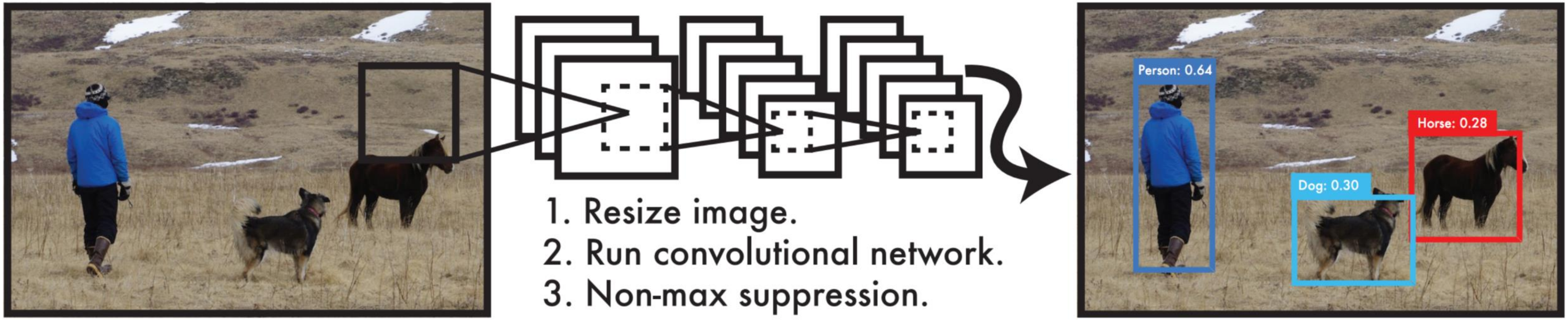
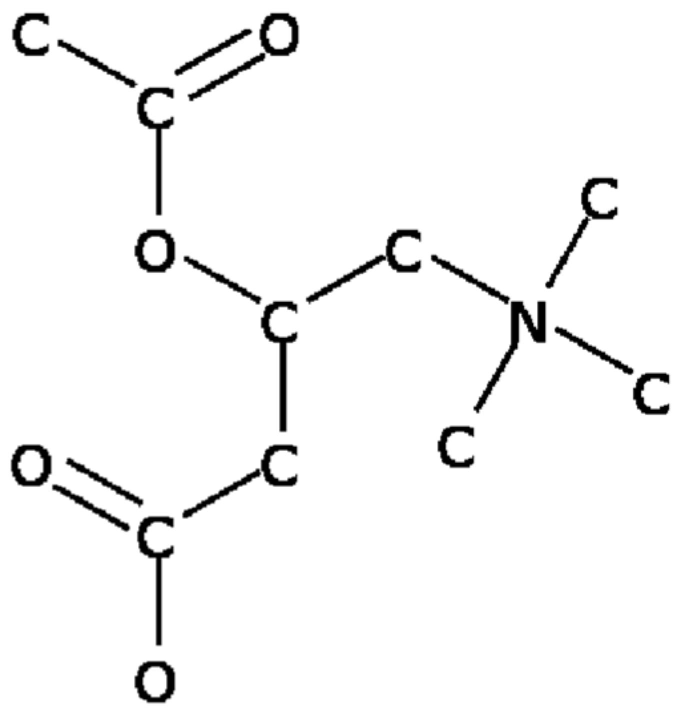


Object Detection - YOLO

Literature Sharing: Xiaokang Guo

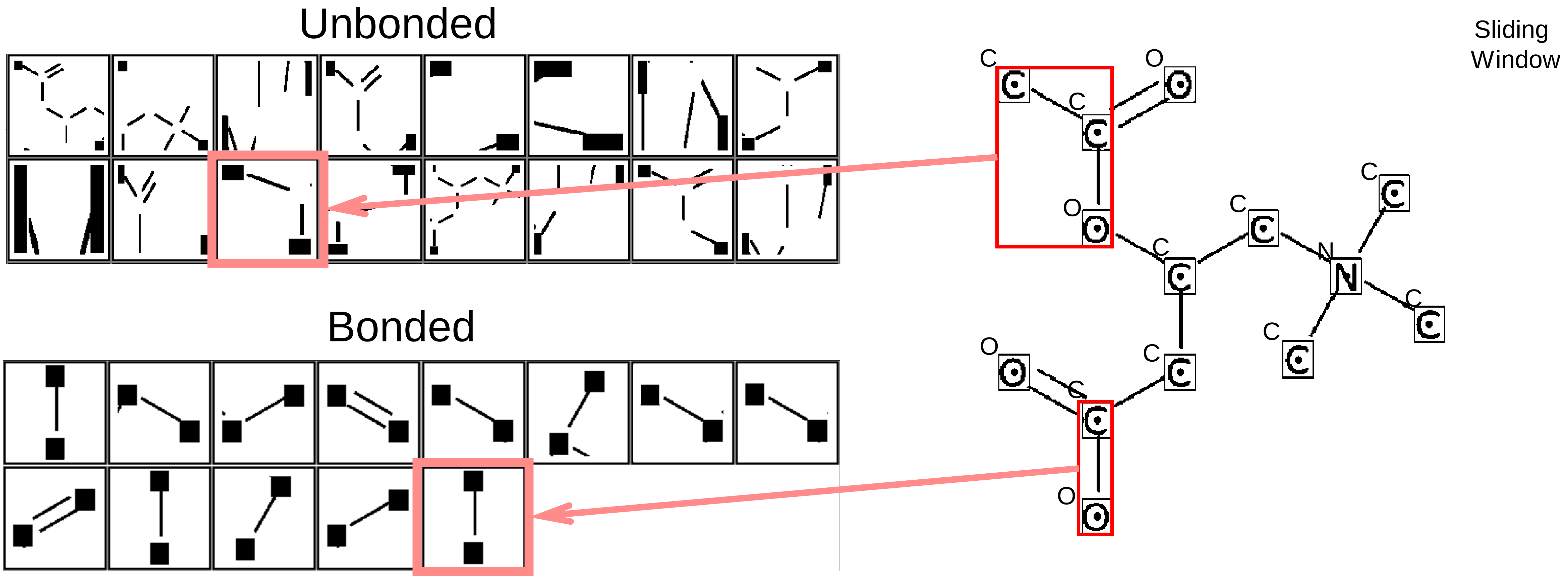
Term Project

- Get SMILES
  - C-C(=O)-O-C(-C-C(=O)-O)-C-N(-C)(-C)-C
  - Presence of Atom → Character
  - Atom Type → C, N, O, S, …
  - Presence of Bond → Symbol
  - Bond Type → = or #



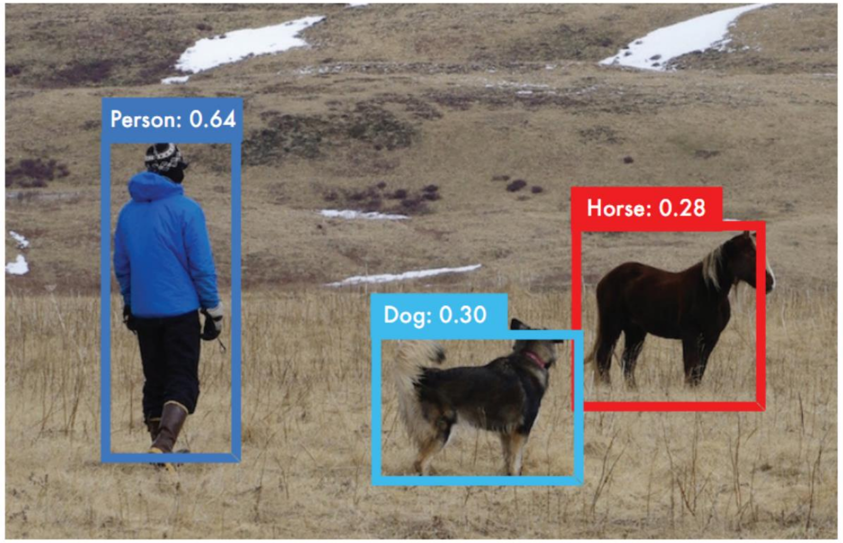
Term Project

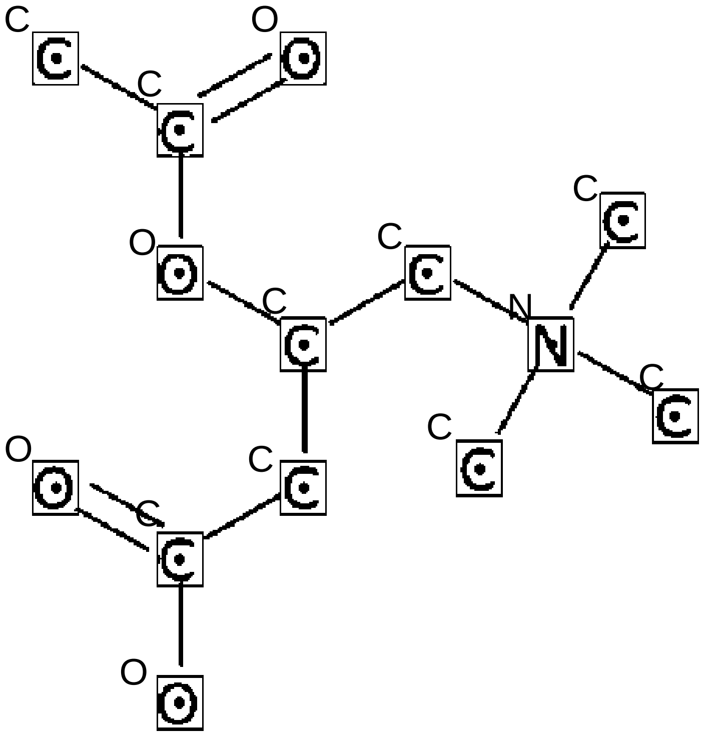
Presence of Atom
Atom Type
Presence of Bond
Bond Type



Capability of Object Detector

- Add bounding box and class identifier simultaneously





You Only Look Once Version 1

- Fast end-to-end CNN
  - Fast enough for real-time detection
  - One shot object detector
- Other mentioned models ?
  - Region CNN
  - Fast R-CNN
  - Faster R-CNN

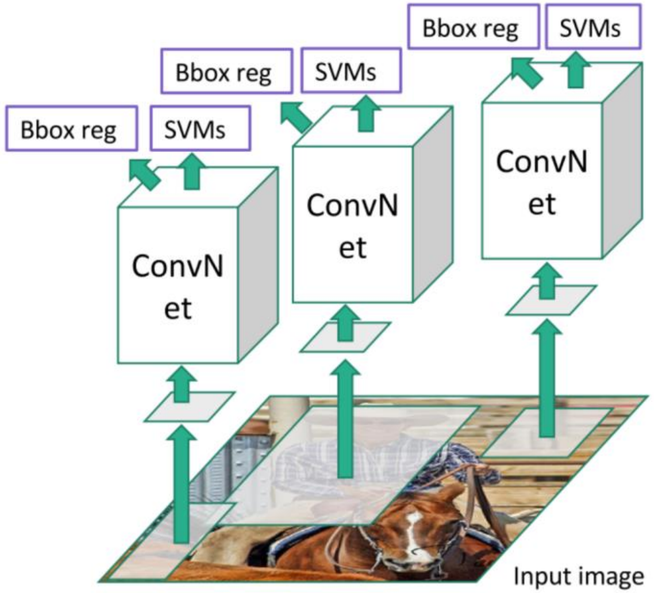
Region Proposal

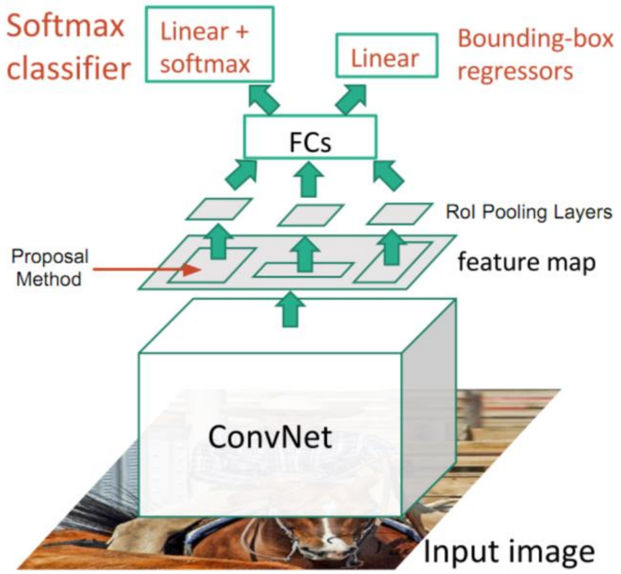
- Selective Search (SS):
  1. Generate initial sub-segmentation, we generate many candidate regions
  2. Use greedy algorithm to recursively combine similar regions into larger ones
  3. Use the generated regions to produce the final candidate region proposals

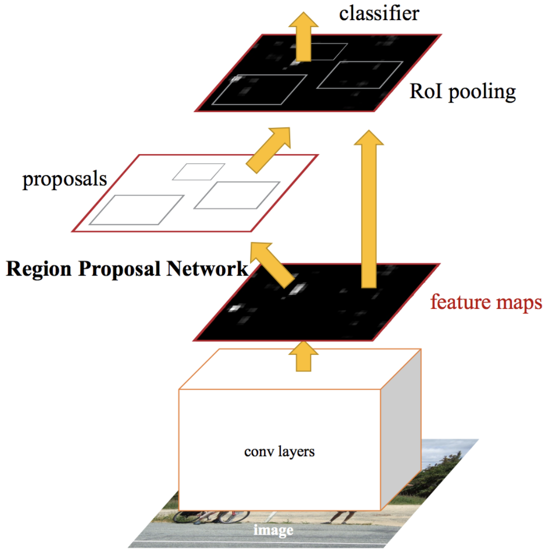
(a) (b)

R-CNN / Fast R-CNN / Faster R-CNN

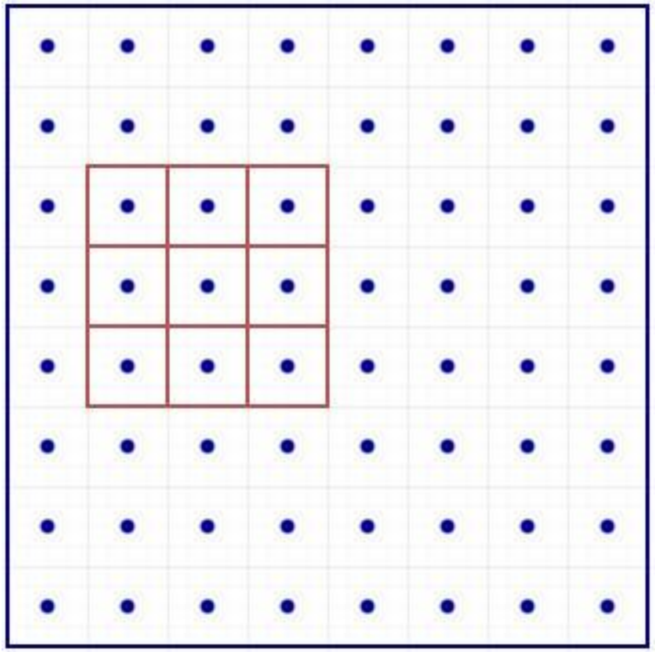
- Image → SS → Region Proposal (2k) → CNN → Features → SVM → Objectness
    - 47s / image with VGG16 [Simonyan & Zisserman. ICLR15]
- Image → ConvL → Features → SS → Proposal → RoI Pooling → FCL → Result
    - Around 1 second per image
- Proposal Method $\Longrightarrow$ Region Proposal Network



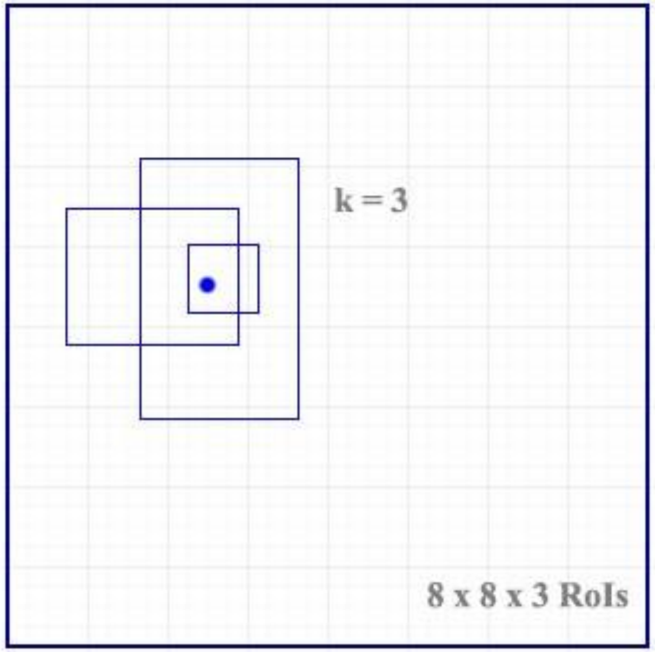




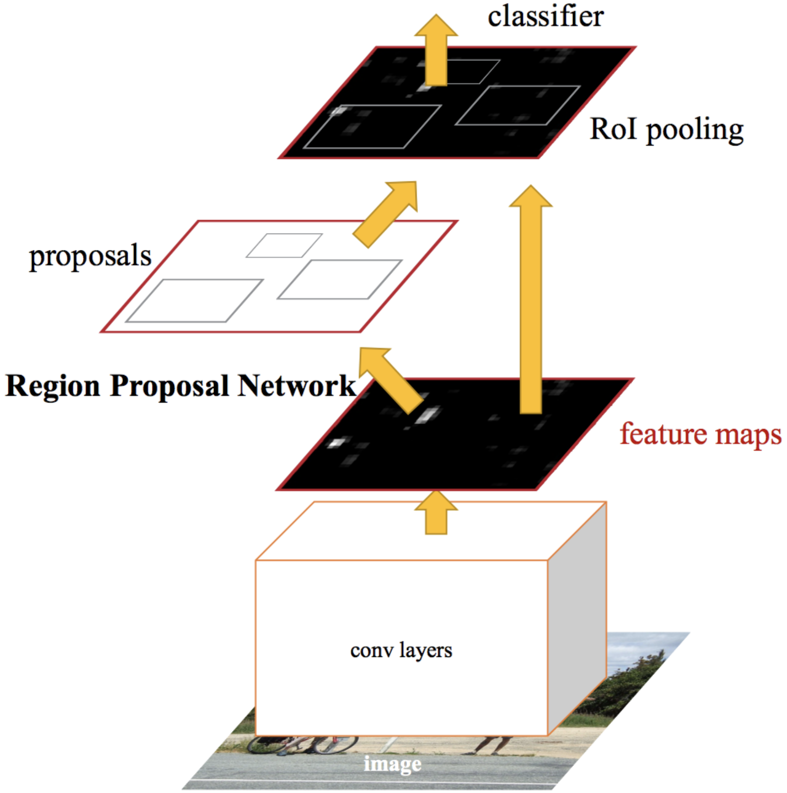
Region Proposal Network



8 x 8 feature maps



3 proposals for each location



You Only Look Once

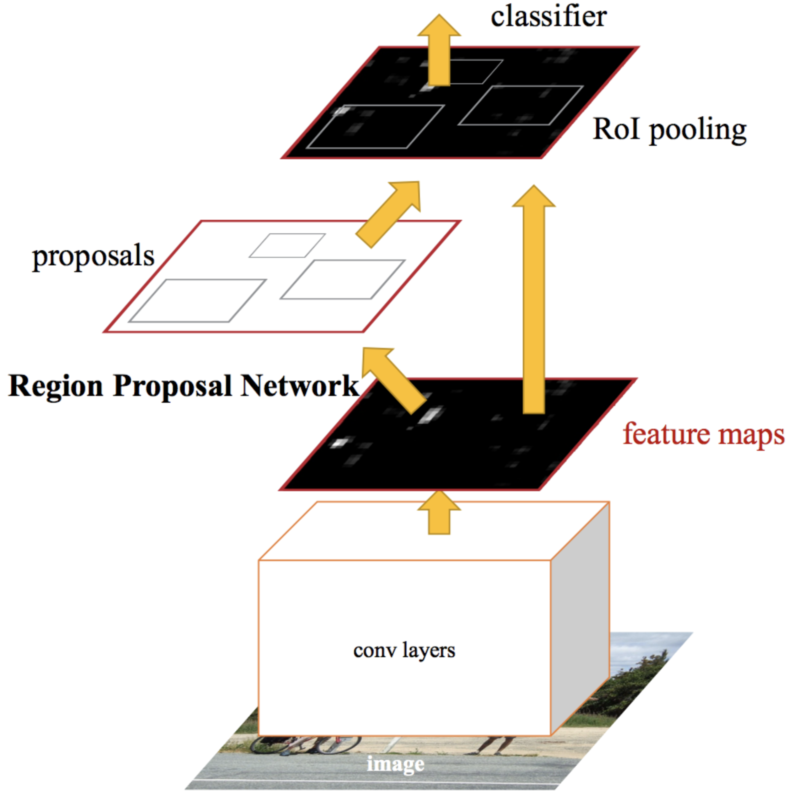
- Arbitrary Size

You Only Look Once

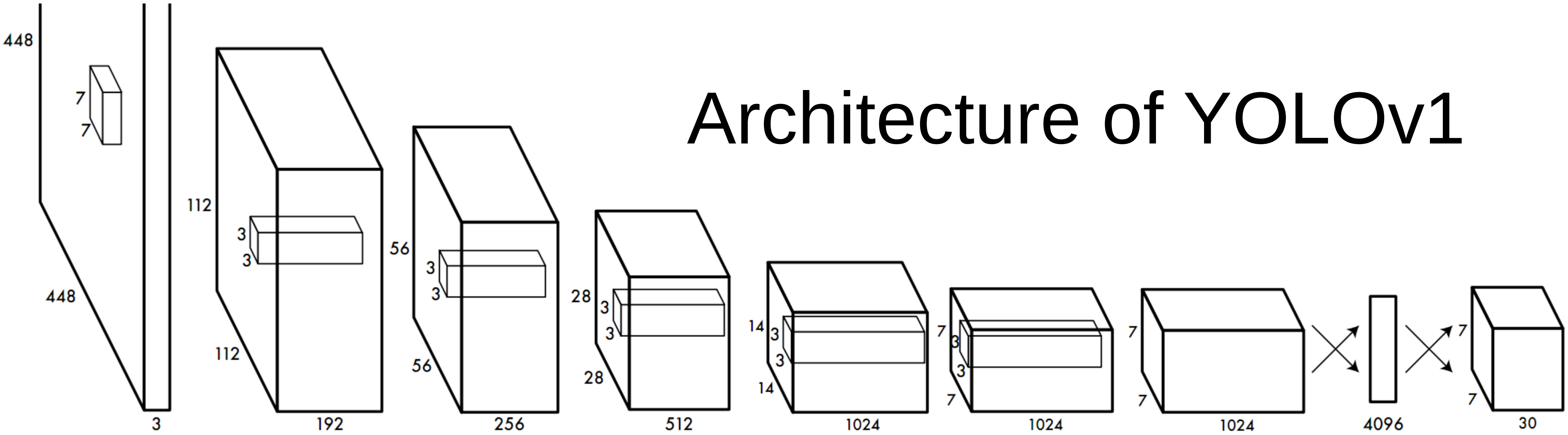
- Fixed Size (448px×448px)

Capability

- Localized detector which might be responsible for the whole image



Architecture of YOLOv1



	Conv 1	Max Pool 1	Conv 2	Max Pool 2
Filters	(7,7,64,2)	(2,2,2)	(3,3,192)	(2,2,2)
Output	224×224×64	112×112×64	112×112×192	56×56×192

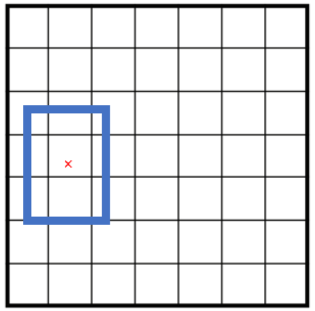
Conv 3	Conv 4	Conv 5	Conv 6	Max Pool 3
(1,1,128)	(3,3,256)	(1,1,256)	(1,1,512)	(2,2,2)
56×56×128	56×56×256	56×56×256	56×56×512	28×28×512

	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	P4
Filters	(1,1,256)	(3,3,512)	(1,1,256)	(3,3,512)	(1,1,256)	(3,3,512)	(1,1,256)	(3,3,512)	(1,1,512)	(3,3,1024)	(2,2,2)
Output	28×28×256	28×28×512	28×28×256	28×28×512	28×28×256	28×28×512	28×28×256	28×28×512	28×28×512	28×28×1024	14×14×1024

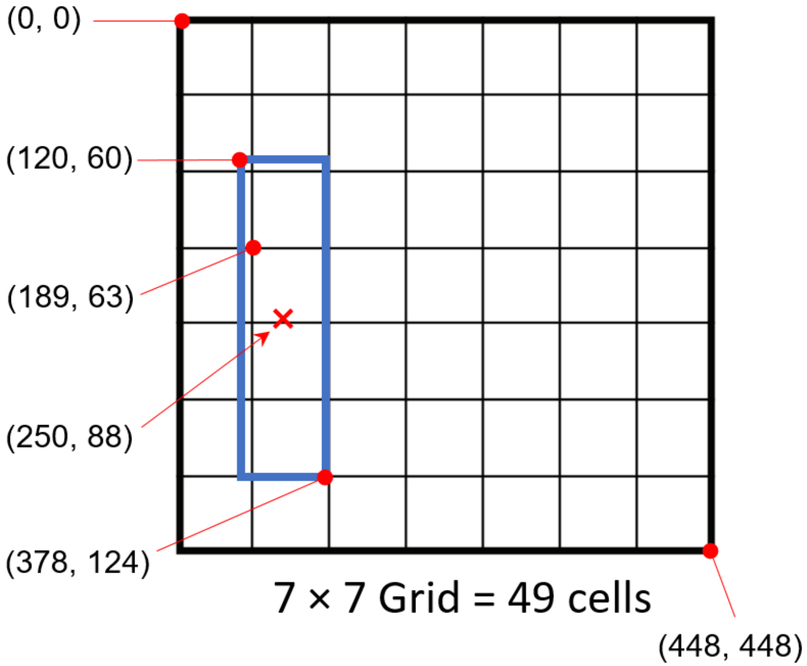
	C17	C18	C19	C20	C21	C22	C23	C24	F1
Filters	(1,1,512)	(3,3,1024)	(1,1,512)	(3,3,1024)	(3,3,1024)	(3,3,1024,2)	(3,3,1024)	(3,3,1024)	-
Output	14×14×512	14×14×1024	14×14×512	14×14×1024	14×14×1024	7×7×1024	7×7×1024	7×7×1024	4096

F2
-
7×7×30

Output Layer



- Grid (7×7), Cells (3,1), Bounding boxes (2), Prediction Vector (30)



One predictions vector for each cell (49 cells in total):

x_1,y_1,w_1,h_1,c_1 , x_2,y_2,w_2,h_2,c_2 , $C_1,C_2,C_3,\ldots,C_{20}$ PASCAL VOC

confidence score c_i = Pr(Obj) $*$ IOU $^{\text{truth}}_{\text{pred}}$ Objectness and accuracy

conditional class probability C_n = Pr(Class_n | Obj) = p_i(n)

The prob of belonging to *n*-th class if an obj is in *i*-th cell.

First Bounding Box

x_1 = (250-189)/63 = 0.968

y_1 = (85-63)/63 = 0.349

w_1 = (124-60)/448 = 0.143

h1 = (378-120)/448 = 0.576

At test time, for each box the confidence score times with conditional class probability gives class-specific confidence scores: $c_i * C_n = \Pr(\text{Class}_n \mid \text{Obj}) * \Pr(\text{Obj}) * \text{IOU}^{\text{truth}}_{\text{pred}}$

Loss Function

x_1, y_1, w_1, h_1, c_1 , x_2, y_2, w_2, h_2, c_2 , $C_1, C_2, C_3, \ldots, C_{20}$

$$\boldsymbol{RSS} = \lambda_{\boldsymbol{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{\boldsymbol{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (c_i - \hat{c}_i)^2 + \lambda_{\text{no\_obj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{no\_obj} (c_i - \hat{c}_i)^2 + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

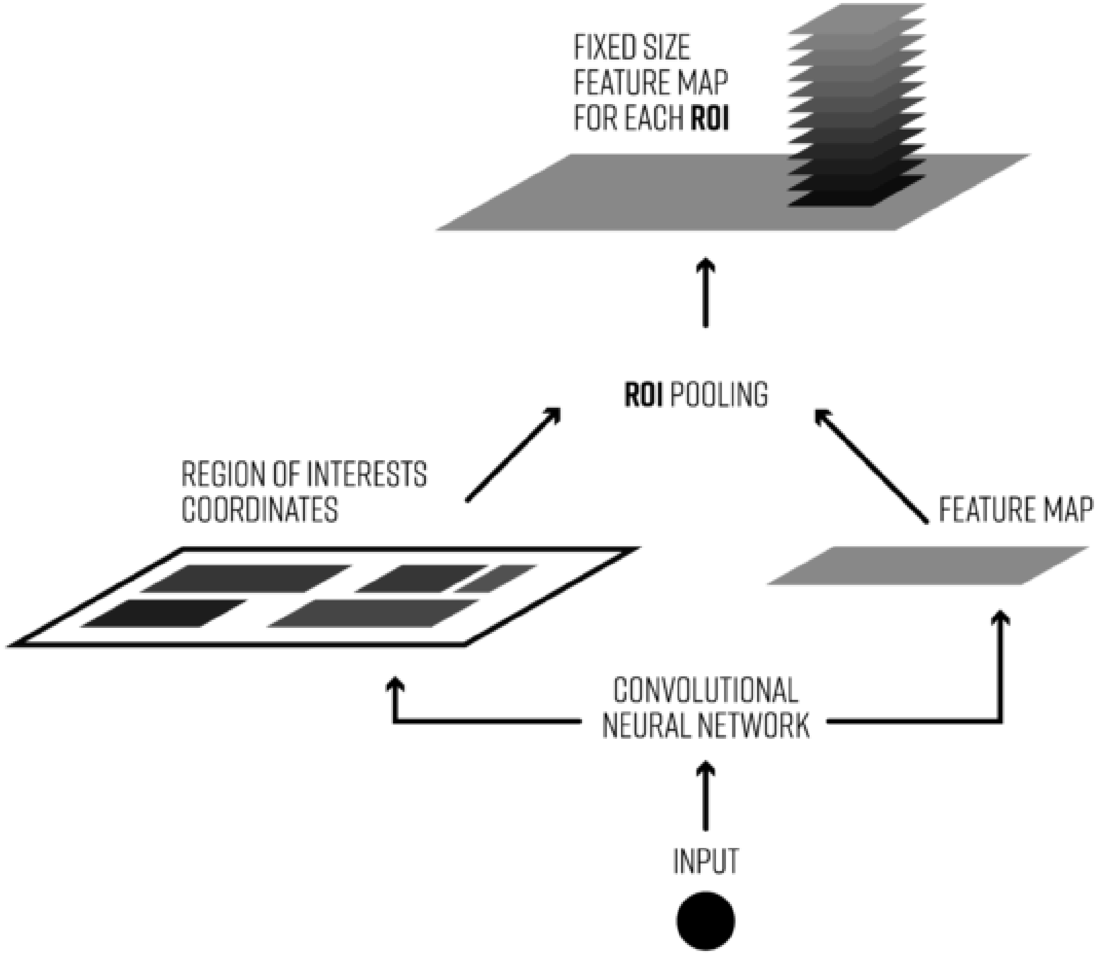
λ_{coord} = 5, λ_{no_obj} = 0.5, $\mathbb{1}_{ij}^{obj} = \begin{cases} 1 & \text{,obj in} \\ 0 & \text{,obj out} \end{cases}$, $\mathbb{1}_{ij}^{no\_obj} = \begin{cases} 0 & \text{,obj in} \\ 1 & \text{,obj out} \end{cases}$

References

- YOLO v1 Original Paper
  - https://arxiv.org/abs/1506.02640
- Understanding YOLO
  - https://hackernoon.com/understanding-yolo-f5a74bbc7967
- Selective Search
  - https://www.koen.me/research/pub/uijlings-ijcv2013-draft.pdf
- R-CNN, Fast R-CNN, Faster R-CNN, YOLO – Object Detection Algorithm
  - https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
- RPN
  - https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9
- F.F. Li et., CS231n, Lecture 11, Stanford, 2017
  - http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

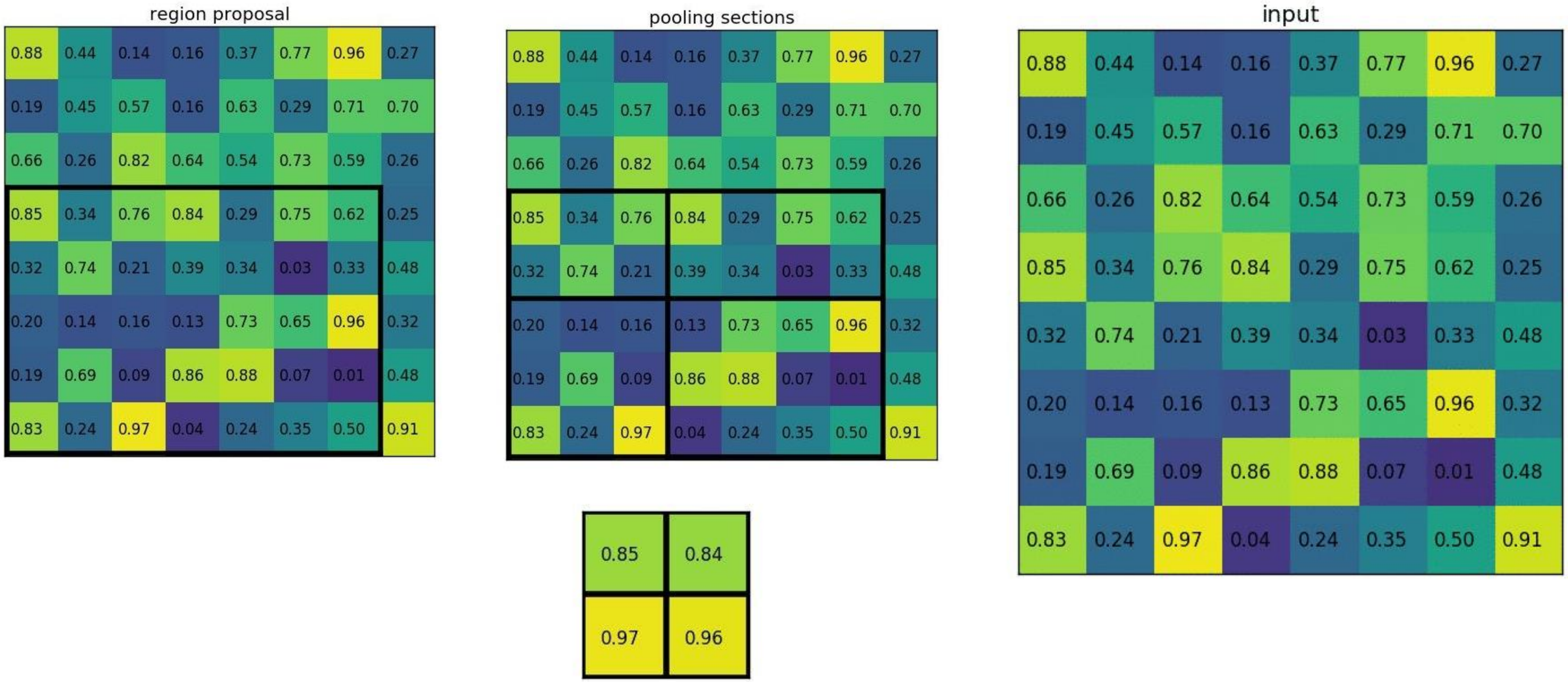
Thanks for your attention!

RoI Pooling in Fast R-CNN



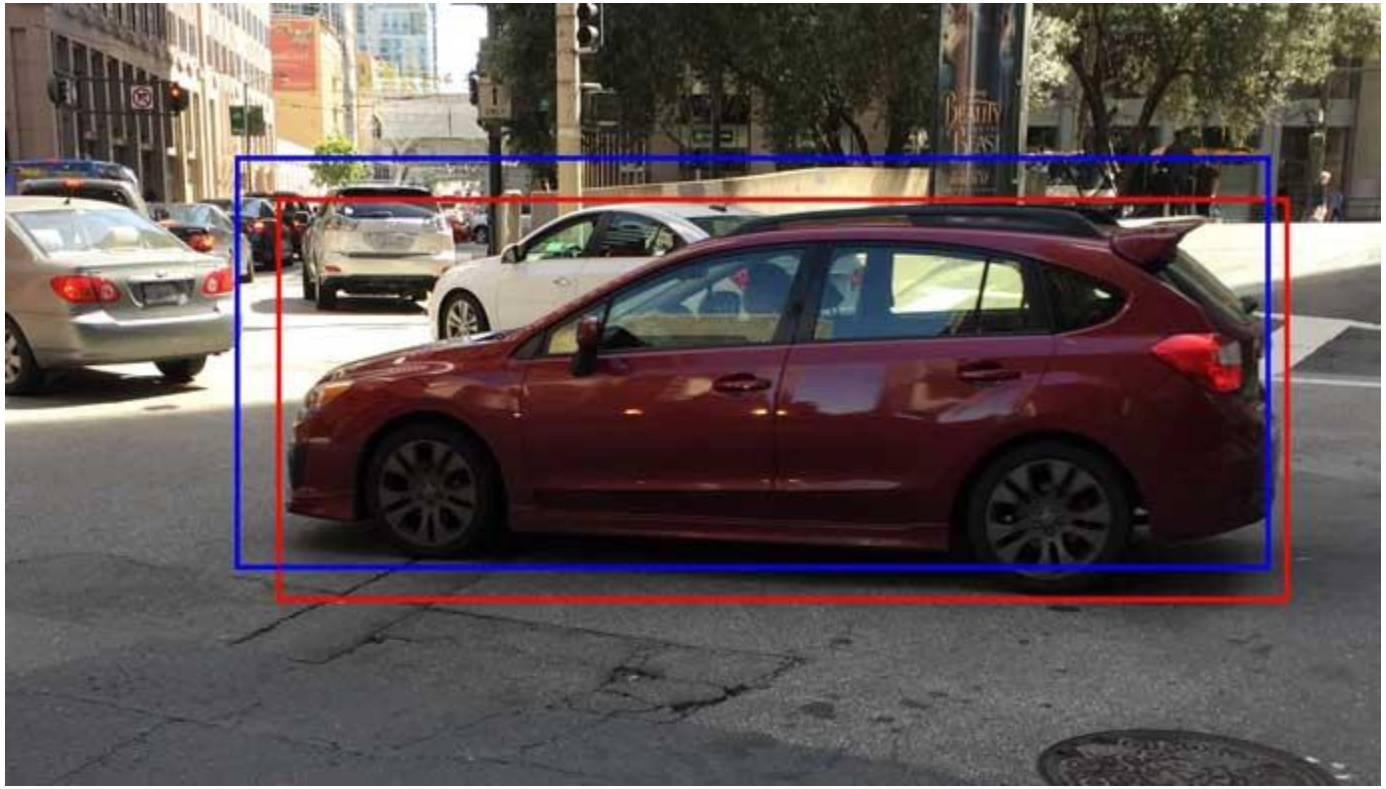
https://deepsense.ai/region-of-interest-pooling-explained/

Region of Interest Pooling Layer



https://deepsense.ai/region-of-interest-pooling-explained/

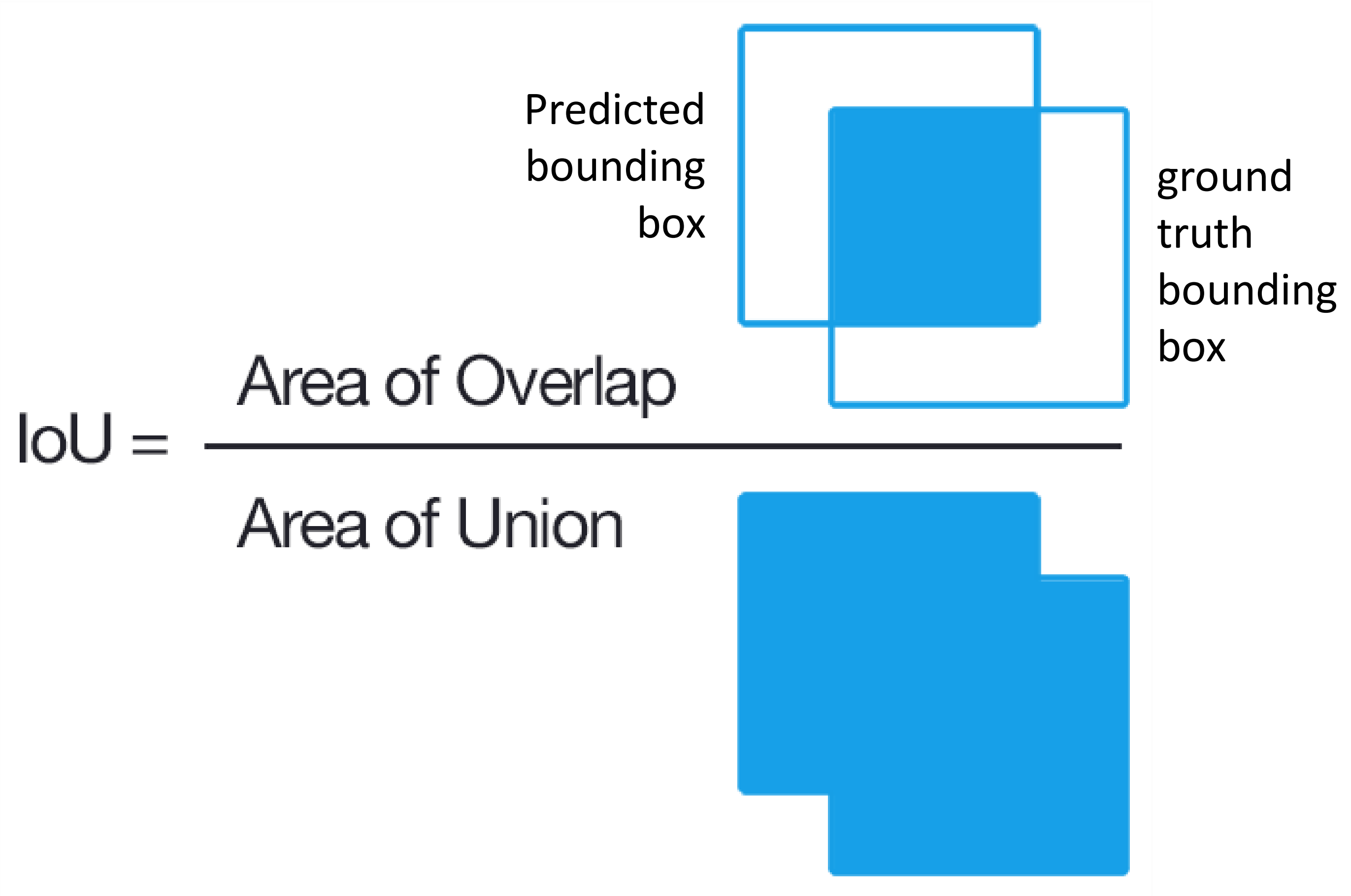
BBox Regressor



Use regression to refine the original ROI in blue to the red one

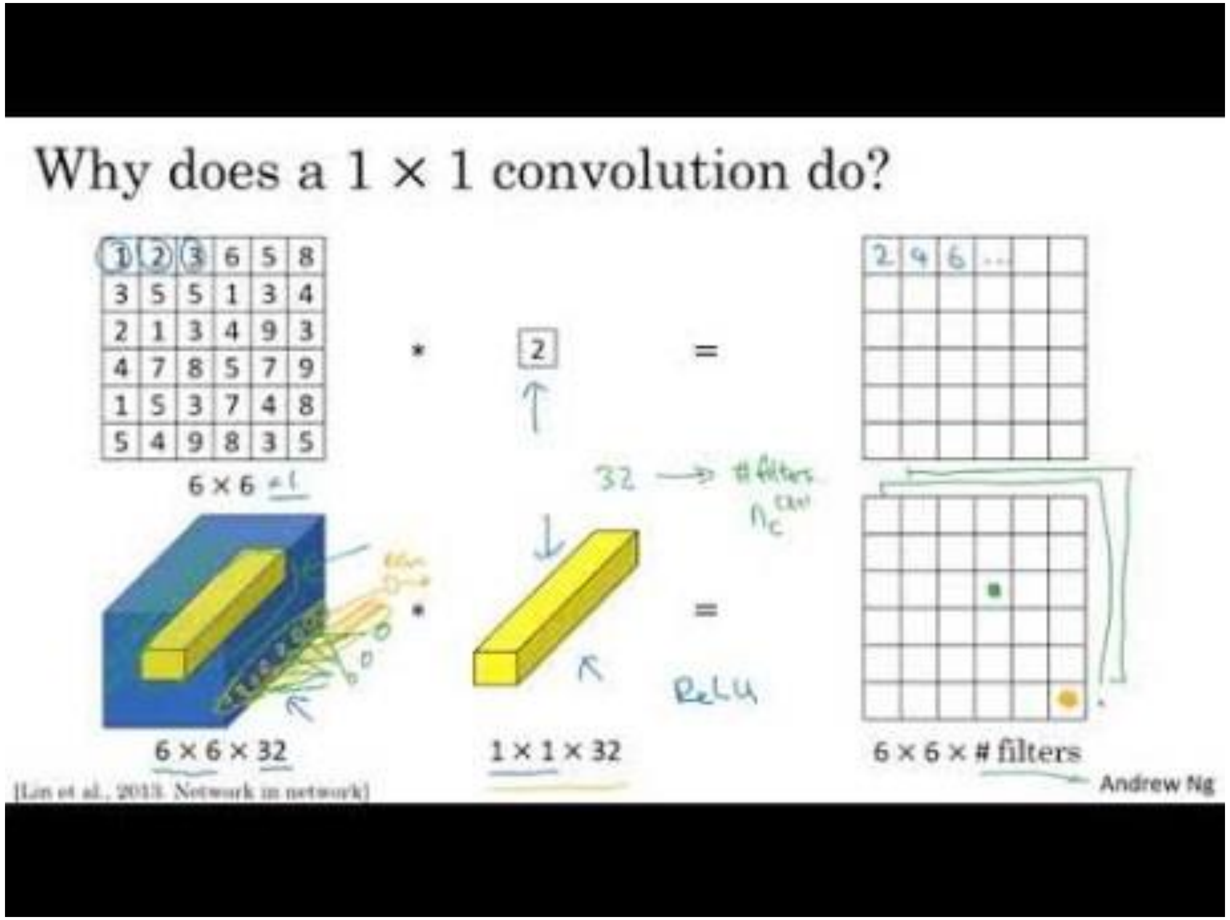
https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9

Intersection over union



https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

1×1 Convolutional Layer



Andrew Ng

https://www.youtube.com/watch?v=vcp0XvDAX68

Train Strategy

- The authors describe the training in the following way
  - First, pretrain the first 20 convolutional layers using the ImageNet 1000-class competition dataset, using a input size of 224x224
  - Then, increase the input resolution to 448x448
  - Train the full network for about 135 epochs using a batch size of 64, momentum of 0.9 and decay of 0.0005
  - Learning rate schedule: for the first epochs, the learning rate was slowly raised from 0.001 to 0.01. Train for about 75 epochs and then start decreasing it.
  - Use data augmentation with random scaling and translations, and randomly adjusting exposure and saturation.