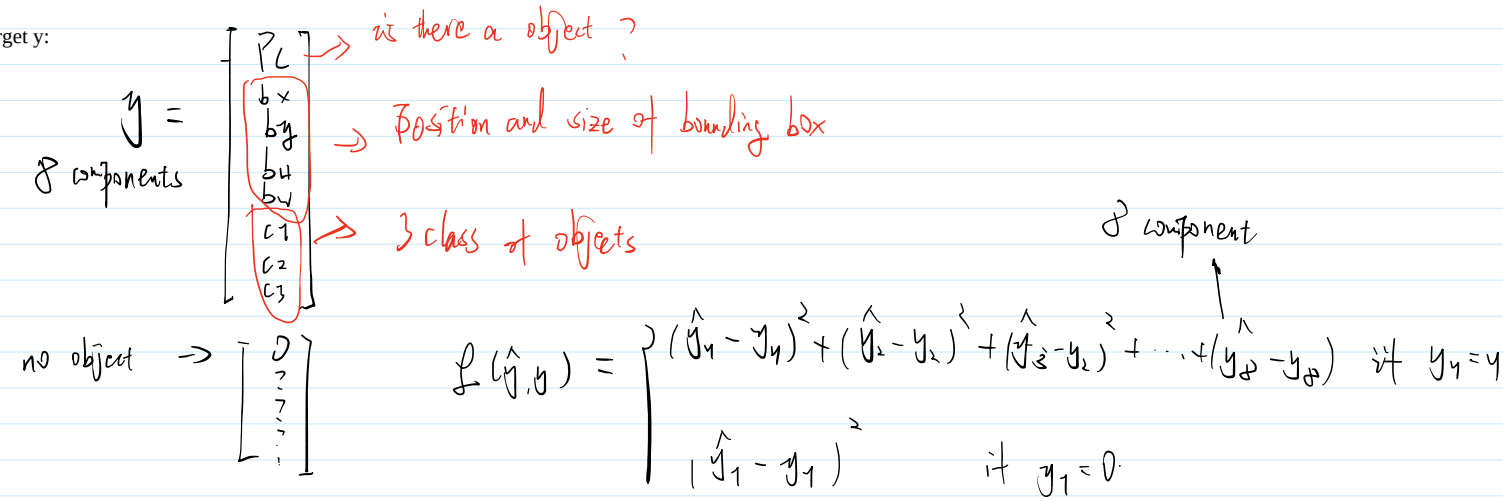
Detection Algorithms

Dienstag, 21. Juli 2020 14:00

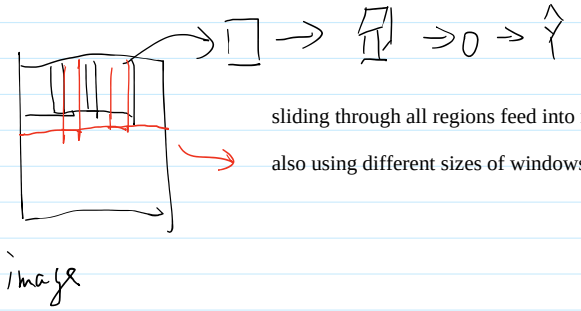
Object Localization

target y:



Object detection

conventional method: sliding window

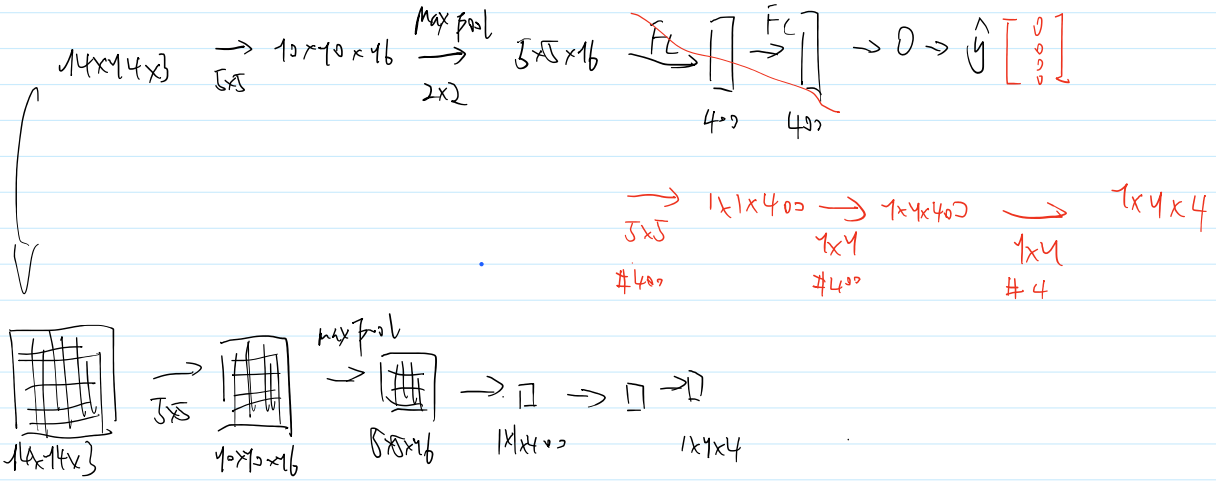


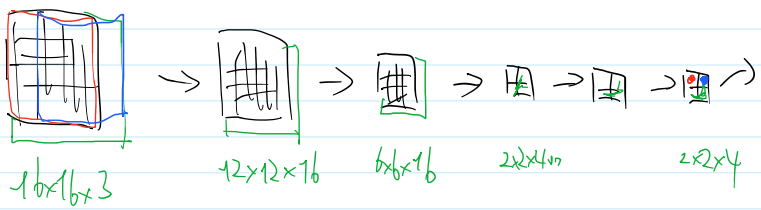
sliding through all regions feed into network to predict.

also using different sizes of windows

convolutional implementation of sliding window

Turing FC layer into convolutional layer



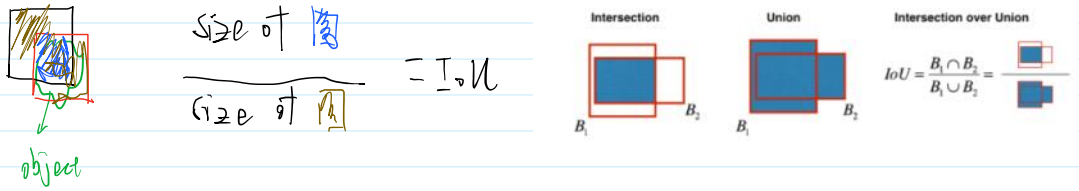


instead of running propagation on four subsets of the input image (sliding windows) independently, Instead, it combines all four into one form of computation and shares a lot of the computation in the regions of image that are common. -> turn four propagations into one propagation -> much faster

To the bounding box

↳ assign the bounding box to the cell has the center of the object

Intersection Over Union -- to evaluate how accurate the bounding box is



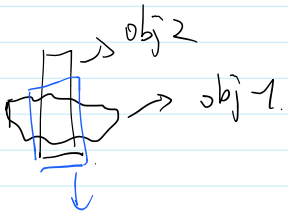
None-max suppression

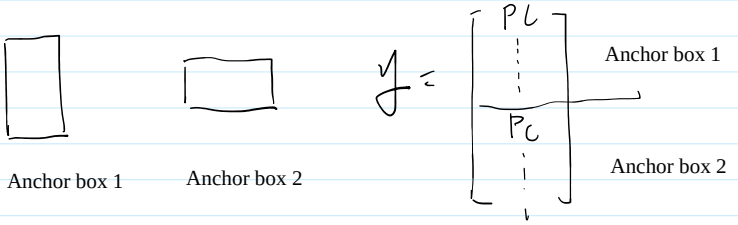
when for one object multiple bounding boxes are predicted

- remove all boxes Pc < 0.6
- while any remaining boxes
  - pick boxes with largest Pc output as prediction
  - discard any remaining boxes with IoU > 0.5 with previous box

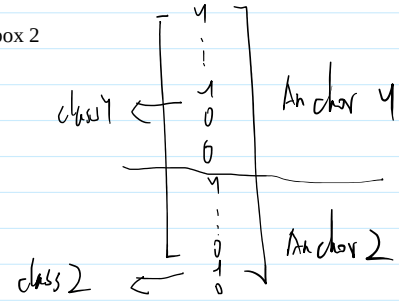
Anker box

when in single grid cell there are multiple object of different shapes detected





for this grid cell, it has too objects, the output y label will be:



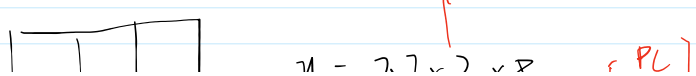
without anchor box, for each object in the training set and the training set image, it was assigned to the grid cell that corresponds to that object's midpoint.

Now, each object is assigned to the same grid cell as before, assigned to the grid cell that contains the object's midpoint, but it is assigned to a grid cell and anchor box with the highest IoU with the object's shape.

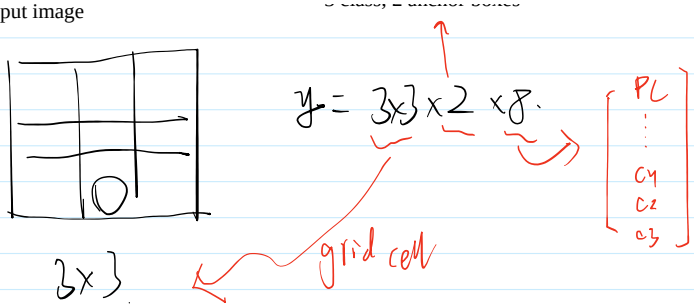
YOLO

one input image

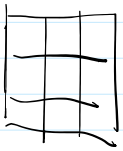
3 class, 2 anchor boxes



one input image



applying non max suppressed outputs



- using two anchor boxes, then for each of the non-grid cells, get two predicted bounding boxes. Some of them will have very low probability, very low Pc, but still get two predicted bounding boxes for each of the nine grid cells. So those are the bounding boxes you get.
- Next, get rid of the low probability predictions.
- for each of classes, independently run non-max suppression for the objects that were predicted to come from that class.

- **R-CNN -- segmentation algorithms**

  Fast R-CNN -- convolutional implementation of sliding windows

  Faster R-CNN -- region proposal using NN

Take away from programming assignments:

- YOLO (you only look once): requires only one forward propagation pass through the network to make predictions. After non-max suppression, it then outputs recognized objects together with the bounding boxes.
- The input of YOLO is a batch of images of shape (m, n_h, n_w, n_channel). The output is a list of bounding boxes with recognized classes. Each bounding box is represented by 6 numbers (pc,bx,by,bh,bw,c). If you expand cc into an 80-dimensional vector, each bounding box is then represented by 85 numbers.
- Anchor boxes are chosen by exploring the training data to choose reasonable height/width ratios that represent the different classes.
- The YOLO architecture is: IMAGE (m, n_H, n_W, 3) -> DEEP CNN -> ENCODING (m, n_h, n_w, #anchors, #classes+5).
- If the center/midpoint of an object falls into a grid cell, that grid cell is responsible for detecting that object.
- for each box (of each cell) we will compute the following element-wise product and extract a probability that the box contains a certain class.
- The class score is:
  - scores = pc * ci, meaning the probability that there is an object pc times the probability that the object is a certain class ci.
- So, one idea is:
  - For each of the grid cells, find the maximum of the probability scores (taking a max across all the classes, one maximum for each of all the anchor boxes).
  - Color that grid cell according to what object that grid cell considers the most likely with corresponding color.
  - For instance, if the current grid cell has the maximal likelihood as the class "sky", mark the gird cell with the color assigned to the class "sky"
- You can also draw bounding boxes to visualize the prediction:
  - each cell gives out #anchors boxes, in total there will be #grid_h * #grid_w * #anchorboxes bounding boxes.
  - Then conduct non-max suppression, namely:
    - Get rid of boxes with a low score, meaning, the box is not very confident about detecting a class; either due to the low probability of any object, or low probability of this particular class.
    - Select only one box when several boxes overlap with each other and detect the same object.

- As result, there will be less bounding boxes presented in the image.

Summary for YOLO:

- Input image (608, 608, 3)
- The input image goes through a CNN, resulting in a (19,19,5,85) dimensional output.
- After flattening the last two dimensions, the output is a volume of shape (19, 19, 425):
  - Each cell in a 19x19 grid over the input image gives 425 numbers.
  - 425 = 5 x 85 because each cell contains predictions for 5 boxes, corresponding to 5 anchor boxes, as seen in lecture.
  - 85 = 5 + 80 where 5 is because $(p_c, b_x, b_y, b_h, b_w)$ has 5 numbers, and 80 is the number of classes we'd like to detect
- You then select only few boxes based on:
  - Score-thresholding: throw away boxes that have detected a class with a score less than the threshold
  - Non-max suppression: Compute the Intersection over Union and avoid selecting overlapping boxes
- This gives you YOLO's final output.

- YOLO is a state-of-the-art object detection model that is fast and accurate
- It runs an input image through a CNN which outputs a 19x19x5x85 dimensional volume.
- The encoding can be seen as a grid where each of the 19x19 cells contains information about 5 boxes.
- You filter through all the boxes using non-max suppression. Specifically:
  - Score thresholding on the probability of detecting a class to keep only accurate (high probability) boxes
  - Intersection over Union (IoU) thresholding to eliminate overlapping boxes
- Because training a YOLO model from randomly initialized weights is non-trivial and requires a large dataset as well as lot of computation, we used previously trained model parameters in this exercise. If you wish, you can also try fine-tuning the YOLO model with your own dataset, though this would be a fairly non-trivial exercise.