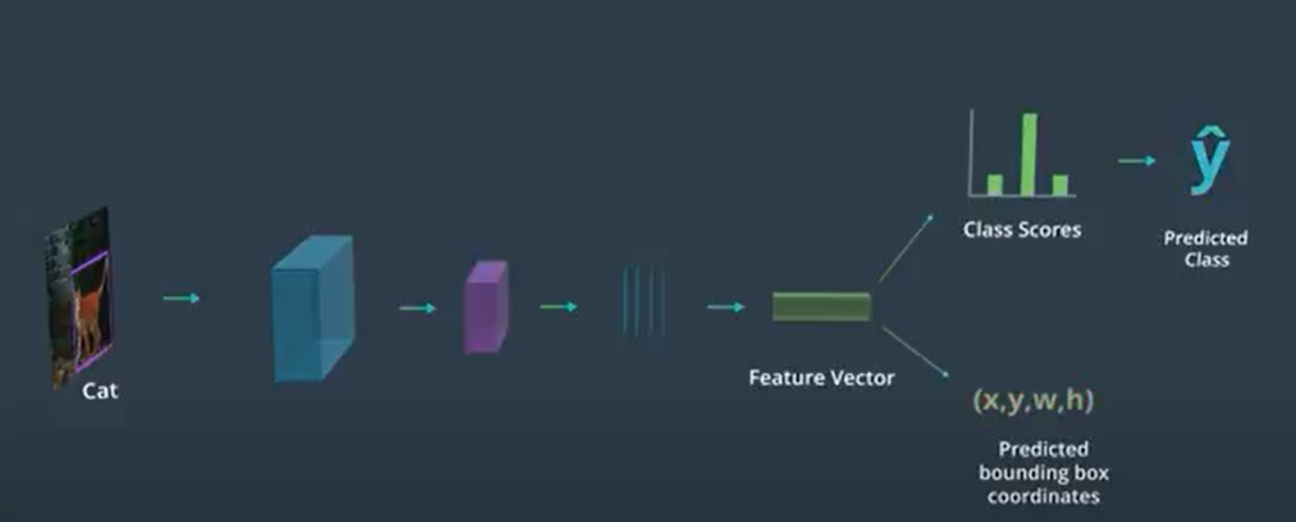
Localization of an image

Region based CNN – Fast R-CNN models- it analyses different cropped areas in a single input image, decides which regions corresponds to objects, and then performs classifications as usual.

Application- if baby is safely in the crib,



This predicts the location and size of the bounding box. Or the bounding box coordinates. So, in this single CNN, one output path’s job is to produce a class for the object pictured in an image, and another who’s job is to produce the bounding box coordinates for the object. In this case we assume, that the input image not only has a true label, but also has a true bounding box. This way we can train our network by comparing the predicted and true values for both the classes and bounding boxes.

Measure performance using a cross entropy loss.

For bounding box, we need a function that measures the error between our predicted bounding box and our true bounding box. So it’s a regression kind of problem. Loss functions appropriate for this are(that compare quantities, instead of class scores): Mean squared error.

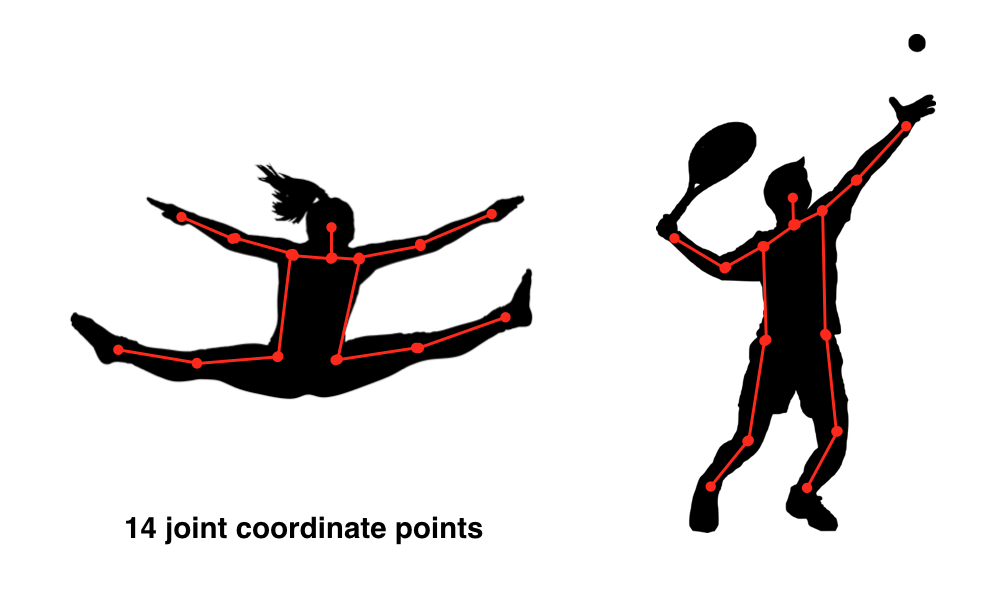
So when we look at comparing a set of points, say locations or points on a face, or points that define a specific region in an image, we need a loss function that measures the similarity between these coordinate values. So it’s a regression kind of problem. Mean squared error.

The simplest measure is L1 loss which measures the element-wise difference between a predicted output P and a target T. L1 loss returns a value that represents the distance between the predicted and true points. Disadvantage is that L1 loss may become negligible for small error values and MSE loss responds the most to large errors, so it might end up amplifying errors that are big but infrequent/ outliers.

Smooth L1 loss – for small differences between the predicted and true values uses a squared error function and for larger errors, uses L1 loss. So it tries to combine the best aspects of MSE and L1.

Application – **Human Pose Estimation**

This kind of model can be extended to any problem that has coordinate values as outputs! One such example is human pose estimation.



Huan pose estimation points.

In the above example, we see that the pose of a human body can be estimated by tracking 14 points along the joints of a body.

Weighted Loss Functions

You may be wondering: how can we train a network with two different outputs (a class and a bounding box) and different losses for those outputs?

We know that, in this case, we use categorical cross entropy to calculate the loss for our predicted and true classes, and we use a regression loss (something like smooth L1 loss) to compare predicted and true bounding boxes. But, we have to train our whole network using one loss, so how can we combine these?

There are a couple of ways to train on multiple loss functions, and in practice, we often use a weighted sum of classification and regression losses **(ex. 0.5\*cross\_entropy\_loss + 0.5\*L1\_loss)**; the result is a single error value with which we can do backpropagation. This does introduce a hyperparameter: the loss weights. We want to weight each loss so that these losses are balanced and combined effectively, and in research we see that another regularization term is often introduced to help decide on the weight values that best combine these losses.

Region Proposals

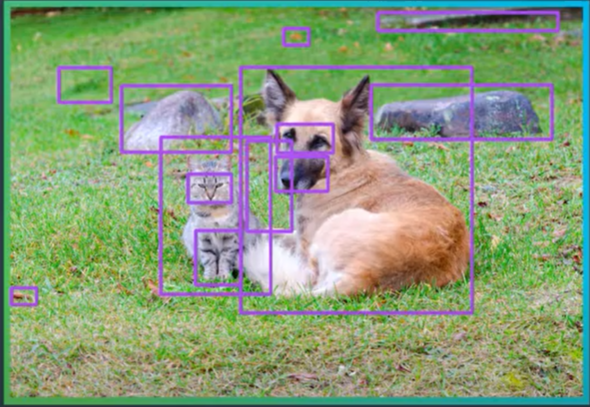
We don’t know ahead of time how many objects are going to be in an image. So there’s a variable output. CNNs and most neural networks have a defined unchanging output size. So, to detect a variable amount of objects in any image, you first must break that image up into smaller regions and produce bounding boxes and class labels for one region and one object at a time. Then we’ll be able to locate and classify any object that appears in an original image, whether its one object or three or twenty.

How to break up an image into regions?

Regions to correspond to different objects, so that we don’t miss any object. For this we could just make a bunch of cropped regions to make sure we don’t miss anything. This would mean defining a small sliding window and passing it over the entire image using some value for stride to create mini different crops of the original input image. For each of the cropped image, we can put it through a CNN and perform classification. But this approach produces a huge amount of cropped images, some of which don’t even contain objects and is highly time intensive.

**So, to better choose these cropped regions, especially when objects vary in size and location.**

The regions we want to analyze are those with complete objects in them. We want to get rid of regions that contain image background or only a portion of an object. So, two common approaches are suggested: **1.** identify similar regions using feature extraction or a clustering algorithm **like k-means**, as you've already seen; these methods should identify any areas of interest. **2.** Add another layer to our model that performs a binary classification on these regions and labels them: object or not-object; this gives us the ability to discard any non-object regions!



R-CNN (Region Convolutional Neural Network)

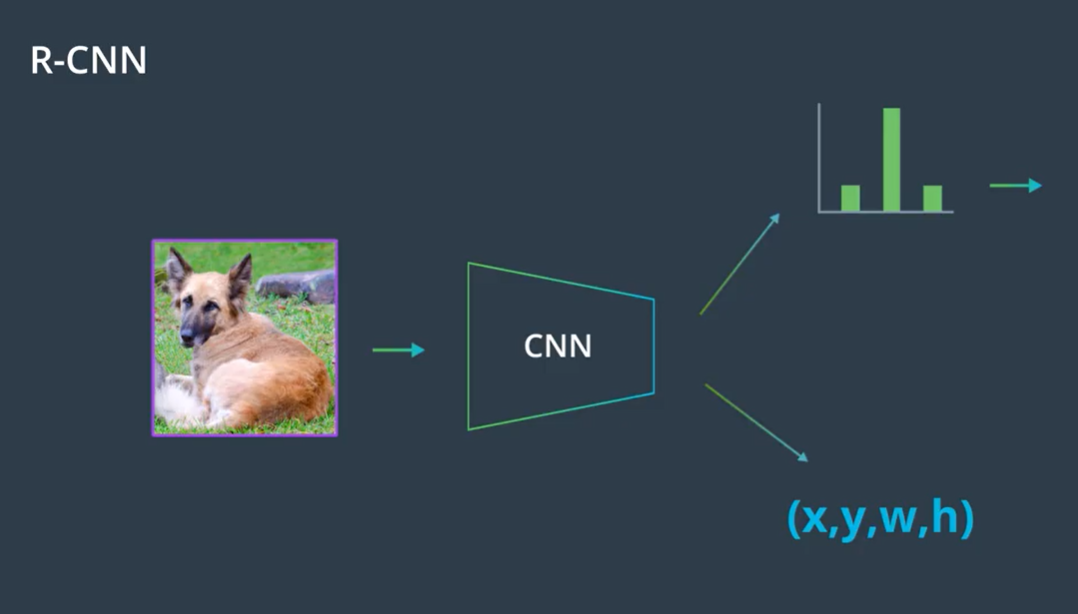
To localize and classify multiple objects in an image, we want to be able to identify a limited set of cropped regions for a CNN to look at. In ideal case, we would generate three perfectly cropped regions for three different objects in an image. To approach this goal and generate a good limited set of cropped regions, **the idea of region proposals was introduced.**

We can use traditional computer vision techniques that detect things like edges and textured bobs to produce a set of regions in which objects are most likely to be found, areas of similar texture or the same unifying boundary for example. These proposals often produce noisy non-object regions, but they are also very likely to include the regions in which objects are located. So the noise is considered a worthwhile cost for not missing any objects.

We can use Region Proposal Algorithms to produce a limited set of cropped regions, often called regions of interests or ROIs. We put these regions through a classification CNN one at a time and see what kind of class label the network predicts for each crop. R-CNN produces a class for each region of interest **(ROI)** – eg. A cat or a dog. In this case we also include a class called **‘background’** – meant to capture any noisy regions.

Since these regions are often different sizes, they need to be transformed and warped into a standard size that a CNN can accept as input.

Shortcoming – Time intensive still- since each cropped image has to go through an entire CNN before a class label can be produced.

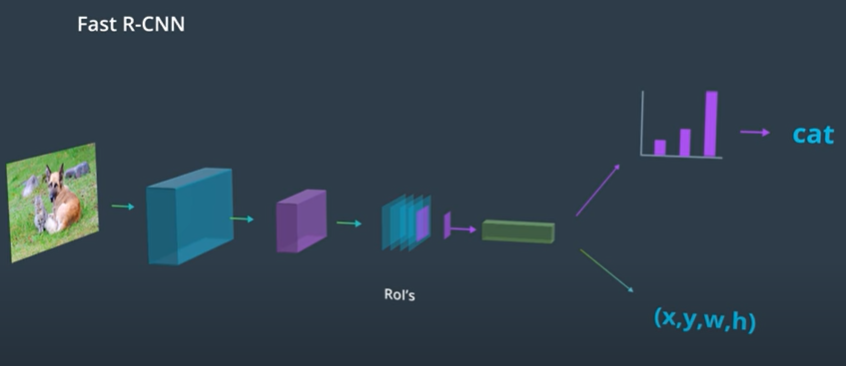


## R-CNN Outputs

The R-CNN is the least sophisticated region-based architecture, but it is the basis for understanding how multiple object recognition algorithms work! It outputs a class score and bounding box coordinates for every input RoI.

An R-CNN feeds an image into a CNN with regions of interest (RoI’s) already identified. Since these RoI’s are of varying sizes, they often need to be **warped to be a standard size**, since CNN’s typically expect a consistent, square image size as input. After RoI's are warped, the R-CNN architecture, processes these regions one by one and, for each image, produces 1. a class label and 2. a bounding box (that may act as a slight correction to the input region).

1. R-CNN produces bounding box coordinates to reduce localization errors; so a region comes in, but it may not perfectly surround a given object and the output coordinates (x,y,w,h) aim to perfectly localize an object in a given region.
2. R-CNN, unlike other models, does not explicitly produce a confidence score that indicates whether an object is in a region, instead it cleverly produces a set of class scores for which one class is "background". This ends up serving a similar purpose, for example, if the class score for a region is Pbackground = 0.10, it likely contains an object, but if it's Pbackground = 0.90, then the region probably doesn't contain an object.



**Fast R- CNN**

It speeds up the above process and efficiently classifies multiple objects in an image.

Instead of processing each region of interest individually through a classification CNN, this architecture runs the entire image through the classification CNN only once. The image goes through a series of convolutional and pooling layers and at the end of these layers, we get a stack of feature maps. We still need to identify regions of interests but instead of cropping the original image, we project these proposals into the smaller feature map layer. Each region in the feature map corresponds to a larger region in the original image.

So we can grab selected regions in this feature map and feed them one by one into a fully connected layer, that generates a class for each of these different regions.

In this model we complete the most time consuming steps, processing an image through a series of convolutional layers only once and then selectively use that map to get out desired outputs.

Again, we have to handle variable sizes in these projections since layers further in the network are expecting input of a fixed size.

So we do something called ROI pooling to warp these regions into a consistent size before giving them to a fully connected layer.

**Faster than R-CNN but kind of slow when faced with a test image for which it has to generate region proposals and it’s still looking at regions that do not contain objects at all.**

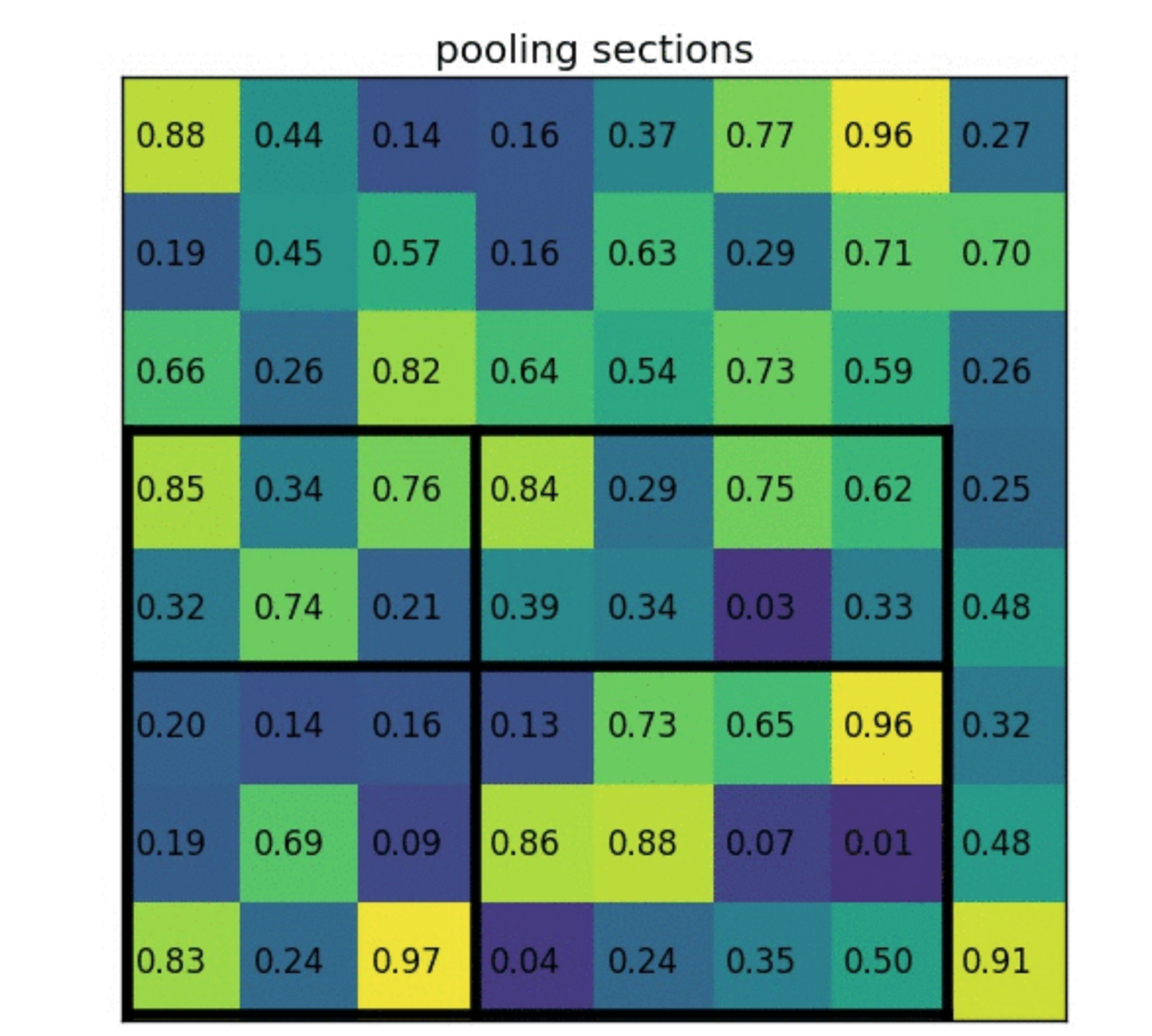
### RoI Pooling

To warp regions of interest into a consistent size for further analysis, some networks use RoI pooling. RoI pooling is an additional layer in our network that takes in a rectangular region of any size, performs a maxpooling operation on that region in pieces such that the output is a fixed shape. Below is an example of a region with some pixel values being broken up into pieces which pooling will be applied to; a section with the values:

[[0.85, 0.34, 0.76],

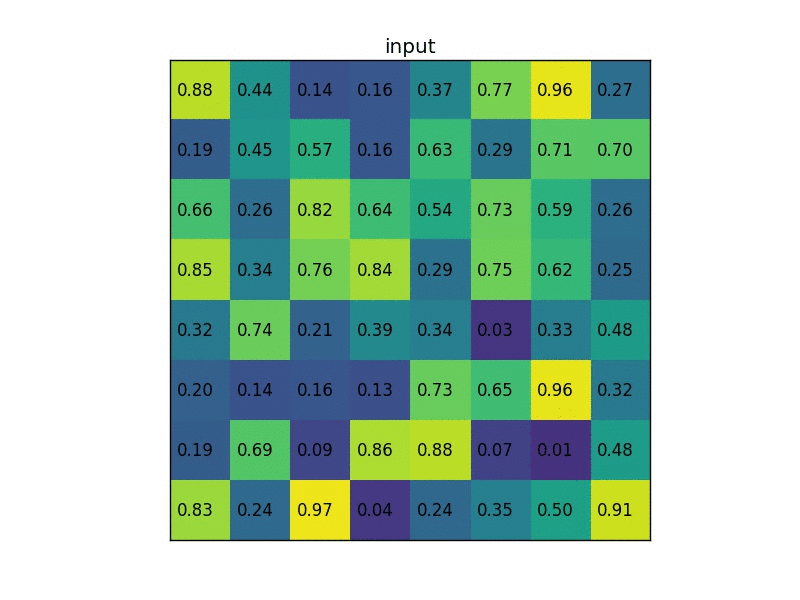
[0.32, 0.74, 0.21]]

Will become a single max value after pooling: 0.85. After applying this to an image in these pieces, you can see how any rectangular region can be forced into a smaller, square representation.



An example of pooling sections, credit to [this informational resource](https://blog.deepsense.ai/region-of-interest-pooling-explained/) on RoI pooling [by Tomasz Grel].

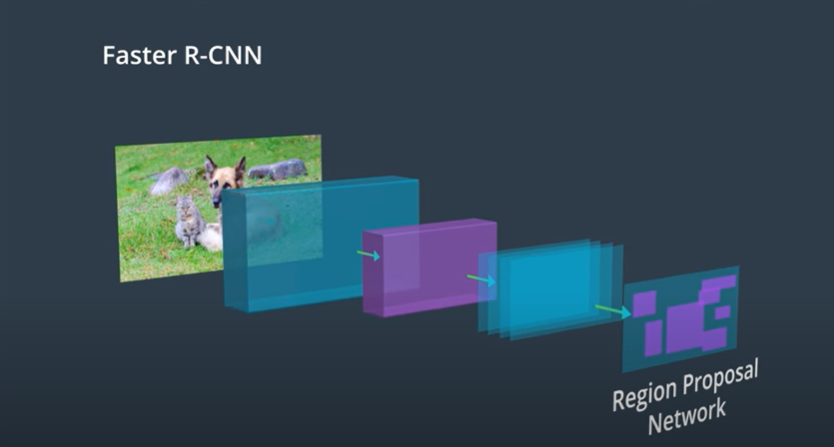
You can see the complete process from input image to region to reduced, maxpooled region, below.



Credit to [this informational resource](https://blog.deepsense.ai/region-of-interest-pooling-explained/) on RoI pooling.

### Speed

Fast R-CNN is about 10 times as fast to train as an R-CNN because it only creates convolutional layers once for a given image and then performs further analysis on the layer. Fast R-CNN also takes a shorter time to test on a new image! It’s test time is dominated by the time it takes to create region proposals.

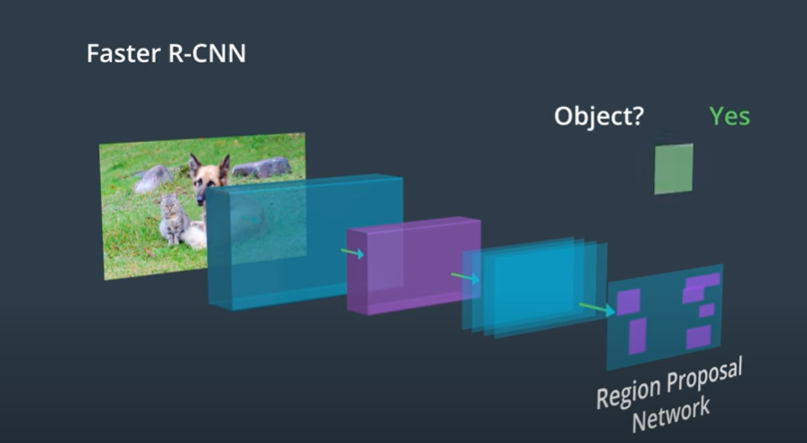


**Faster R-CNN**

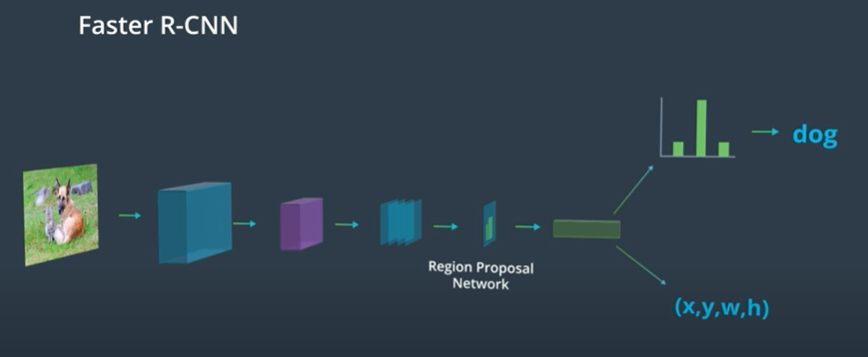
Improves the above region generation step

To speed up the time it takes to run a test image through a network and detect all objects in it, we want to decrease the time it takes to form region proposals. This is achieved by the Faster-RCNN architecture. Faster – R-CNN learns to come up with its own region proposals.

The produced feature map is used as an input into a separate region proposal network. So it predicts its own regions from the features produced inside the network. If an area in the feature map is rich in detected edges or other features, it is identified as a region of interest. Then this part of the network does a quick binary classification. For each ROI it checks whether or not that region contains an object. If it does then the region will continue and go through the classification steps, and if it doesn’t, then the proposal is discarded.



Once we have the final region proposals, the rest of the network looks the same as Fast R-CNN. It takes the cropped regions from the feature map and learns to classify those regions. By eliminating the analysis of the non object regions, this model is the fastest of all the region based CNN that we have seen.



## Region Proposal Network

You may be wondering: how exactly are the RoI's generated in the region proposal portion of the Faster R-CNN architecture?

The region proposal network (RPN) works in Faster R-CNN in a way that is similar to YOLO object detection, which you'll learn about in the next lesson. The RPN looks at the output of the last convolutional layer, a produced feature map, and takes a sliding window approach to possible-object detection. It slides a small (typically 3x3) window over the feature map, then for each window the RPN:

1. Uses a set of defined anchor boxes, which are boxes of a defined aspect ratio (wide and short or tall and thin, for example) to generate multiple possible RoI's, each of these is considered a region proposal.
2. For each proposal, this network produces a probability, Pc, that classifies the region as an object (or not) and a set of bounding box coordinates for that object.
3. Regions with too low a probability of being an object, say Pc < 0.5, are discarded.

#### Training the Region Proposal Network

Since, in this case, there are no ground truth regions, how do you train the region proposal network?

The idea is, for any region, you can check to see if it overlaps with any of the ground truth objects. That is, for a region, if we classify that region as an object or not-object, which class will it fall into? For a region proposal that does cover some portion of an object, we should say that there is a high probability that this region has an object init and that region should be kept; if the likelihood of an object being in a region is too low, that region should be discarded.

I'd recommend [this blog post](https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9) if you'd like to learn more about region selection.

### Speed Bottleneck

Now, for all of these networks including Faster R-CNN, we've aimed to improve the speed of our object detection models by reducing the time it takes to generate and decide on region proposals. You might be wondering: is there a way to get rid of this proposal step entirely? And in the next section we'll see a method that does not rely on region proposals to work!

Fast object detection is typically only required in applications that analyze object motion in real time.

<https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9>

<https://github.com/jwyang/faster-rcnn.pytorch>

Object detection/ recognition without region proposals

YOLO – you only look once

SSD – single shot detection