S14-S15

Due Sunday by 11:59pm **Points** None **Available** Apr 26 at 10am - May 3 at 11:59pm 8 days

.

RCNN FAMILY

The story of RCNN actually starts from this paper: **Selective Search for Object Recognition (https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf)**

SSOR addresses the problem of generating possible object locations for use in object recognition.
It introduced Selective Search which combines the strength of both an exhaustive search and segmentation.
Like segmentation, it use the image structure to guide their sampling strategy.

But, look at this image below:

There is a high variety of reasons that an image region forms an object.

In the cats can be distinguished by color, not texture.

In the chameleon can be distinguished from the surrounding leaves by texture, not color.

In the wheels can be part of the car because they are enclosed, not because they are similar in texture or color.

Therefore, to find objects in a structured way it is necessary to use a variety of diverse strategies. Furthermore, an image is intrinsically hierarchical as there is no single
scale for which the complete table, salad bowl, and spoon can be found in a).

SSOR, in turn, depends on this paper: **Efficient Graph-based Image Segmentation (http://people.cs.uchicago.edu/~pff/papers/seg-ijcv.pdf)** to find it's first proposals.

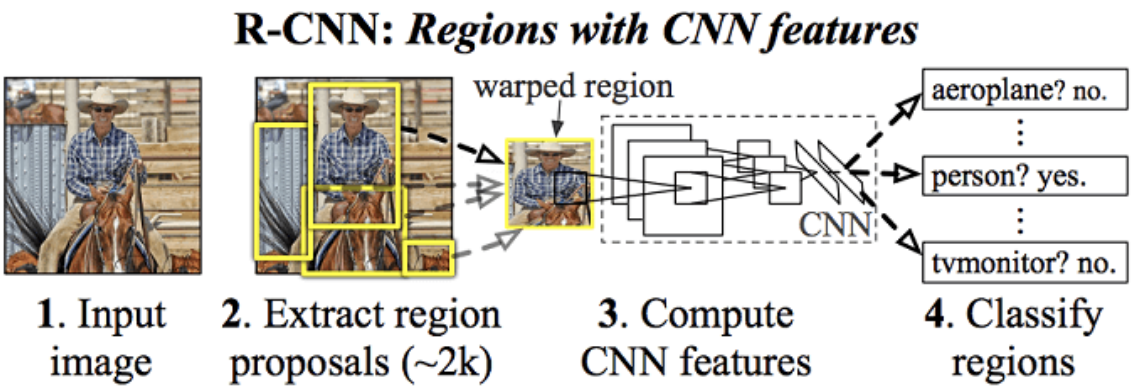
SSOR algorithms goes as follows:

1. it first uses EGIS to create initial regions
2. it then uses a greedy algorithm to interactively group similar regions together.
    1. first, the similarities between all neighboring regions are calculated. The two most similar regions are grouped together, and the new similarities are calculated between the resulting region and its neighbor.
    2. the process of grouping the most similar regions is repeated until the whole image is covered.
3. The selection criterion of these regions is dependent on a lot of strategies including finding Complementary Color Spaces. The following color spaces with an increasing degree of invariance is listed below
    1. R
    2. G
    3. B
    4. I (greyscale intensity)
    5. **LAB (http://www.photoshopbuzz.com/what-is-lab-color-photoshop/)** (Lightness Channel, A-Channel (representing green-red) and B channel (Blue-yellow)
    6. r (normalized R channel)
    7. g (normalized G channel)

8. HSV

4. Simply put it was an advanced Computer Vision approach to segmentation.

R-CNN



Selective Search:

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

4. In order to compute the features for a region proposal, the image must be converted into a form compatible with CNN (227x227). The proposals are wrapped in a tight square and then fed to the CNN

5. Feed these 2000 regions to AlexNet and get a 4096-dimensional feature vector.

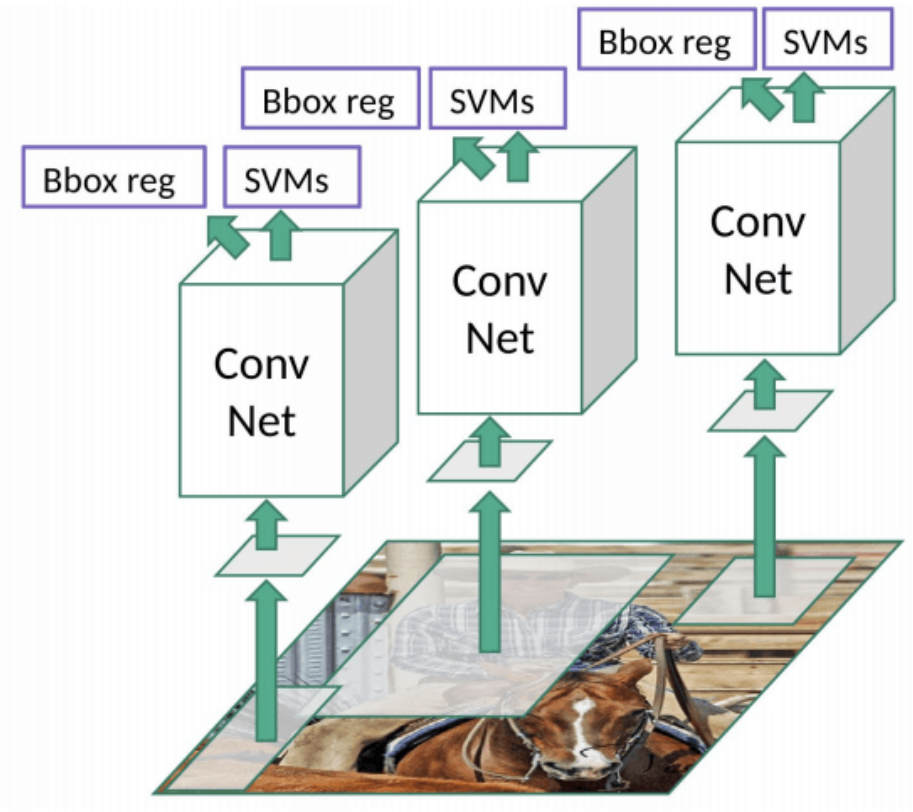
6. Then for each class, they score the extracted feature vector using the SVM trained for that class.

7. Given all scored regions in an image, it then applies a greedy NMS that rejects a region if it has an IOU overlap with a higher scoring selected region larger than a learned threshold.

RCNN limits the proposals to 2000.

These 2000 candidates region proposals are **warped into a square** and fed into a
convolution neural network (AlexNet) that produces a 4096-dimension vector.
We have just performed Feature extraction.

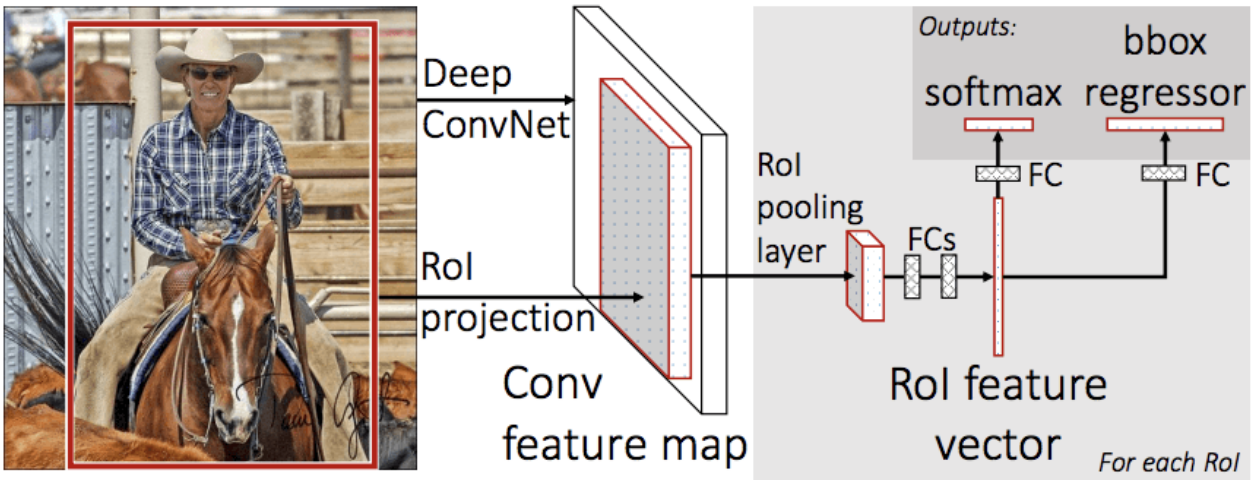
The extracted region are fed into an **SVM** (https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47) to classify the presence of the object within that candidate region proposals.



Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
- It cannot be implemented real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

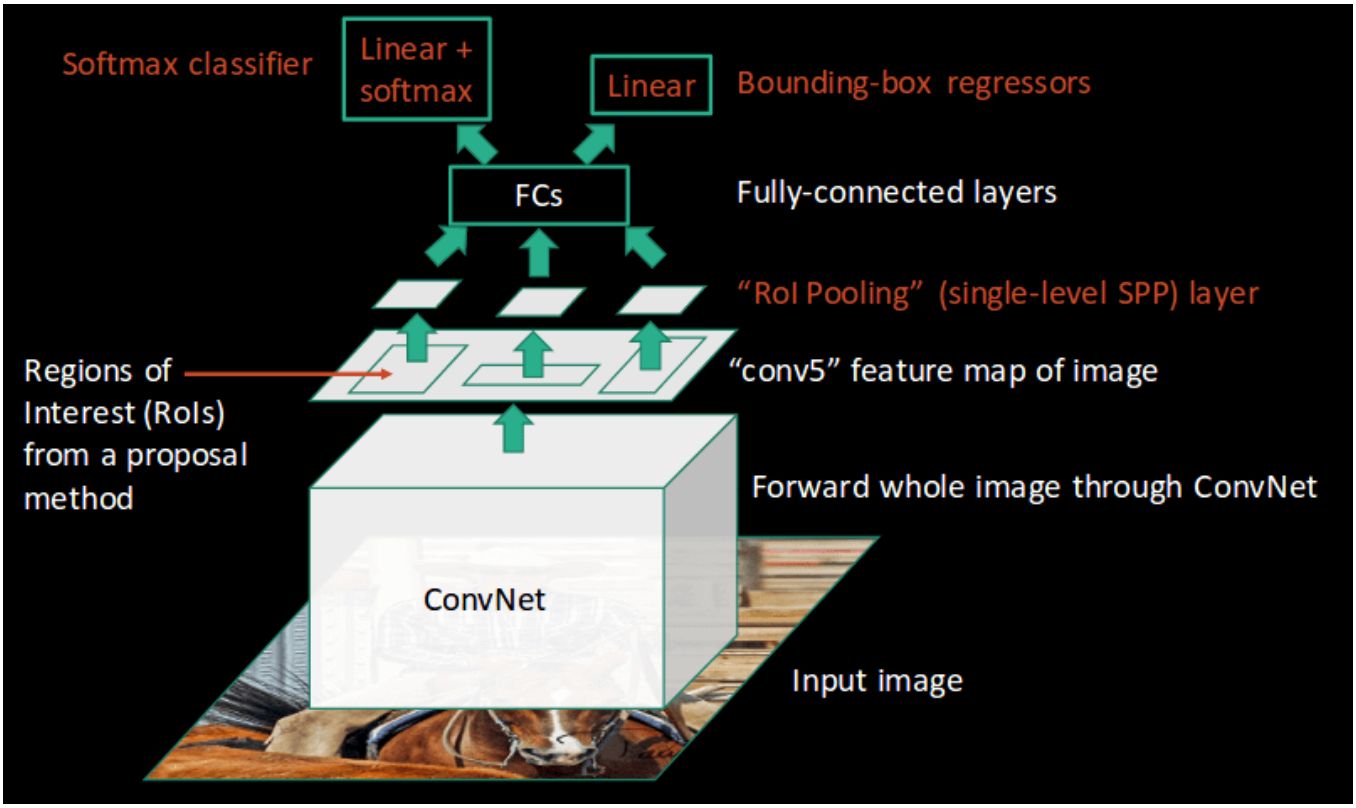
Fast R-CNN

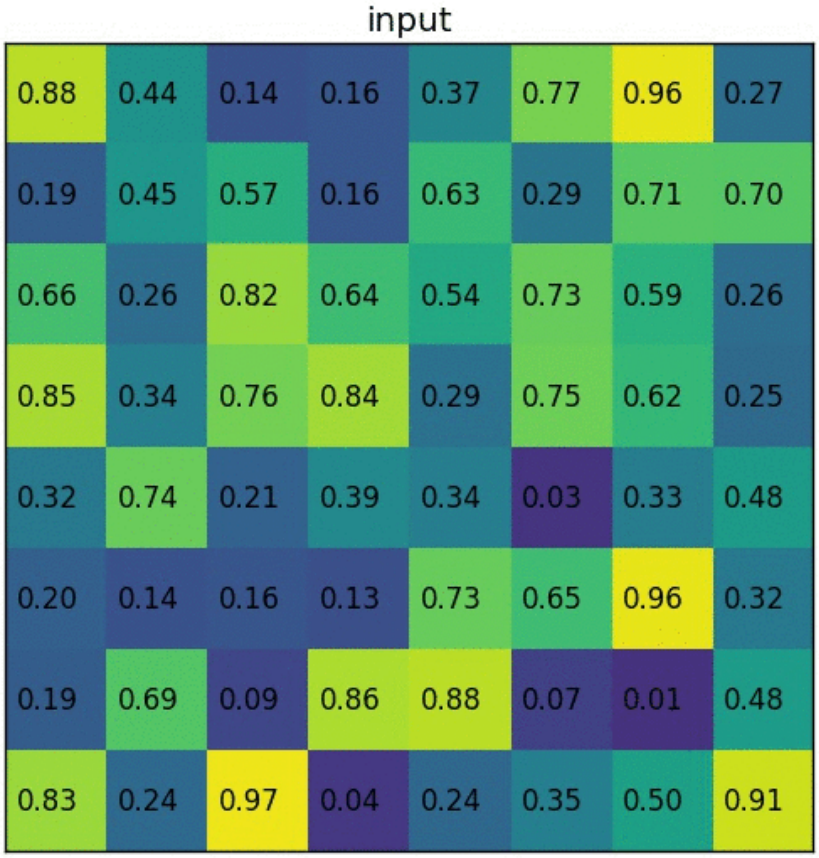


- The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map.
- Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map (7x7).
- Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers:
  - one that produces softmax probability estimates over K object classes plus a catch-all "background" class and

- another layer that outputs four real-valued numbers for each of the K object classes. Each set of 4 values encodes refined bounding-box positions for one of the K classes.

- The **RoI** Pooling layer uses maxpooling to convert the features inside any valid region of interest into a small feature map with resolution of 7x7.





Faster R-CNN

Both the above algos uses selective search to find out the region proposals.

Selective search is a slow and time-consuming process affecting the performance of the network.
Faster RCNN eliminated the selective search algorithm and lets the network learn the region proposals.

In this paper, we show that an algorithmic change— computing proposals with a deep

convolutional neural network—leads to an elegant and effective solution where proposal
computation is nearly cost-free given the detection network's computation.

To this end, we introduce novel **Region Proposal Networks** (RPNs) that share convolutional layers with state-of-the-art object detection networks. By sharing convolutions
at test-time, the marginal cost for computing proposals is small (e.g., 10ms per image).

Our observation is that the convolutional feature maps used by region-based detectors,
like Fast RCNN, can also be used for generating region proposals. On top of these convolutional
features, we construct an RPN by adding a few additional convolutional layers that simultaneously
regress region bounds and objectness scores at each location on a regular grid. The RPN
is thus a kind of fully convolutional network (FCN) [7] and **can be trained end-to end
specifically for the task for generating detection proposals**.

Region Proposal Networks

1. First the picture goes through conv layers and feature maps are extracted

2. Then a sliding window is used in RPN for each location over the feature map

3. For each location, 9 anchor boxes are used (3 scales of 128, 256 and 512,

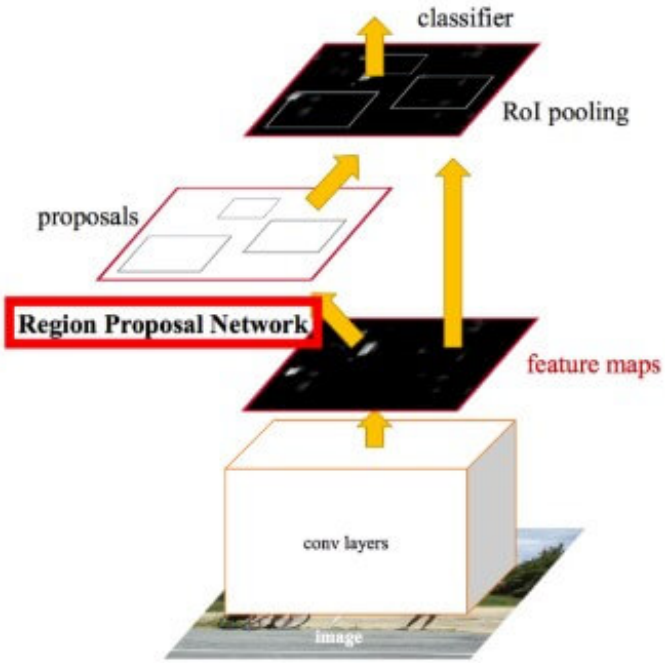
and 3 aspect ratios of 1:1, 1:2 and 2:1) for generating region proposals

4. A **cls** layer outputs **2x**9 scores whether is object or not for 9 anchors.

5. A **reg** layer outputs 4x9 scores for coordinates for 9 anchor boxes.

6. With a size of WxH feature map, there are WH9 anchors in total.

We then feed the region proposals to the RoI layer of the Fast R-CNN.



Let's look at the whole thing again

[(http://www.telesens.co/2018/03/11/object-detection-and-classification-using-r-cnns/)](http://www.telesens.co/2018/03/11/object-detection-and-classification-using-r-cnns/)

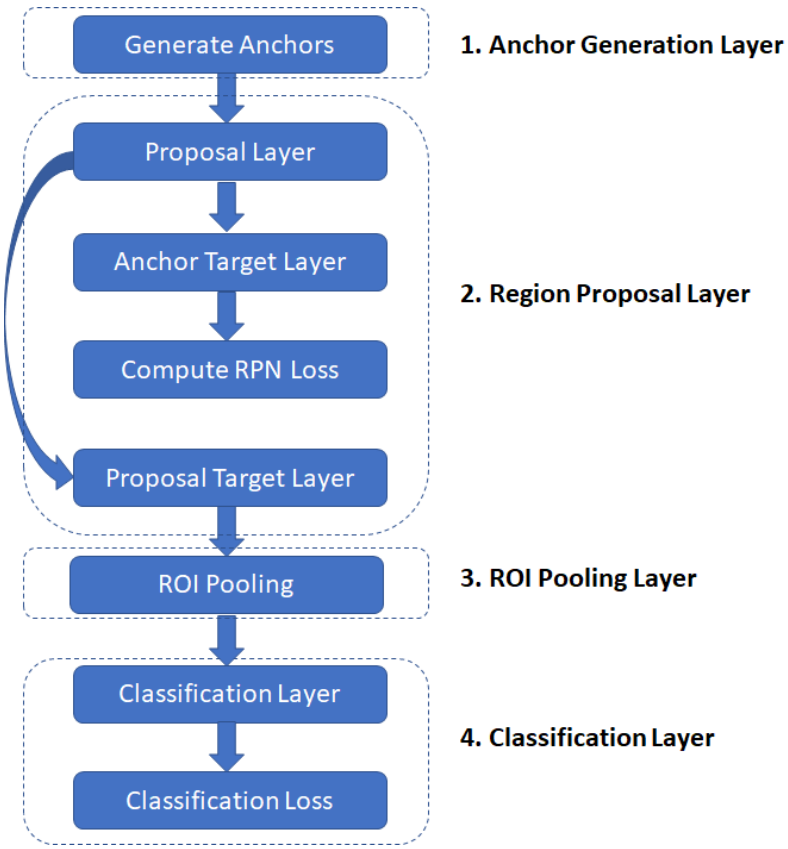
Faster RCNN (or others) have three main types of networks:

Head (say ResNet50 etc)

Region Proposal Network

Classification Network

In the software implementation, Faster R-CNN execution is broken down into several layers, as shown below. A layer encapsulates a sequence of logical steps that can
involve running data through one of the neural networks and other steps such as comparing
overlap between bounding boxes, performing non-maxima suppression etc.



- **Anchor Generation Layer:** This layer generates a fixed number of “anchors” (bounding boxes) by first generating 9 anchors of different scales and aspect ratios and then replicating these anchors by translating them across uniformly spaced grid points spanning the input image.
- **Proposal Layer:** Transform the anchors according to the bounding box regression coefficients to generate transformed anchors. Then prune the number of anchors

by applying non-maximum suppression (see Appendix) using the probability of an anchor being a foreground region

- **Anchor Target Layer**: The goal of the anchor target layer is to produce a set of “good” anchors and the corresponding foreground/background labels and target regression coefficients to train the Region Proposal Network. The output of this layer is only used to train the RPN network and is not used by the classification layer. Given a set of anchors (produced by the anchor generation layer, the anchor target layer identifies promising foreground and background anchors. Promising foreground anchors are those whose overlap with some ground truth box is higher than a threshold. Background boxes are those whose overlap with any ground truth box is lower than a threshold. The anchor target layer also outputs a set of bounding box regressors i.e., a measure of how far each anchor target is from the closest bounding box. These regressors only make sense for the foreground boxes as there is no notion of “closest bounding box” for a background box.
- **RPN Loss:** The RPN loss function is the metric that is minimized during optimization to train the RPN network. The loss function is a combination of:
  - The proportion of bounding boxes produced by RPN that are correctly classified as foreground/background
  - Some distance measure between the predicted and target regression coefficients.
- **Proposal Target Layer:** The goal of the proposal target layer is to prune the list of anchors produced by the proposal layer and produce *class specific* bounding box regression targets that can be used to train the classification layer to produce good class labels and regression targets

- **ROI Pooling Layer:** Implements a spatial transformation network that samples the input feature map given the bounding box coordinates of the region proposals produced by the proposal target layer. These coordinates will generally not lie on integer boundaries, thus interpolation based sampling is required.
- **Classification Layer:** The classification layer takes the output feature maps produced by the ROI Pooling Layer and passes them through a series of convolutional layers. The output is fed through two fully connected layers. The first layer produces the class probability distribution for each region proposal and the second layer produces a set of class specific bounding box regressors.
- **Classification Loss:** Similar to RPN loss, classification loss is the metric that is minimized during optimization to train the classification network. During back propagation, the error gradients flow to the RPN network as well, so training the classification layer modifies the weights of the RPN network as well. We'll have more to say about this point later. The classification loss is a combination of:
  - The proportion of bounding boxes produced by RPN that are correctly classified (as the correct object class)
  - Some distance measure between the predicted and target regression coefficients.

Anchor Generation Layer

The anchor generation layer produces a set of bounding boxes (called "anchor boxes") of

varying sizes and aspect ratios spread all over the input image. These bounding boxes are the same for all images i.e., they are agnostic of the content of an image. Some of these bounding boxes will enclose foreground objects while most won't. The goal of the RPN network is to learn to identify which of these boxes are good boxes – i.e., likely to contain a foreground object and to produce target regression coefficients, which when applied to an anchor box turns the anchor box into a better bounding box (fits the enclosed foreground object more closely).

Our objective is to find bounding boxes in the image. These have rectangular shape and can come in different sizes and aspect ratios. Imagine we were trying to solve the problem knowing beforehand that there are two objects on the image. The first idea that comes to mind is to train a network that returns 8 values: **two x_{min}, y_{min}, x_{max}, y_{max} tuples** defining a bounding box for each object.

This approach has some fundamental problems. For example, images may have

different sizes and aspect ratios, having a good model trained to predict raw coordinates can turn out to be very complicated (if not impossible). Another problem is invalid predictions: when predicting x_{min} and x_{max} we have to somehow enforce that $x_{min} < x_{max}$.

It turns out that there is a simpler approach to predicting bounding boxes by learning to predict offsets from reference boxes. We take a reference box x_{center}, y_{center}, width, height and learn to predict Δ_{x_center}, Δ_{y_center}, Δ_{width}, Δ_{height} , which are usually small values that tweak the reference box to better fit what we want.

The diagram below demonstrates how these anchor boxes are generated.

Region Proposal Layer

The region proposal layer has two goals:

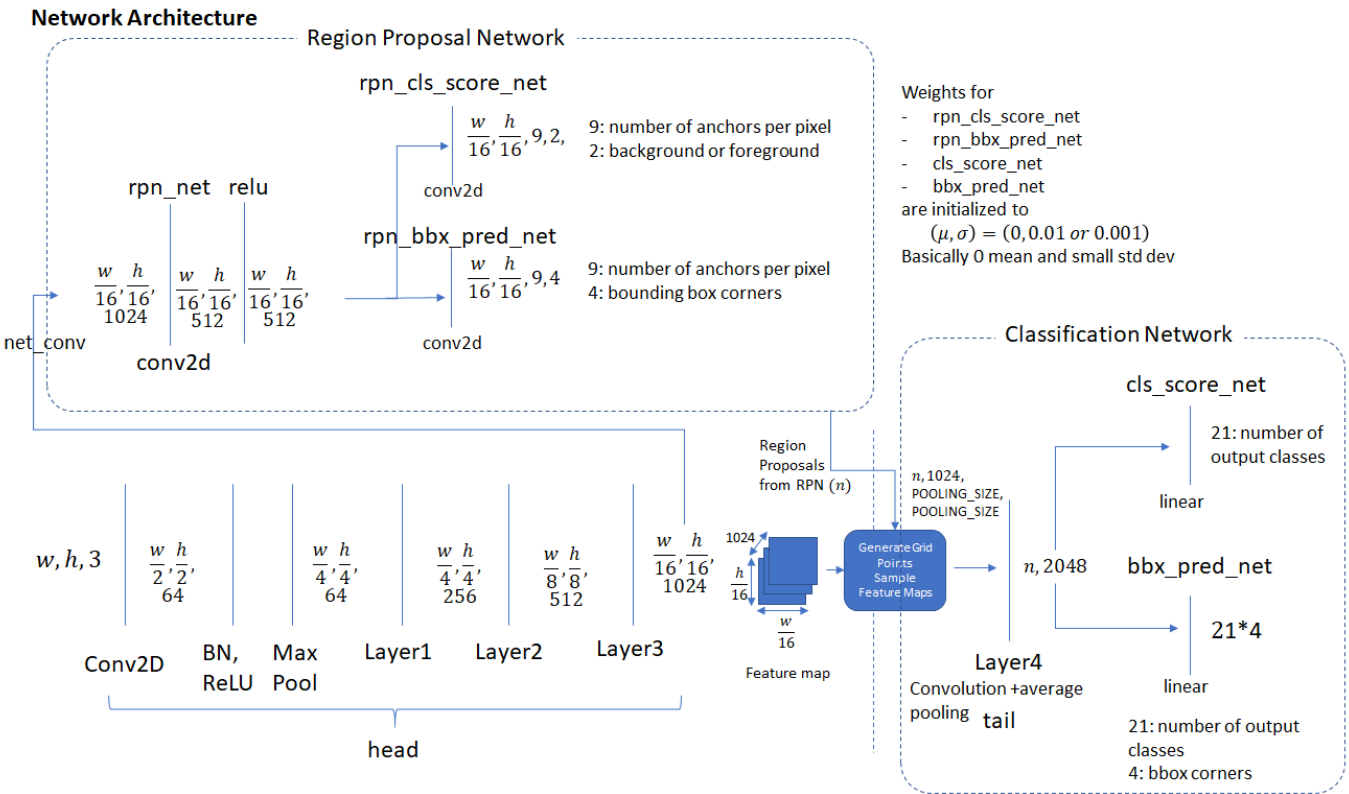
- From a list of anchors, identify background and foreground anchors
- Modify the position, width and height of the anchors by applying a set of “regression coefficients” to improve the quality of the anchors (for example, make them fit the boundaries of objects better)

The region proposal layer consists of a Region Proposal *Network* and three layers –

Proposal Layer
Anchor Target Layer and
Proposal Target Layer.

These three layers are described in detail in the following sections.

Region Proposal Network



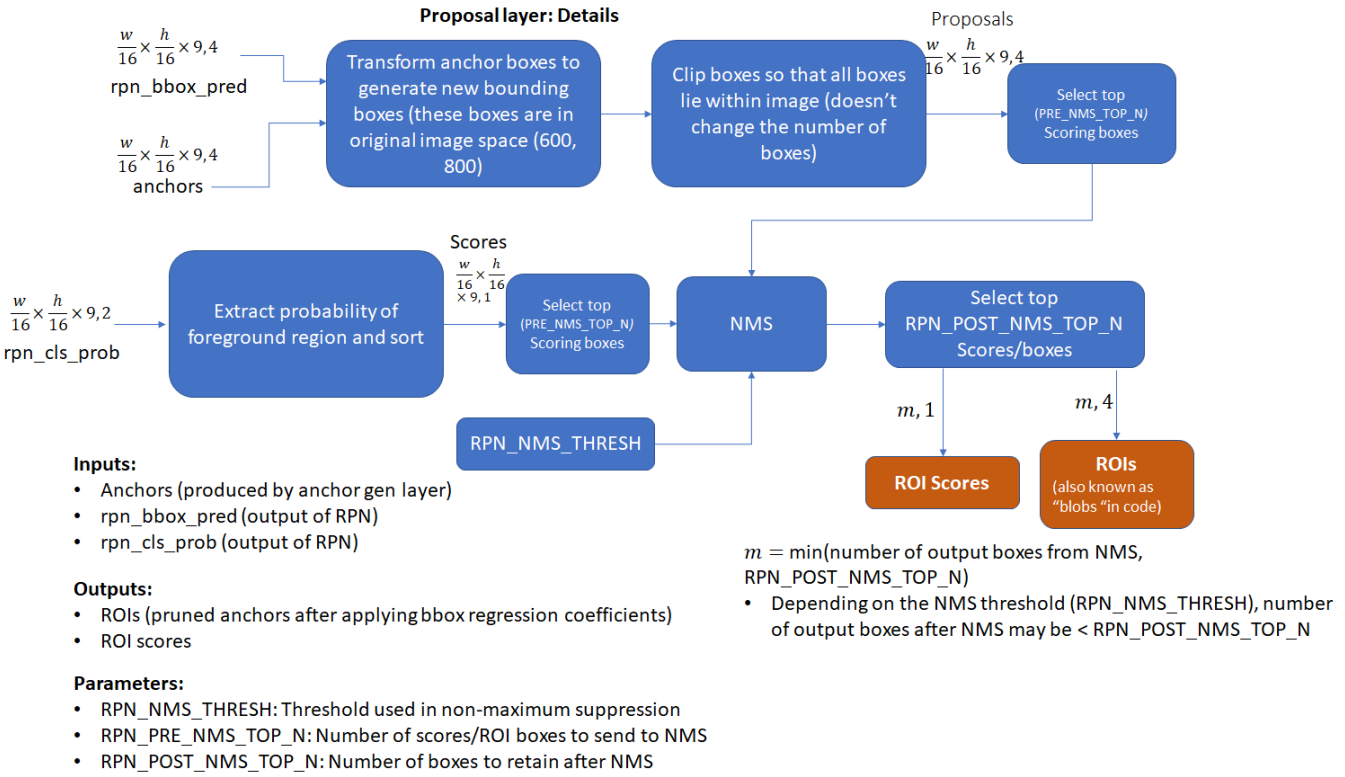
The region proposal layer runs feature maps produced by the head network through a

convolutional layer (called rpn_net in code) followed by RELU. The output of rpn_net is run
through two (1,1) kernel convolutional layers to produce background/foreground class
scores and probabilities and corresponding bounding box regression coefficients. The stride
length of the head network matches the stride used while generating the anchors, so the number
of anchor boxes are in 1-1 correspondence with the information produced by the region
proposal network

(number of anchor boxes = number of class scores = number of bounding box regression coefficients = $\frac{w}{16} \times \frac{h}{16} \times 9$)

Proposal Layer

The proposal layer takes the anchor boxes produced by the anchor generation layer and
prunes the number of boxes by applying non-maximum suppression based on the
foreground scores (see appendix for details). It also generates transformed bounding
boxes by applying the regression coefficients generated by the
RPN to the corresponding anchor boxes.



Anchor Target Layer

The goal of the anchor target layer is to select promising anchors that can be used to train the RPN network to:

distinguish between foreground and background regions and

generate good bounding box regression coefficients for the foreground boxes.

It is useful to first look at how the RPN Loss is calculated.
This will reveal the information needed to calculate the RPN loss which makes it easy to
follow the operation of the Anchor Target Layer.

Calculating RPN Loss

Remember the goal of the RPN layer is to generate good bounding boxes.
To do so from a set of anchor boxes, the RPN layer must learn to classify an anchor box as
background or foreground and calculate the regression coefficients to modify the position,
width and height of a foreground anchor box to make it a “better” foreground box (fit a foreground object more closely). RPN Loss is formulated in such a way
to encourage the network to learn this behaviour.

RPN loss is a sum of the classification loss and bounding box regression loss.
The classification loss uses cross entropy loss to penalize incorrectly classified boxes
and the regression loss uses a function of the distance between the true regression
coefficients (calculated using the closest matching ground truth box for a foreground anchor box)
and the regression coefficients predicted by the network (see rpn_bbx_pred_net in the
RPN network architecture diagram).

$$RPNLoss = \text{Classification Loss} + \text{Bounding Box Regression Loss}$$

Classification Loss

cross_entropy(predicted _class, actual_class)

Bounding Box Regression Loss

$$L_{loc} = \sum_{u \in \text{all foreground anchors}} l_u$$

Sum over the regression losses for all foreground anchors.
Doing this for background anchors doesn't make sense as there is no associated
ground truth box for a background anchor

$$l_u = \sum_{i \in x,y,w,h} smooth_{L1}(u_i(predicted) - u_i(target))$$

This shows how the regression loss for a given foreground anchor is calculated.
We take the difference between the predicted (by the RPN) and target (calculated using
the closest ground truth box to the anchor box) regression coefficients.
There are four components –
corresponding to the coordinates of the top left corner and
the width/height of the bounding box.

The smooth L1 function is defined as follows:

$$smooth_{L1}(x) = \begin{cases} \frac{\sigma^2 x^2}{2} & \|x\| < \frac{1}{\sigma^2} \\ \|x\| - \frac{0.5}{\sigma^2} & otherwise \end{cases}$$

Only anchor boxes whose overlap with some ground truth box exceeds a threshold
are selected as foreground boxes. This is done to avoid presenting the RPN with the “hopeless learning task” of learning the regression coefficients of boxes that are too far from the best match ground truth box. Similarly, boxes whose overlap
are less than a negative threshold are labeled background boxes. Not all boxes that
are not foreground boxes are labeled background. Boxes that are neither foreground
or background are labeled “don’t care”.
These boxes are not included in the calculation of RPN loss.

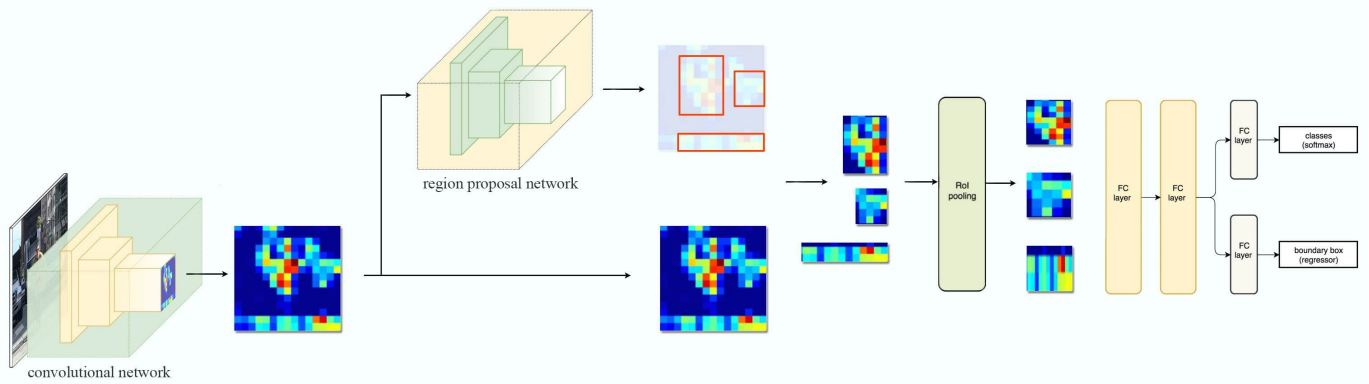
Classification Layer

The feature vector is then passed through two fully connected layers – bbox_pred_net
and cls_score_net. The cls_score_net layer produces the class scores for each bounding
box (which can be converted into probabilities by applying softmax). The bbox_pred_net
layer produces the class specific bounding box regression coefficients which are

combined with the original bounding box coordinates produced by the proposal target layer to produce the final bounding boxes. These steps are shown below.

RECAP

Faster-RCNN



Faster R-CNN uses a CNN feature extractor to extract image features. Then it uses
a CNN region proposal network to create region of interests (RoIs). We apply RoI
pooling to warp them into fixed dimension. It is then feed into fully connected layers
to make classification and boundary box prediction.

Mask R-CNN

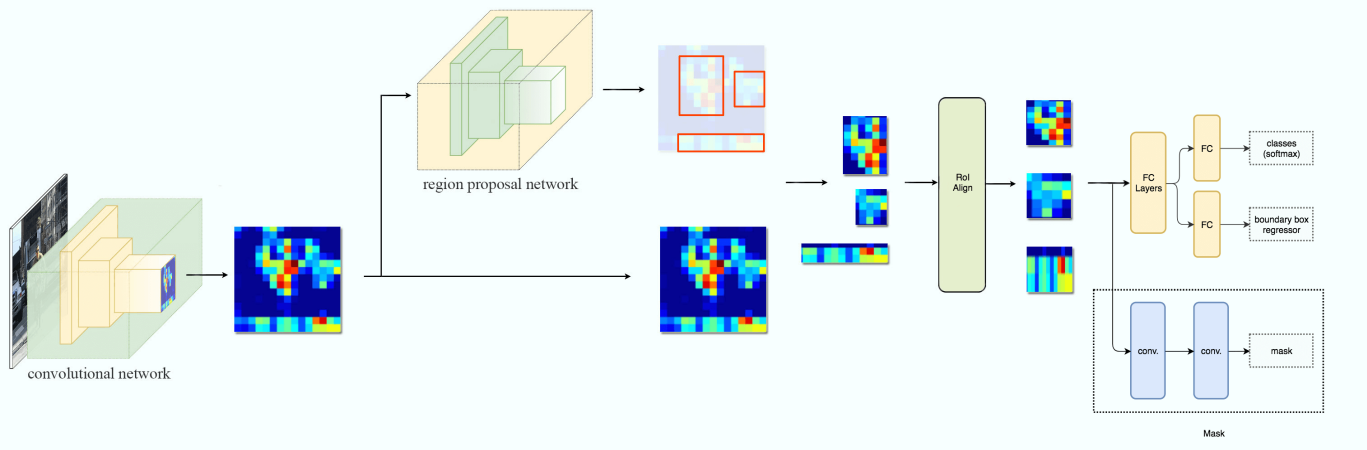
The Faster R-CNN builds all the ground works for feature extractions and ROI proposals.

At first sight, performing image segmentation may require more detail analysis to colorize

the image segments. By surprise, not only we can piggyback on this model, the extra work

required is pretty simple. After the ROI pooling, we add 2 more convolution

layers to build the mask.



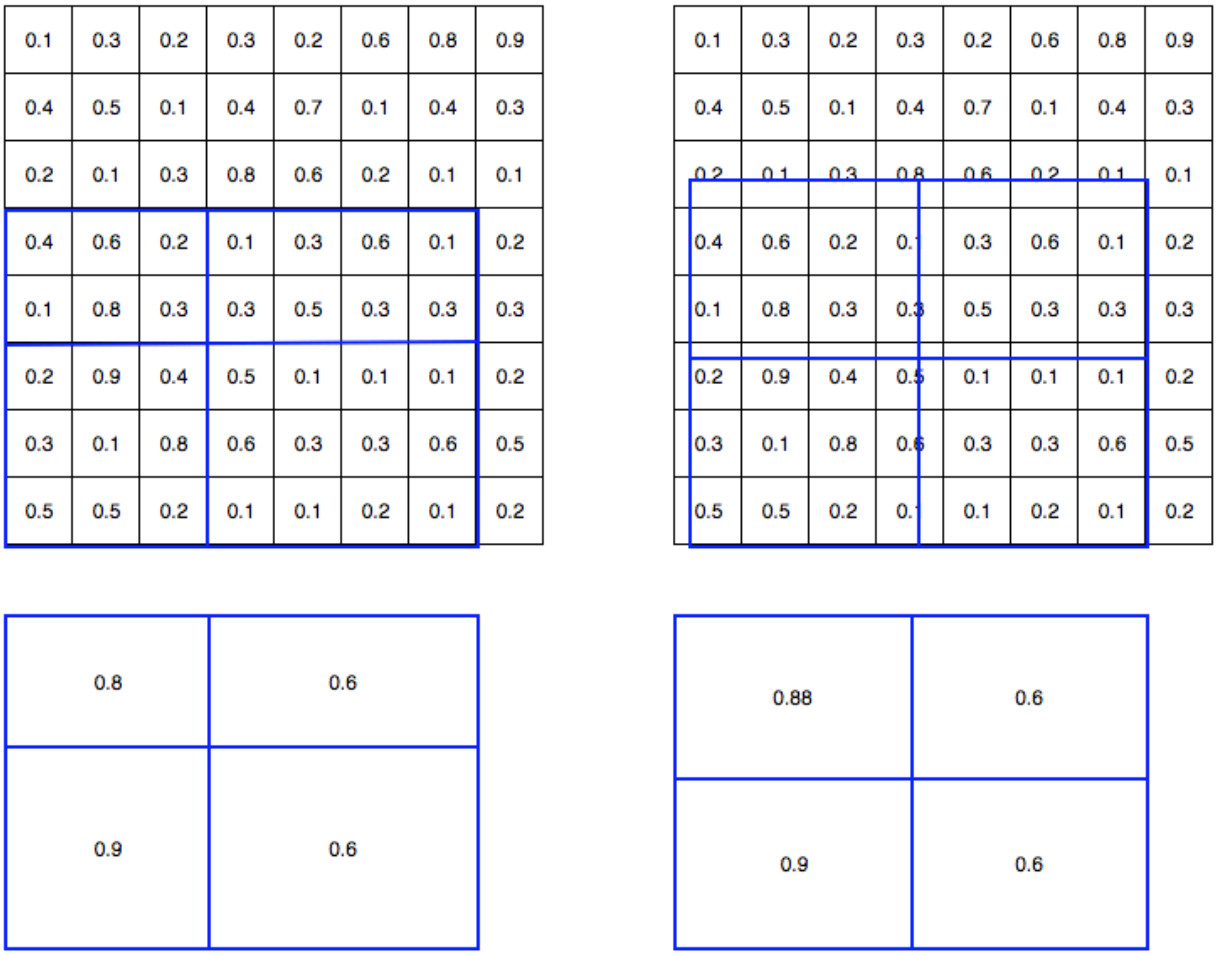
ROI Align

Another major contribution of Mask R-CNN is the refinement of the ROI pooling. In ROI, the warping is digitalized (top left diagram below): the cell boundaries of the target feature map are forced to realign with the boundary of the input feature maps.

Therefore, each target cells may not be in the same size (bottom

left diagram). Mask R-CNN uses **ROI Align** which does not digitalize the boundary of the cells (top right) and make every target cell to have the same size (bottom right).

It also applies interpolation to calculate the feature map values within the cell better. For example, by applying interpolation, the maximum feature value on the top left is changed from 0.8 to 0.88 now.



ROI Align makes significant improvements in the accuracy.

	AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
RoIPool	23.6	46.5	21.6	28.2	52.7	26.9
RoIAlign	**30.9**	**51.8**	**32.1**	**34.0**	**55.3**	**36.4**
	+7.3	*+ 5.3*	*+10.5*	*+5.8*	*+2.6*	*+9.5*

This assignment is optional.

1. Run [this (https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5?pli=1&authuser=2#scrollTo=0e4vdDIOXyxF)](https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5?pli=1&authuser=2#scrollTo=0e4vdDIOXyxF) on your own video.
2. Upload the video on YouTube
3. Share the link.

Assignment 15A

In this assignment, you'll learn how to build a custom dataset. We would be building a
dataset for monocular depth estimation and segmentation simultaneously.

The ability to make and link your own dataset to a model is key to any ML Engineer.

Since we don't have resources like companies or universities we will use some ingenuity to create this dataset.

1. Select "scene" images. Like the front of shops, etc. We call this background.

2. Find or make 100 images of objects with transparent background. Use GIMP. We call this foreground.

3. Create 100 Masks for the above image. Use GIMP

4. Overlay the foreground on top or background randomly. Flip foreground as well. We call this fg_bg

5. Don't forget to create equivalent masks for these images:

6. Use this or similar [Depth Models (https://github.com/ialhashim/DenseDepth/blob/master/DenseDepth.ipynb)](https://github.com/ialhashim/DenseDepth/blob/master/DenseDepth.ipynb) to create depth maps for the fg_bg images:

You must have 100 background, 100x2 (including flip), and you randomly place the foreground on the background 20 times, you have in total 100x200x20 images.

In total you MUST have:

1. 400k fg_bg images
2. 400k depth images
3. 400k mask images
4. generated from:
    1. 100 backgrounds
    2. 100 foregrounds, plus their flips
    3. 20 random placement on each background.
5. Now add a readme file on GitHub for Project 15A:
    1. Create this dataset and share a link to GDrive (publicly available to anyone) in this readme file.
    2. Add your dataset statistics:
        1. Kinds of images (fg, bg, fg_bg, masks, depth)
        2. Total images of each kind
        3. The total size of the dataset
        4. Mean/STD values for your fg_bg, masks and depth images
    3. Show your dataset the way I have shown above in this readme
    4. Explain how you created your dataset
        1. how were fg created with transparency
        2. how were masks created for fgs
        3. how did you overlay the fg over bg and created 20 variants
        4. how did you create your depth images?
6. Add the notebook file to your repo, one which you used to create this dataset
7. Add the notebook file to your repo, one which you used to calculate statistics for this dataset

Things to remember while creating this dataset:

1. stick to square images to make your life easy.
2. We would use these images in a network which would take an fg_bg image AND bg image, and predict your MASK and Depth image. So the input to the network is, say, 224x224xM and 224x224xN, and the output is 224x224xO and 224x224xP.
3. pick the resolution of your choice between 150 and 250 for ALL the images

15A is a group assignment.

Questions asked in 15A:

1. Share the link to the readme file for your Assignment 15A. Read the assignment again to make sure you do not miss any part which you need to explain. -2500
2. Share the link to your notebook which you used to create this dataset. We are expecting to see how you manipulated images, overlay code, depth predictions. -250
3. Surprise question. -Surprise Marks.

VIDEO