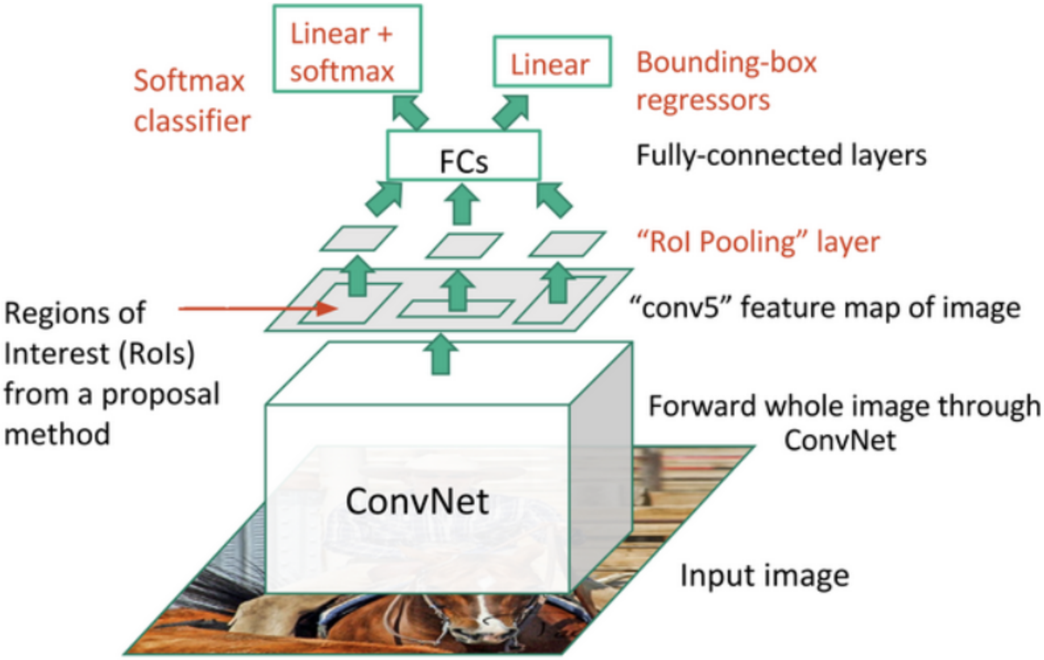

Linear Regression for bounding box offsets
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Classify regions with SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Region-based Convolution Network (R-CNN). Each region proposal feeds a CNN to extract a features vector, possible objects are detected using multiple SVM classifiers and a linear regressor modifies the coordinates of the bounding box. Source: J. Xu's Blog

Fast Region-based Convolutional Network (Fast R-CNN) The purpose of the Fast Region-based Convolutional Network (Fast R-CNN) developed by R. Girshick (2015) is to reduce the time consumption related to the high number of models necessary to analyze all region proposals.

A main CNN with multiple convolutional layers is taking the entire image as input instead of using a CNN for each region proposals (R-CNN). Region of Interests (RoIs) are detected with the selective search method applied on the produced feature maps. Formally, the feature maps size is reduced using a RoI pooling layer to get valid Region of Interests with fixed height and width as hyperparameters. Each RoI layer feeds fully-connected layers creating a features vector. The vector is used to predict the observed object with a SoftMax classifier and to adapt bounding box localizations with a linear regressor.

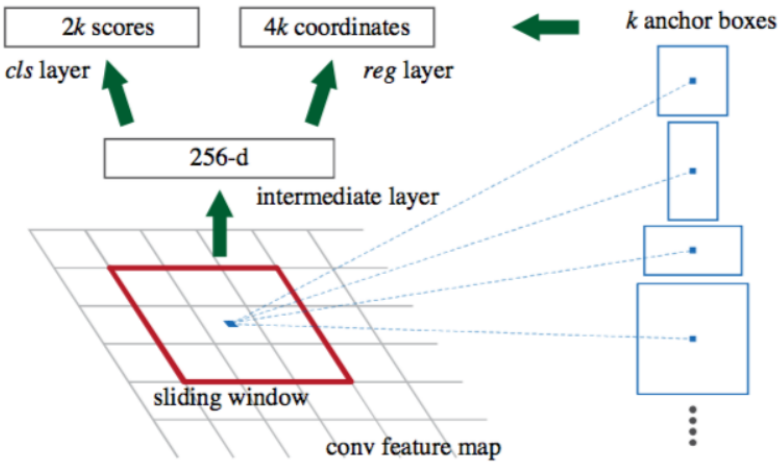




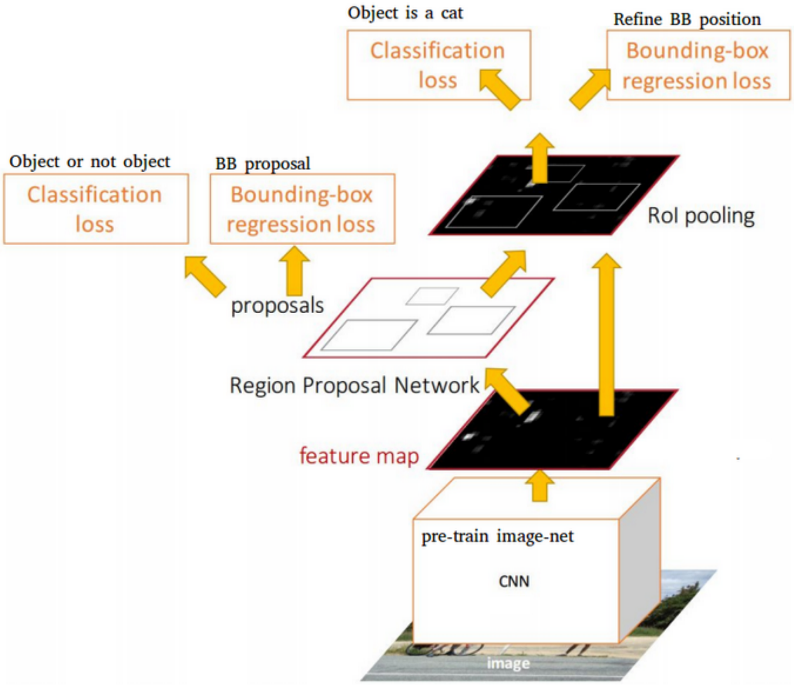
The entire image feeds a CNN model to detect RoI on the feature maps. Each region is separated using a RoI pooling layer and it feeds fully-connected layers. This vector is used by a softmax classifier to detect the object and by a linear regressor to modify the coordinates of the bounding box. Source: J. Xu's Blog

(*Also refer to the chapter 5 of lecture slides)

Faster Region-based Convolutional Network (Faster R-CNN) Region proposals detected with the selective search method were still necessary in the previous model, which is computationally expensive. S. Ren *et al.* (2016) have introduced Region Proposal Network (RPN) to directly generate region proposals, predict bounding boxes and detect objects. The Faster Region-based Convolutional Network (Faster R-CNN) is a combination between the RPN and the Fast R-CNN model. A CNN model takes as input the entire image and produces feature maps. A window of size 3 × 3 slides all the feature maps and outputs a features vector linked to two fully-connected layers, one for box-regression and one for box-classification. Multiple region proposals are predicted by the fully-connected layers. A maximum of k regions is fixed thus the output of the box-regression layer has a size of 4k (coordinates of the boxes, their height and width) and the output of the box-classification layer a size of 2k ("objectness" scores to detect an object or not in the box). The k region proposals detected by the sliding window are called anchors. When the anchor boxes are detected, they are selected by applying a threshold over the "ob-



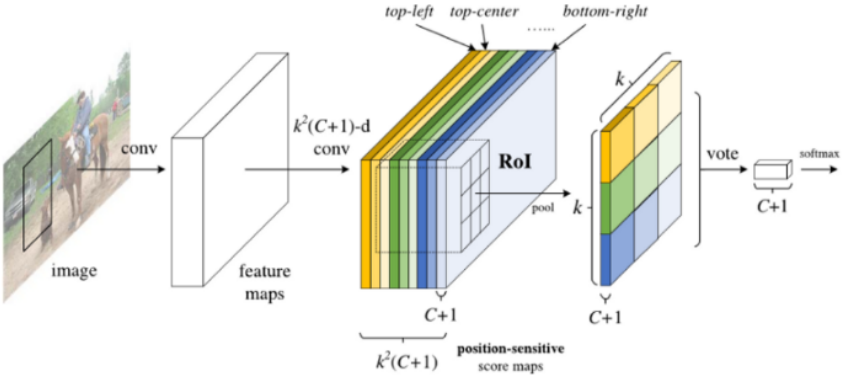
Detecting the anchor boxes for a single 3x3 window. Source: S. Ren and al. (2016)



The entire image feeds a CNN model to produce anchor boxes as region proposals with a confidence to contain an object. A Fast R-CNN is used taking as inputs the feature maps and the region proposals. For each box, it produces probabilities to detect each object and correction over the location of the box. Source: J. Xu's Blog jectness" score to keep only the relevant boxes. These anchor boxes and the feature maps computed by the initial CNN model feeds a Fast R-CNN model.

Faster R-CNN uses RPN to avoid the selective search method, it accelerates the training and testing processes, and improve the performances. The RPN uses a pre-trained model over the ImageNet dataset for classification and it is fine-tuned on the PASCAL VOC dataset. Then the generated region proposals with anchor boxes are used to train the Fast R-CNN. This process is iterative.

Region-based Fully Convolutional Network (R-FCN) Fast and Faster R-CNN methodologies consist in detecting region proposals and recognize an object in each region. The Region-based Fully Convolutional Network (R-FCN) released by J. Dai *et al.* (2016) is a model with only convolutional layers allowing complete backpropagation for training and inference. The authors have merged the two basic steps in a single model to take into account simultaneously the object detection (location invariant) and its position (location variant).



The input image feeds a ResNet model to produce feature maps. A RPN model detects the Region of Interests and a score is computed for each region to determine the most likely object if there is one. Source: J. Dai and al. (2016)

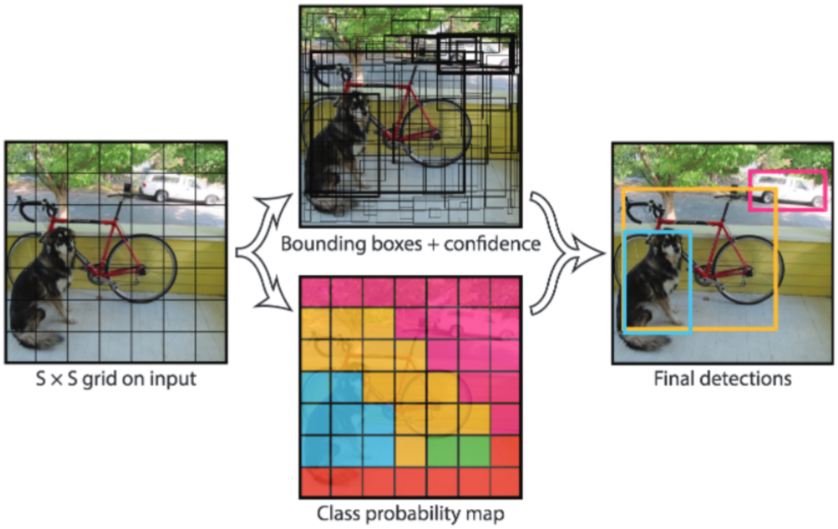
A ResNet-101 model takes the initial image as input. The last layer outputs feature maps, each one is specialized in the detection of a category at some location. For example, one feature map is specialized in the detection of a cat, another one in a banana and so on. Such feature maps are called position-sensitive score maps because they take into account the spatial localization of a particular object. It consists of $k \times k \times (C+1)$ score maps where k is the size of the score map, and C the number of classes. All these maps form the score bank. Basically, we create patches that can recognize part of an object. For example, for k=3, we can recognize 3x3 parts of an object.

In parallel, we need to run a RPN to generate Region of Interest (RoI). Finally, we cut each RoI in bins and we check them against the score bank. If enough of these parts are activated, then the patch vote 'yes', I recognized the object.

You look only once (YOLO) The YOLO model (J. Redmon *et al.*, 2016) directly predicts bounding boxes and class probabilities with a single network in a single evaluation. The simplicity of the YOLO model allows real-time predictions.

Initially, the model takes an image as input. It divides it into an SxS grid. Each cell of this grid predicts B bounding boxes with a confidence score. This confidence is simply the probability to detect the object multiply by the IoU between the predicted and the ground truth boxes.

The CNN used is inspired by the GoogLeNet model introducing the inception modules. The network has 24 convolutional layers followed by 2 fully-connected layers. Reduction layers with 1x1 filters followed by 3x3 convolutional layers replace the initial inception modules. The Fast YOLO model is a lighter

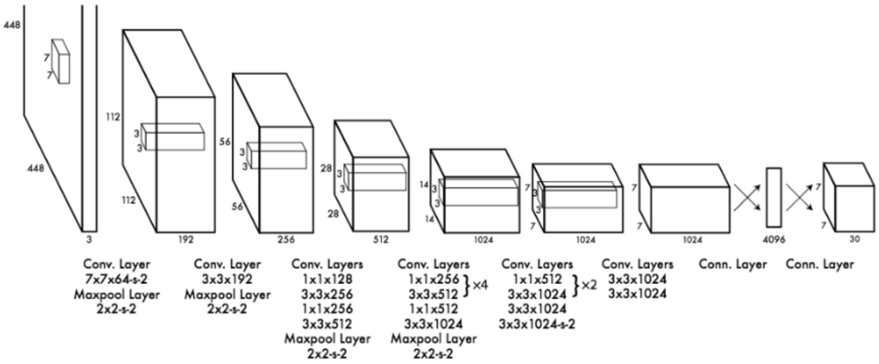


Example of application. The input image is divided into an SxS grid, B bounding boxes are predicted (regression) and a class is predicted among C classes (classification) over the most confident ones. Source: J. Redmon and al. (2016)

version with only 9 convolutional layers and fewer number of filters. Most of the convolutional layers are pretrained using the ImageNet dataset with classification. Four convolutional layers followed by two fully-connected layers are added to the previous network and it is entirely retrained with the 2007 and 2012 PASCAL VOC datasets.

The final layer outputs a $S \times S \times (C + 5B)$ tensor corresponding to the predictions for each cell of the grid. C is the number of estimated probabilities for each class. B is the fixed number of anchor boxes per cell, each of these boxes being related to 4 coordinates (coordinates of the center of the box, width and height) and a confidence value.

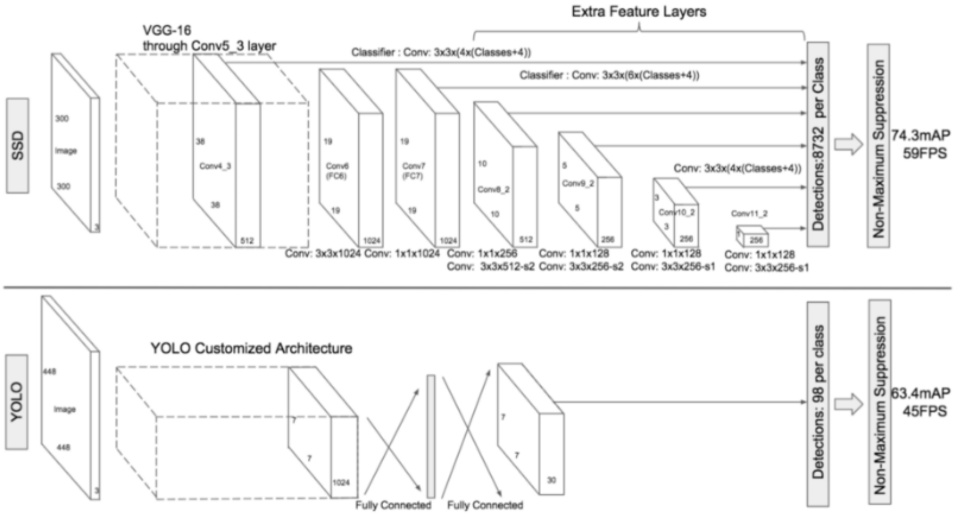
With the previous models, the predicted bounding boxes often contained an object. The YOLO model however predicts a high number of bounding boxes. Thus there are a lot of bounding boxes without any object. The Non-Maximum Suppression (NMS) method is applied at the end of the network. It consists in merging highly-overlapping bounding boxes of a same object into a single one. The authors noticed that there are still few false positive detected.



YOLO architecture: it is composed of 24 convolutional layers and 2 fully-connected layers. Source: J. Redmon and al. (2016)

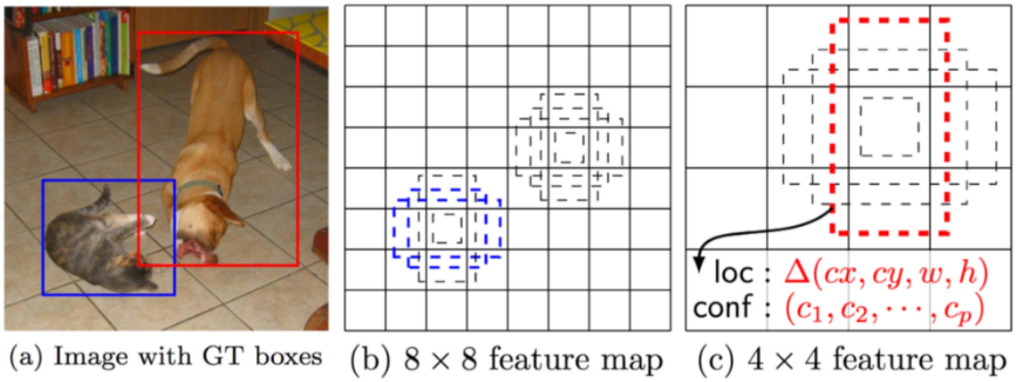
Single-Shot Detector (SSD) Similarly to the YOLO model, W. Liu *et al.* (2016) have developed a Single-Shot Detector (SSD) to predict all at once the bounding boxes and the class probabilities with a end-to-end CNN architecture.

The model takes an image as input which passes through multiple convolutional layers with different sizes of filter (10x10, 5x5 and 3x3). Feature maps from convolutional layers at different position of the network are used to predict the bounding boxes. They are processed by specific convolutional layers with 3x3 filters called extra feature layers to produce a set of bounding boxes similar to the anchor boxes of the Fast R-CNN.



Comparison between the SSD and the YOLO architectures. The SSD model uses extra feature layers from different feature maps of the network in order to increase the number of relevant bounding boxes. Source: W. Liu and al. (2016)

Each box has 4 parameters: the coordinates of the center, the width and the height. At the same time, it produces a vector of probabilities corresponding to the confidence over each class of object.


loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$

(a) Image with GT boxes (b) 8 × 8 feature map (c) 4 × 4 feature map

SSD Framework. (a) The model takes an image and its ground truth bounding boxes. Small sets of boxes with different aspect ratios are fixed by the different feature map ((b) and ©). During training, the boxes localization are modified to best match the ground truth. Source: W. Liu and al. (2016)

The Non-Maximum Suppression method is also used at the end of the SSD model to keep the most relevant bounding boxes. The Hard-Negative Mining (HNM) is then used because a lot of negative boxes are still predicted. It consists in selecting only a subpart of these boxes during the training. The boxes are ordered by confidence and the top is selected depending on the ratio between the negative and the positive which is at most 1/3.

Others There are several other techniques like YOLO9000, YOLOv2, Neural Architecture Search Net (NASNet), Mask Region-based Convolutional Network (Mask R-CNN), YOLOv3

5 Guidelines

- You can pick any method of your choice and implement it and try to get decent results.
- You can also make use of any pre-trained models and fine-tune them.
- After you get decent results by implementing an existing technique, you can try out any novel modifications in the method to get improved results to maximize your project grade.
- Always remember the main motto is to learn, there are many freely available codes online to do this, but you should write your own code.
- Before selecting a method, please find out how long does it take for the network to train if you implement it.
- Towards the end of the quarter, the `ieng6` cluster will become very busy, slow at times and there might be connectivity issues. Please keep these things in mind and start early and also explore other alternatives like google co-lab (12 hours free GPU) etc.
- You are encouraged to implement classes similar to ones introduced in Assignment 3 (`nntools.py`) to structure and manage your project. Make sure you use checkpoints to save your model after every epoch so as to easily resume training in case of any issues.

6 Deliverables

You will have to provide the following

1. **A 10 page final report**:
   - 10 pages **MAX** including figures, tables and bibliography.
   - One column, font size: 10 points minimum, **PDF format**.
   - Use of Latex highly recommended (e.g., NIPS template).
   - Quality of figures matter (*Graph without caption or legend is void.)*
   - The report should contain at least the following:
     - Introduction. What is the targeted task? What are the challenges?
     - Description of the method: algorithm, architecture, equations, etc.
     - Experimental setting: dataset, training parameters, validation and testing procedure (data split, evolution of loss with number of iterations etc.)
     - Results: figures, tables, comparisons, successful cases and failures.
     - Discussion: What did you learn? What were the difficulties? What could be improved?
     - Bibliography.
2. **Link to a Git repository** (such as GitHub, BitBucket, etc) containing at least:

- Python codes (using Python 3). You can use PyTorch, TensorFlow, Keras, etc.
- A jupyter notebook file to rerun the training (if any),
  - → We will look at it but we will probably not run this code (running time is not restricted).
- Jupyter notebook file for demonstration,
  - → We will run this on UCSD DSMLP (running time 3min max).

    This is a demo that must produce at least one illustration showing how well your model solved the target task. For example, if your task is classification, this notebook can just load one single testing image, load the learned model, display the image, and print the predicted class label. This notebook does not have to reproduce all experiments/illustrations of the report. This does not have to evaluate your model on a large testing set.
- As many jupyter notebook file(s) for whatever experiments (optional but recommended)
  - → We will probably not run these codes, but we may (running time is not restricted).

    These notebooks can be used to reproduce any of the experiments described in the report, to evaluate your model on a large testing set, etc.
- Data: learned networks, assets, . . . (**5Gb max**)
- **README file** describing:
  - the organization of the code (all of the above), and
  - if any packages need to be `pip` installed.
  - Example:

```
Description
===========
This is project FOO developed by team BAR composed of John Doe, ...

Requirements
============
Install package 'imageio' as follow:

        $ pip install --user imageio

Code organization
=================
demo.ipynb        --  Run a demo of our code (reproduces Figure 3 of our report)
train.ipynb       --  Run the training of our model (as described in Section 2)
attack.ipynb      --  Run the adversarial attack as described in Section 3
code/backprop.py  --  Module implementing backprop
code/visu.py      --  Module for visualizing our dataset
assets/model.dat  --  Our model trained as described in Section 4
```

7 Grading and submission

The grading policy and submission procedure will be detailed later.