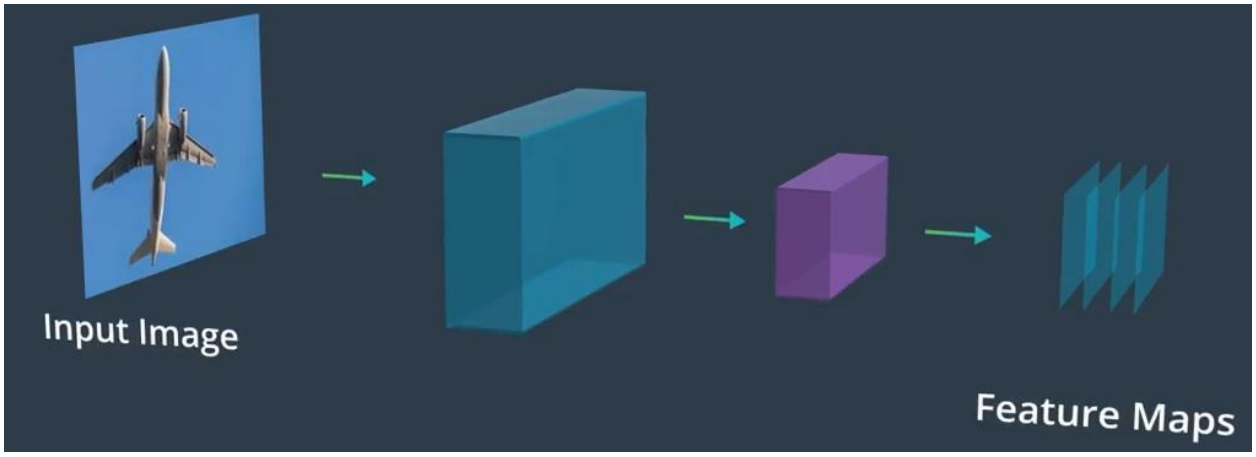
ADVANCED_CNN ARCHTIECTURE

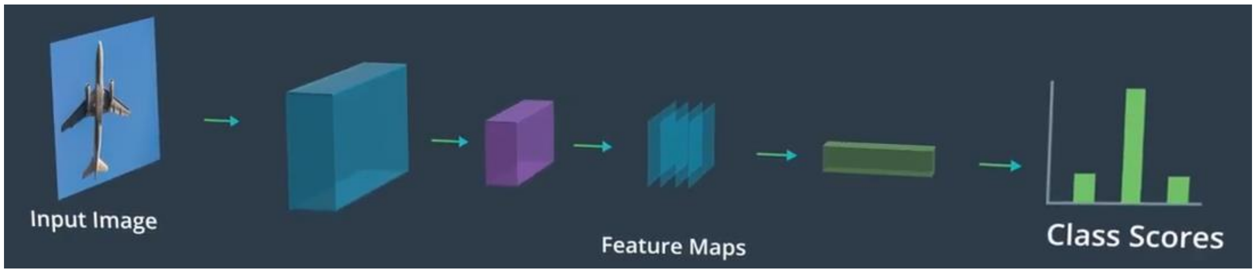
So far , you ‘ve seen a variety of image processing techniques that play a foundational role in pattern recognition tasks, such as image classifications.

You ‘ve seen how convolutional neural network follow a series of steps to classify an image. Just recap a CNN first takes in an input image then puts that image through several convolutional and pooling layers.

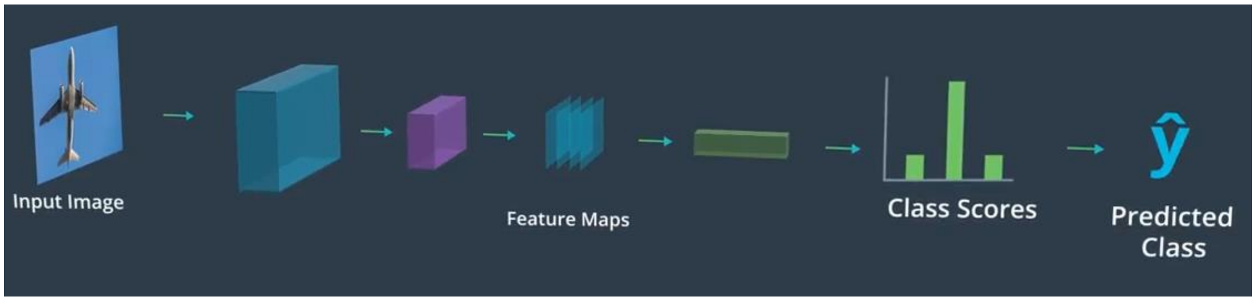
The result is a set of **features maps** reduced in size from the original image that through a training process have learned to distill information about the correct in the original image.



We then flatten this features maps creating a vector that we can then pass to a series of fully connected liner layer to produce a probability distribution of class scores.



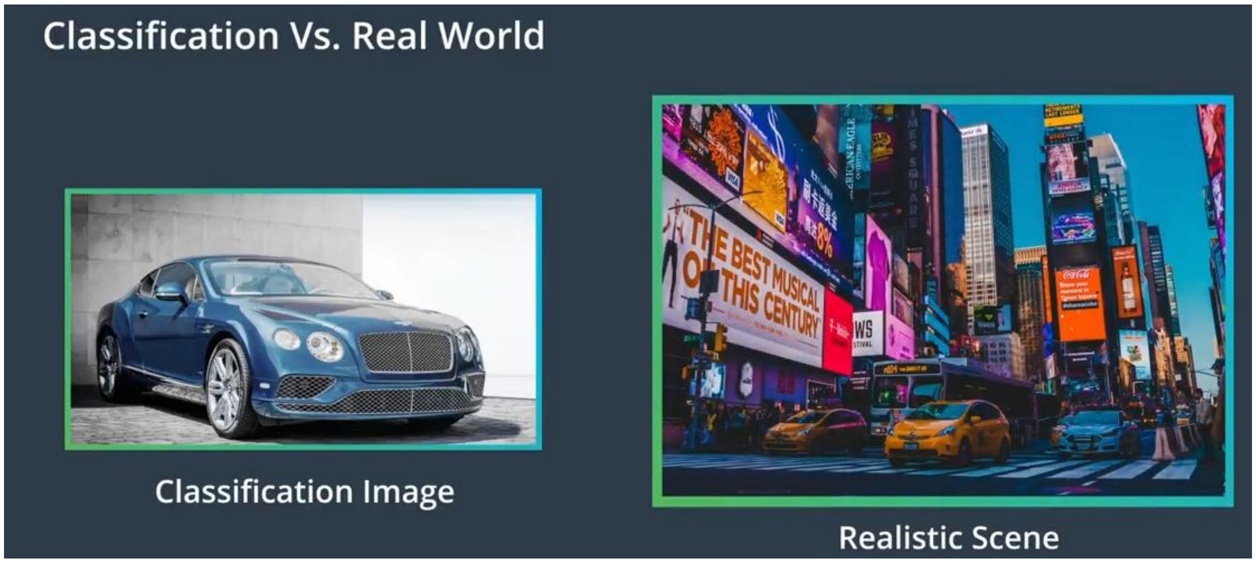
From this, we can extract the predicted class for the input image.



So in short , an image comes in and the predicted class label comes out .

In Classifications tasks like these, there`s usually a single object per image that a network is expected to classify.

But in the real world , we ‘ve often faced wit much more complex visual scenes, scenes with many overlapping objects.



We can and classify many objects at a time, and even estimate things like the distance between objects in a scene

We ‘ll at different kinds of CNN architecture and see how they ‘ve evolved over time Specifically , we ‘ll look at models that detect multiple objects in a scene, Like

1- **Faster R-CNN**
2- **YOLO**

Two kind of networks that can look at in image, break it up into **smaller regions**, and **label each region with class** so that a variable number of objects in a given image can be localized and labeled.

Later on, in the course, you will also learn about recurrent neural network that allow us to **process and generate a sequences** of data such as

1- **image frames**
2- **sequence of words**

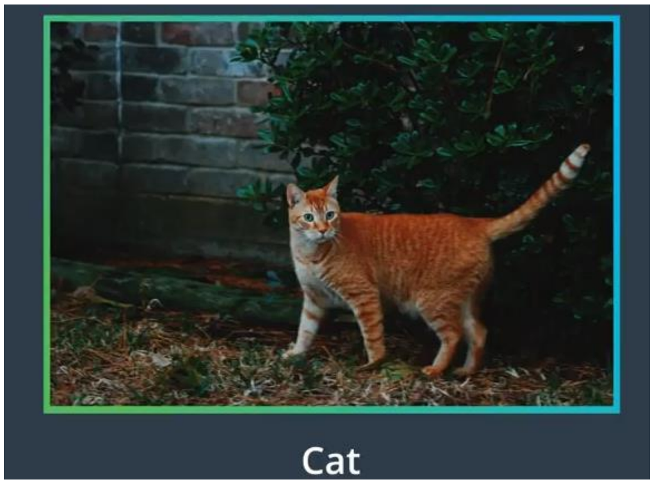
which can be useful should you want to describe visual scenes as in the case of **automatic image captioning**

So, let`s start by looking at some complex tasks that CNNs can be applied to

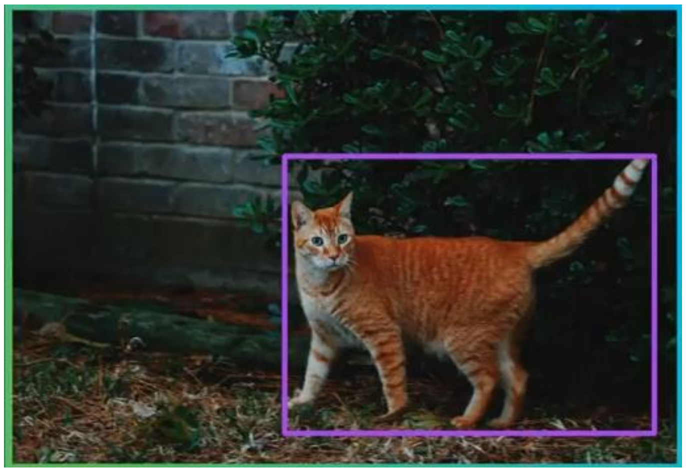
More than classification RENDER

Different type in CNN architectures that can do more than single image classification.

1- **First we 'ill take about classifying one object in an image and localizing it which means fining its locations in the image**



This is typically done by placing a **bounding box** around the object.

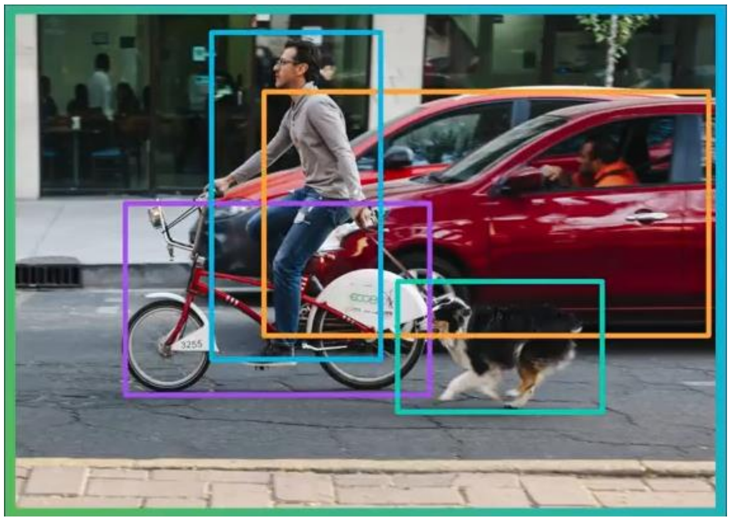


Finding the location of an **object in an image** and finding the location of an object in an image and more generally being able to analyze an image by breaking it up into **smaller bounded regions is the key** creating model that can classify multiple objects in an image.

We ‘ll build up to learning about

1- Region-based CNN’s like (FAST R-CNN image)

Like the **faster R-CNN model** which analyzes different cropped areas of a single input image , decides which regions correspond to objects and that performs classifications as usual



This is kind of architecture that`s used in cutting edge applications, from medical diagnostic to autonomous vehicles.

So in classifications task , we given image to a CNN and it outputs a label for the entire image. But sometimes you want a little more information.

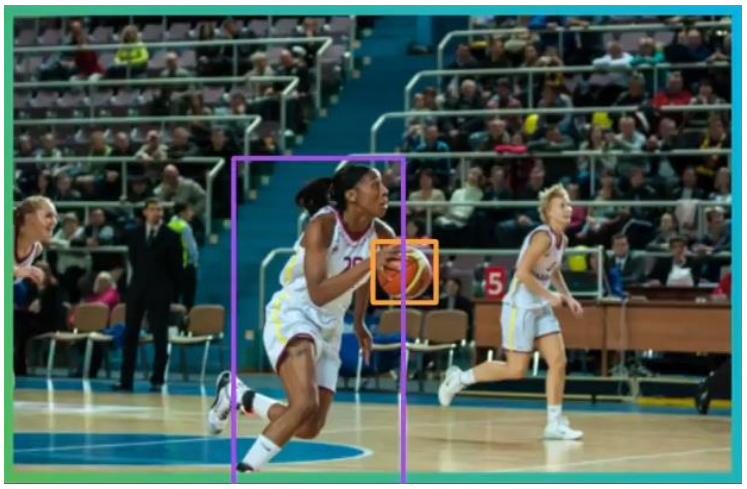
Say where the object actually is located in the image and this is called **localizations.**

For example

Say you have an image of basketball players and you want

1- to identify the player that has possessions of the basketball at the time.

The complete this task, you need to locate and identify the ball and the person that`s holding it



Localizations has used been in a verity of safely applications too.

For example and baby monitors that checks to see if a baby is safely in a crib.

And localizations is even in safe driving applications in which a camera checks to see If a driver is distracted or not by looking

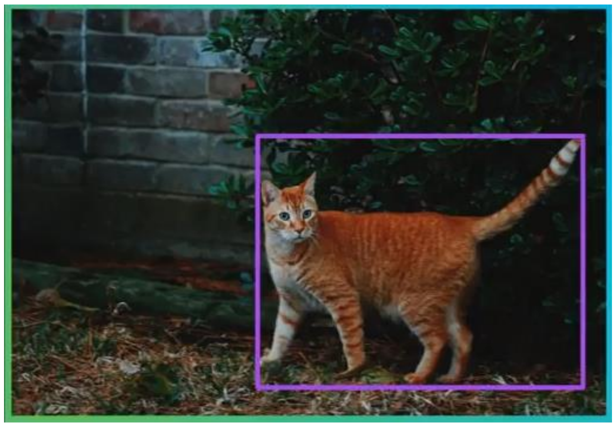
1- **the locations of a driver in a vehicle and**
2- **the locations of their eyes cell phone and other times around them**

Localizations is important for any applications that relies on looking at the proximately of two or more objects.

Let`s look at a simple Localizations example.

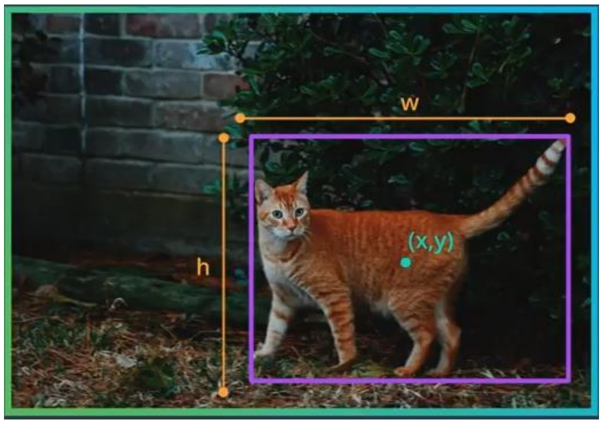
This image of a cat , in addition to labeling this image as a cat we also want to locate the cat in the image.

The typically way of doing this is by drawing a bounding box around the cat.



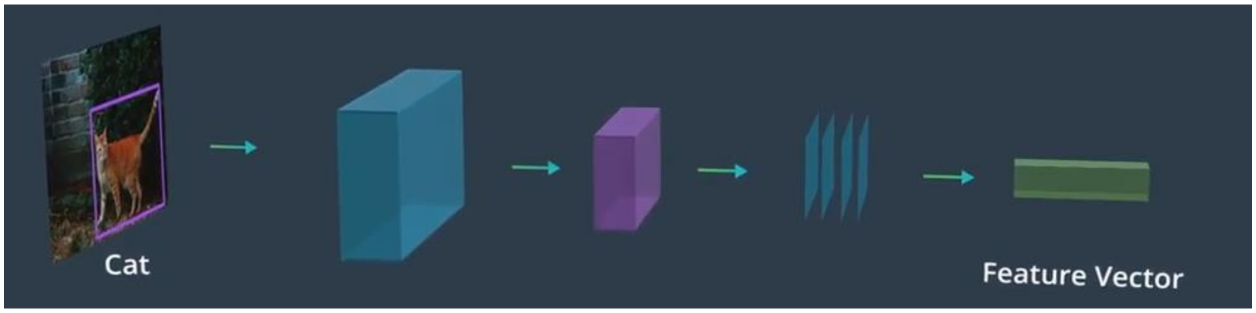
This box can be thought of as collection of coordinates that define the box.

X and y which could be the center of the box and w and h the width and height of the box .



To find this values we can use a lot of the same structure as in typical classification CNN.

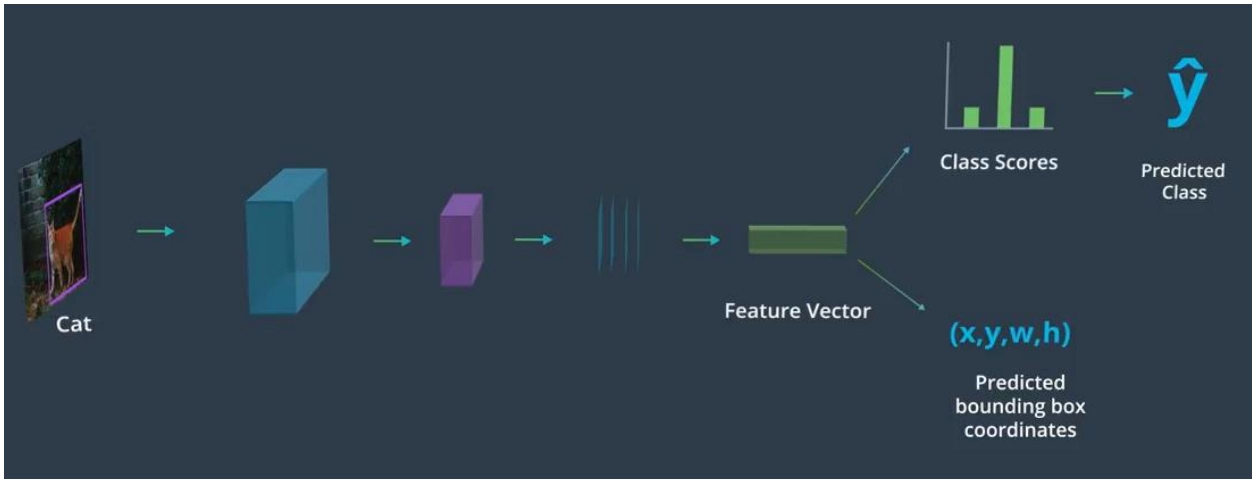
One way to perform **localizations** is to first put a given image through a series of convolutional and pooling layers and create a **feature vector** for that image.



You keep the same fully connected layers to perform classifications and

1-you add another fully connected layer attached the feature vector whose job

is to predict the location and size of a bounding box. (I will call the bounding box coordinates)



In this one CNN, we have one output path whose 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 're assuming that the input image not only has as associated true label but that it 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

Now we know how to use something like cross-entropy to measure the performance of a classifications model

Cross-entropy => operates on probabilities with values between zero and one.

But the bounding box we need something different.

A function that measures the error between our predicated bounding box and a true bounding box.

Next you will see what kinds of the loss functions are appropriate for a regression problem like this, that compare quantities instead of class scores.

Bounding Box and regression

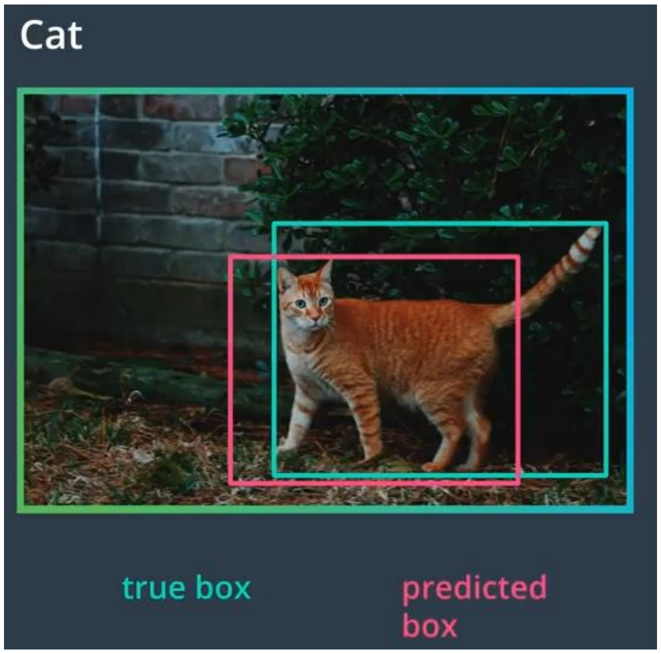
When we train the CNN classify images, we train it by comparing the **output predicted class** with the **true class** label and seeing if they match.

We typically use **Cross-entropy** to measure the error between these this classes Because

Cross-entropy loss decreases as the predicated class which has some uncertainly associated with it gets closer and closer to the true class label. But when we look at comparing a set of points say **locations** or points on a **face or points** that define specific a region in an image**, we need a loss function that measures the similarity between these coordinates values. This is not a classification problem, this is regression problem**

1- **Classifications is about predicting that class label and regression is about predicting a quantity.**

For regression problem, like predicting X_Y coordinates locations we need to use a loss function compares these quantities and that gives us a measure of their closeness.



It also interesting to note that with classifications problems, we have an idea of the accuracy.

1- **If our predicted class matches the true class label then our model is accurate, but with the regression problems we can`t really say whether a point is accurate or not we can only evaluate quantities by looking at something like the mean squared them.**

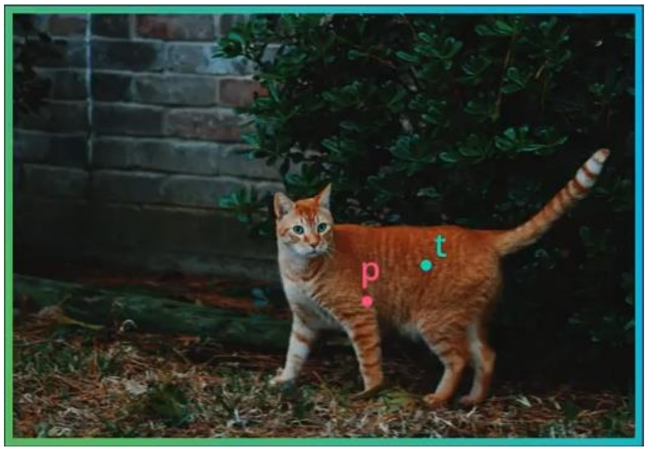
So for **regression problem** we often talk about **models** with a smaller error rather than models are accurate.

To measure error between two quantities, we have a few different types of loss function that we can use.

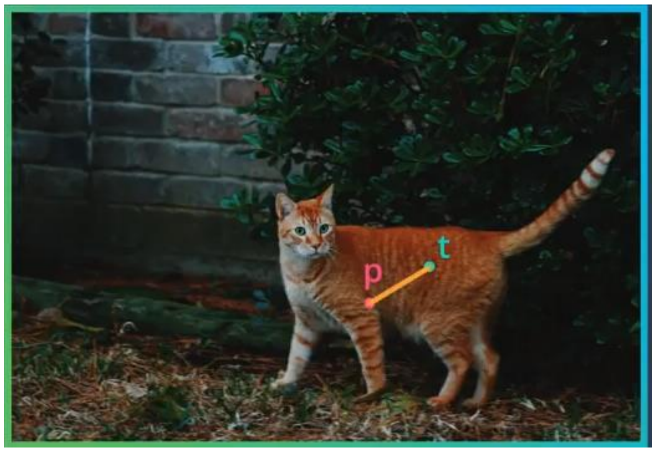
1- The simplest measure is L1 which measure the element-wise difference between a predicted output which I will call P and a Target is T

Say we 're predicting just one point P and X_Y coordinate that indicates the center of an object in an image.

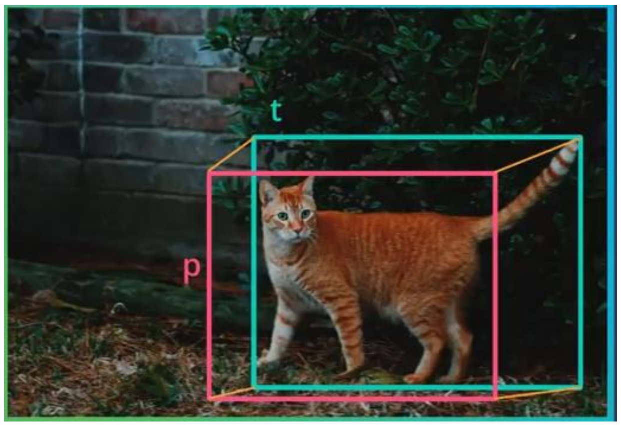
In this case, the loss function we will look at the predicated point P that was generated a CNN, and true target location T of the center of the object,



And L1 loss would return a value that represent the distance between the predicted and true points.

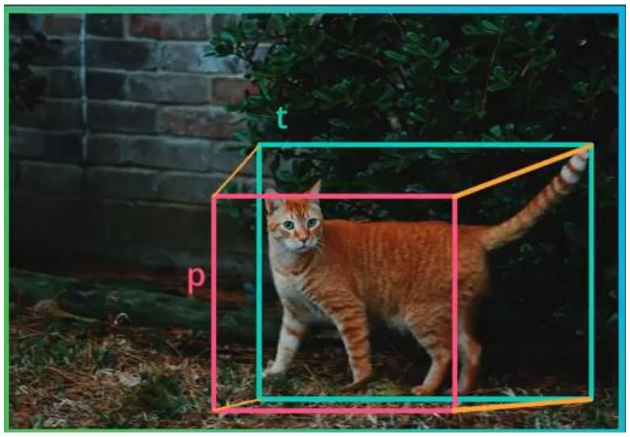


we also have MSE loss, which measure the mean squared error between the elements in a prediction P and a Target T.



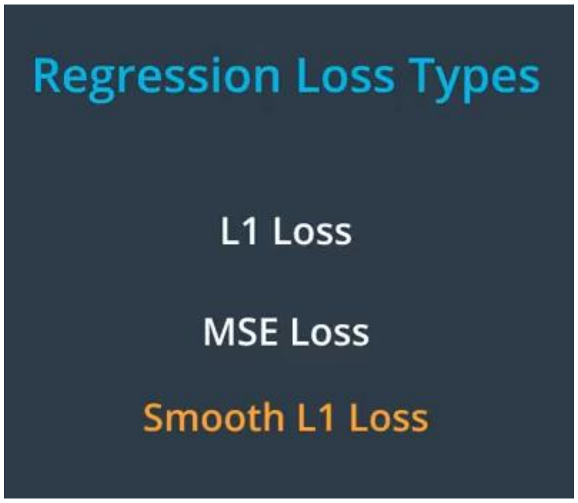
Both of these methods can be good for measuring the distance between points but all loss functions have their strength and weaknesses.

You may consider that **L1 loss function** can be become negligible for لا يكاد يذكر small error values and the MSE loss responds the most to large errors.



And so it may end up amplifying errors that are big but infrequent. Also known as outliers.

There also Smooth L1 loss which for small differneces between predicated and true use a squared error functions and for larger errors, uses L1 so the smooth L1 loss try to combine the best aspects of MSE and L1 loss.


Regression Loss Types

L1 Loss

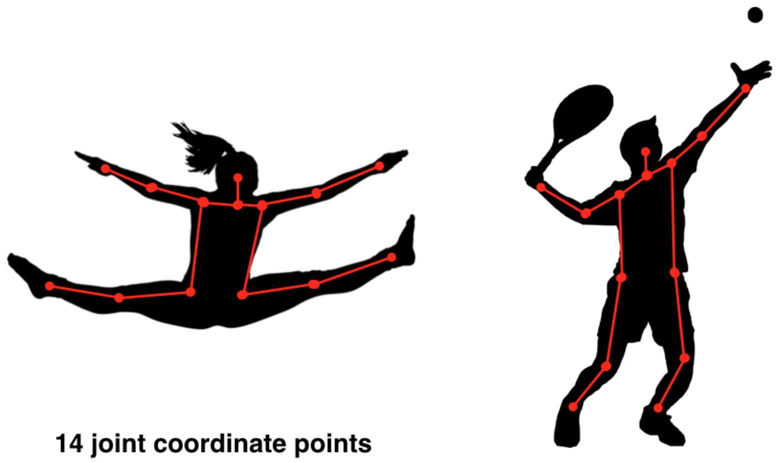
MSE Loss

Smooth L1 Loss

It will really to be up you try these different loss functions, look at how they decreases during training and choose the best one for a given regression task.

Beyond Bounding Boxes

To predict bounding boxes, we train a model to take an image as input and output coordinate values: (x, y, w, h). 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.

Quiz: Loss Values

QUESTION:

Look at the documentation for MSE Loss. For a ground truth coordinate `(2, 5)` and a predicted coordinate `(2.5, 3)`, what is the MSE loss between these points? (You may assume default values for averaging.)

Look at the documentation for Smooth L1 Loss. For a ground truth coordinate `(2, 5)` and a predicted coordinate `(2.5, 3)`, what is the smooth L1 loss between these points?

○ 0.0125

● [✓ 0.55]

○ 0.8125

○ 1.5125

○ 1.825

○ 2.125

Region Proposals

Naïve looking by an image generating a bounding box around that object.

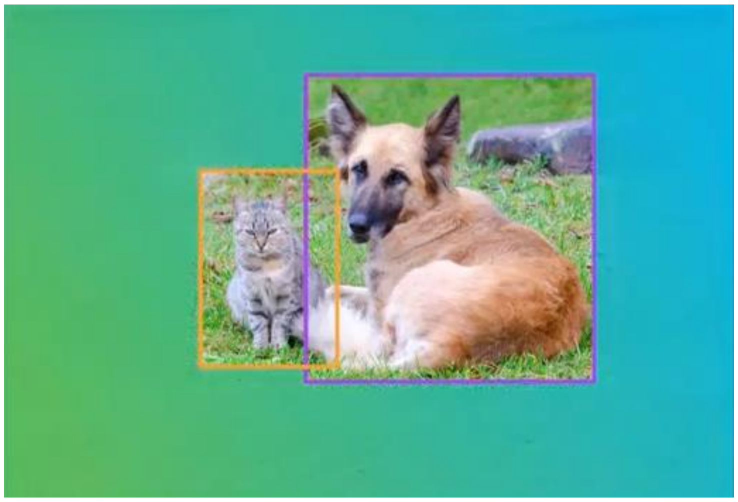
But what if there are multiple objects in an image ?

How can you train a network to detect all of them ?

Well, let`s think about the case where we just have **two objects** in an image.

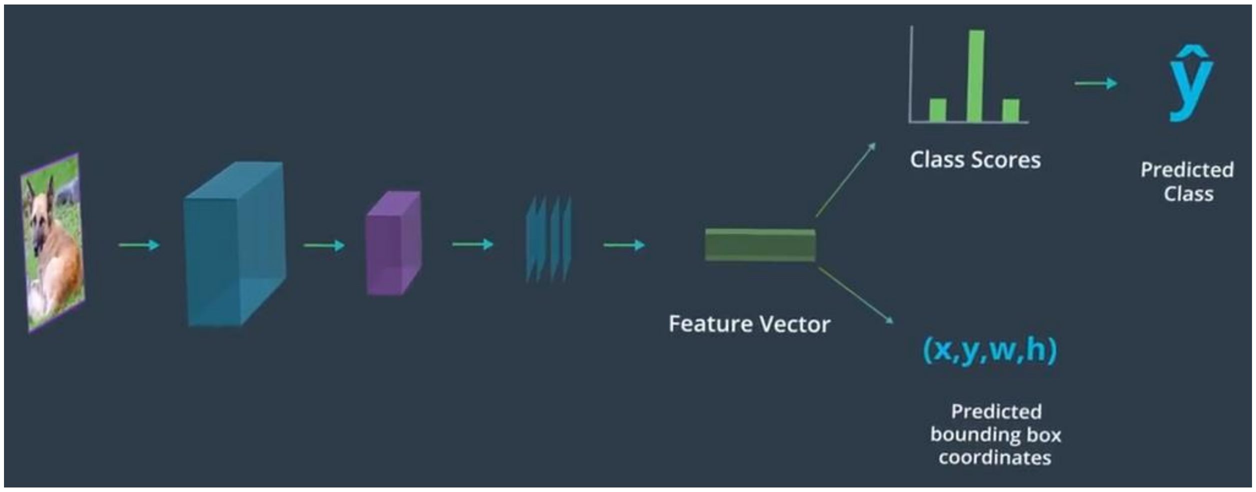
How can localize and label both of these ?

One approach could be to try to simplify the input image and split it into different region,



Each of which only contain one object.

Then we can proceed in the same way as before, putting each region through a CNN that generate one class label and one bounding box.

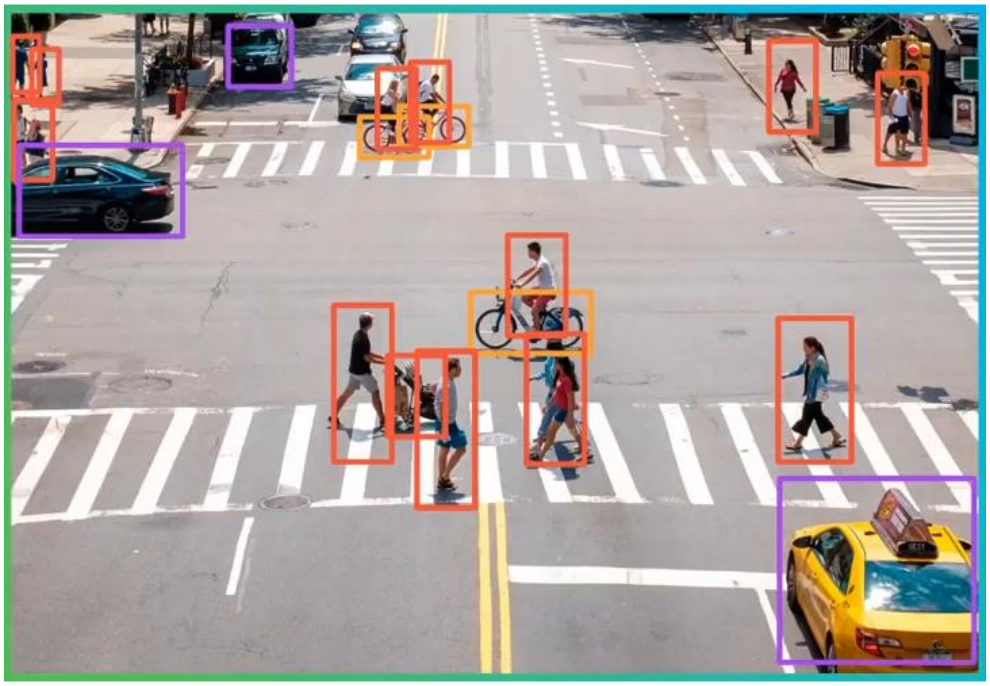


Then you might think, what about if there are there are three objects in an image or four or more ?

The real challenge here is that there`s a variable output. You don`t ahead of time how many objects are going to be in an image a given image and CNNs and most neural networks have a defined unchanging output size.

So to detect a variable amount of objects in any image,

1- **You first must break that image into smaller region and produce bounding boxes and class labels for one region and one object at time.**



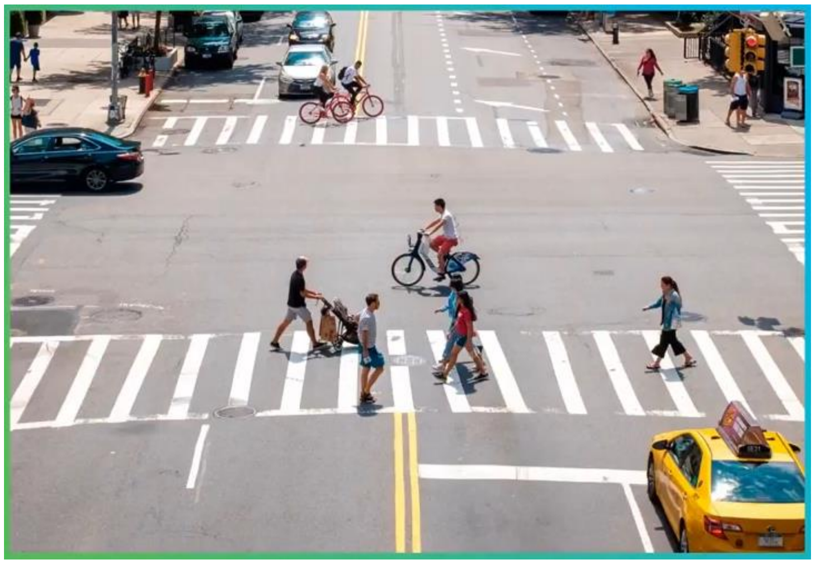
We will learn about techniques for finding these objects regions shortly. Then you should able to locate and classify any objects that appear in an original image whether that`s one object or there or 20

So how can we go about breaking up an image an image into regions ?

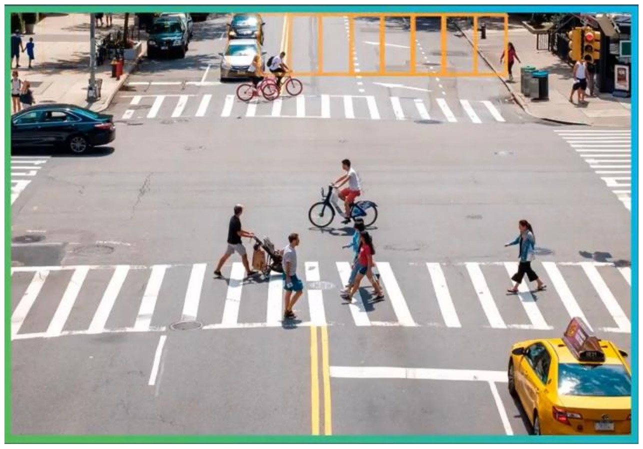
We know that we want these regions to correspond to different objects in the image and we don`t to miss any objects.

We could just make a bunch of cropped regions to make sure we don`t miss an things

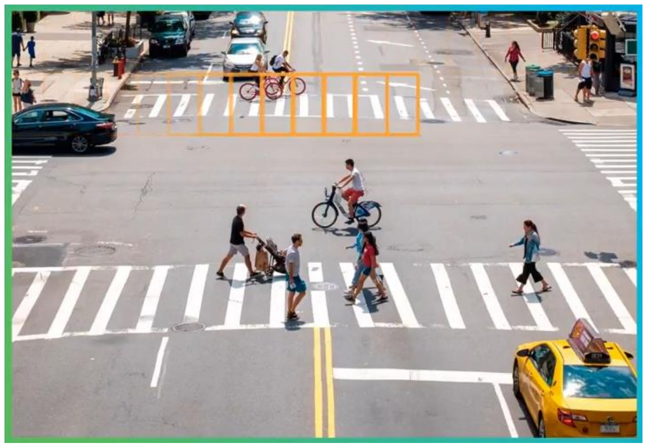
These would mean defining a small sliding window and passing it over the entire image using some for stride to create mini different crops of the original input image.



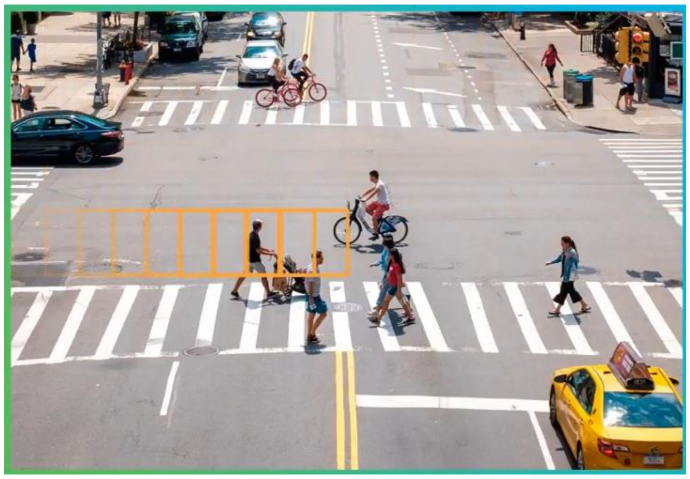
Step 1



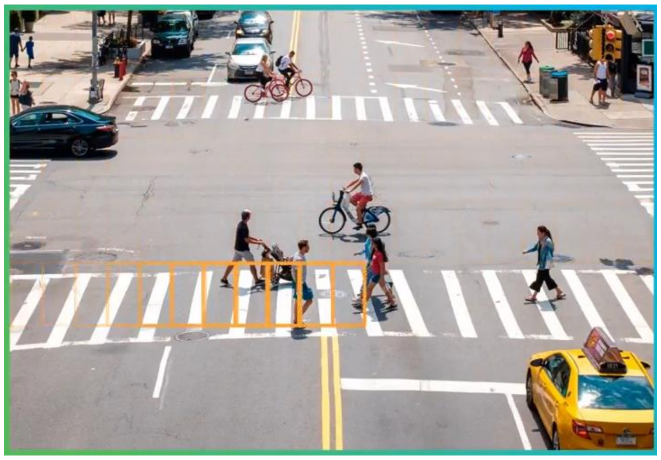
Step 2



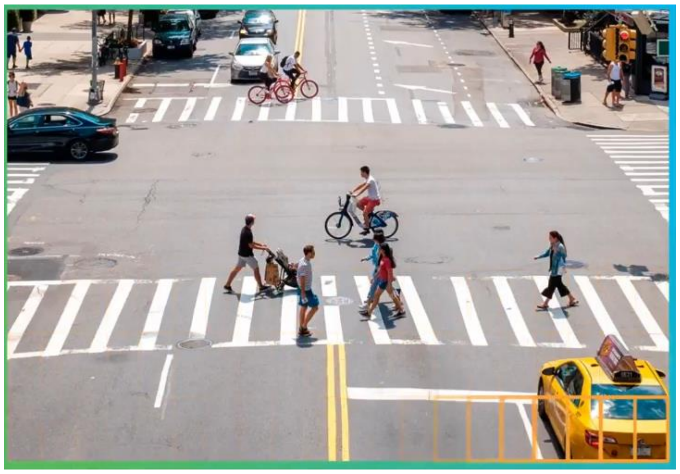
Step 3



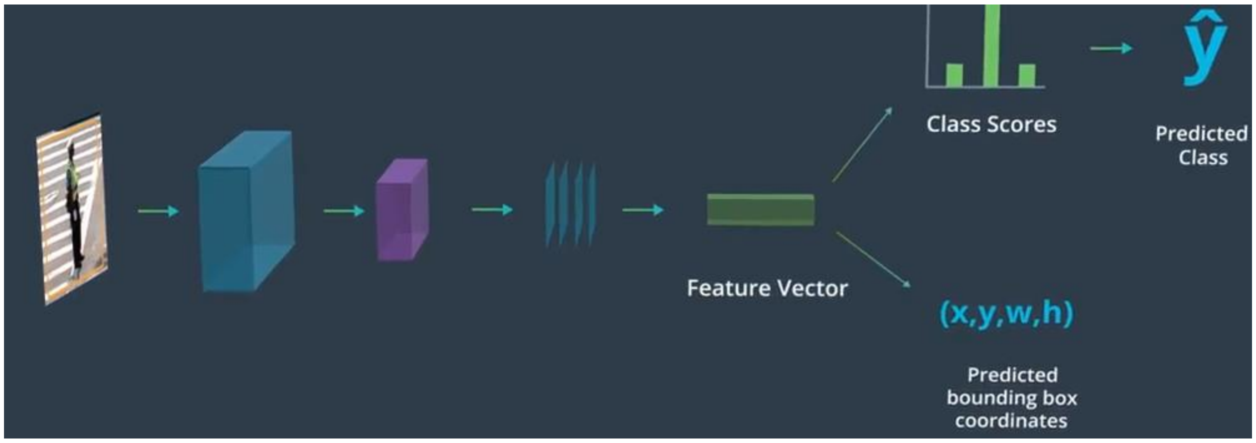
Step 4



Step 5



Then for each cropped region, we can put it through a CNN and perform classifications.



However, this approach produces a huge amount of cropped images and is extremely time-intensive also , in this case most of the cropped images don`t even contain objects.

So how can better choose you these **cropped regions**, especially when objects vary **size and location**

Next ,I want you to think about how you might improves that region selections process.

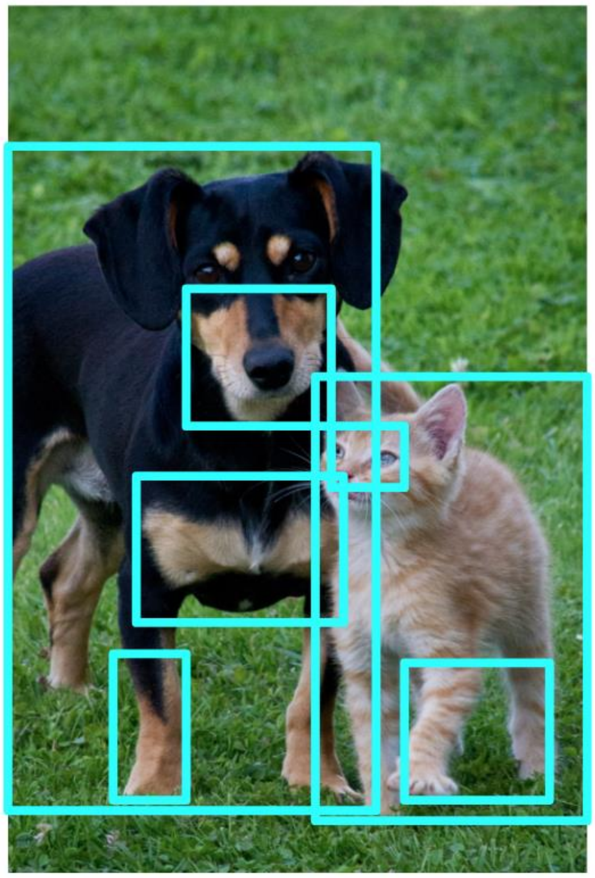
You want make sure not to miss any objects but you also don`t want to put a huge number of cropped regions through a CNN.

Consider the below image, how do you think you would select the best proposed regions; what criteria do good regions have?

ANSWER:

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

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.

To localize an classify multiple object in image we want to be able to identify a limited set of cropped regions for a CNN to look at

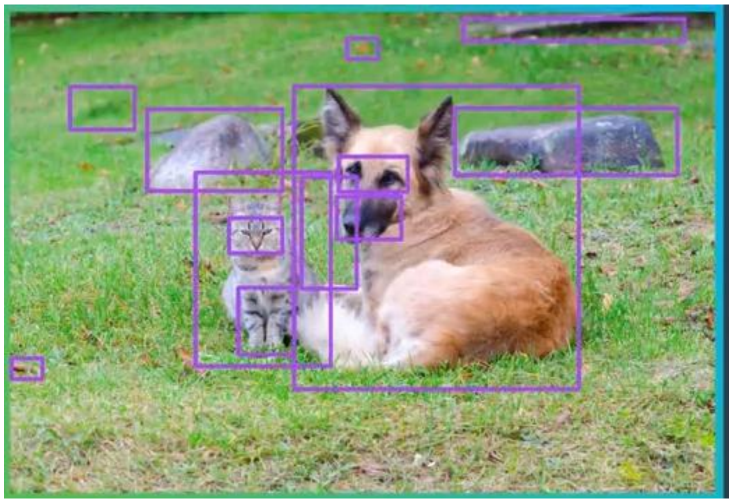
In the idea case.

We would generate **three perfectly cropped regions** for three different object in an image.

To approach this goal and generate a good limited set of cropped regions, the idea of region proposals was introduced

Region proposals

Region proposals give us a way to quickly look at an image and generate regions only for areas in which we think there may be an object.



we can use traditional computer vision techniques that detect things like edges and textured bobs to produce a set of regions which objects are most likely to be found.

Area are **similar texture or the same unifying boundary**

For example

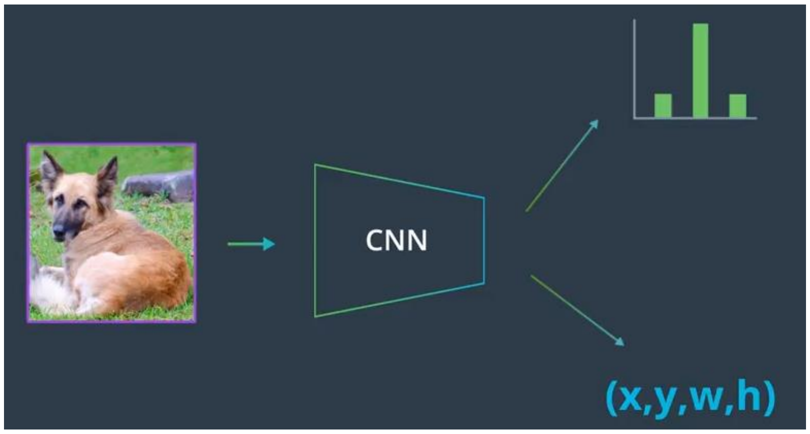
These proposals often produce noisy non-object regions, but they also very likely to include the regions in which objects are located on so the noise is considered a worthwhile cost for not missing any objects

So let`s see how the looks when incorporated into a CNN architecture

We can use **region proposals algorithms** to produce a limited set of cropped regions.

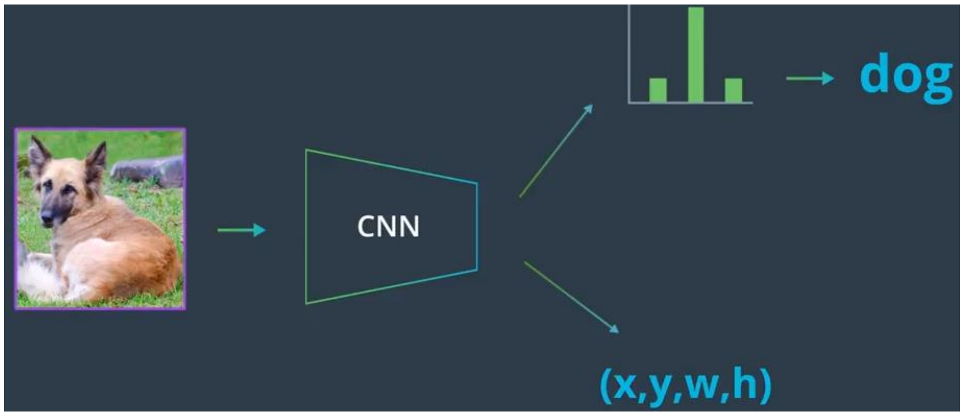
After called regions of interests or ROIs

And then put these regions through a classifications CNN, one at a time and see what kind of class label the network predicts for each crop.

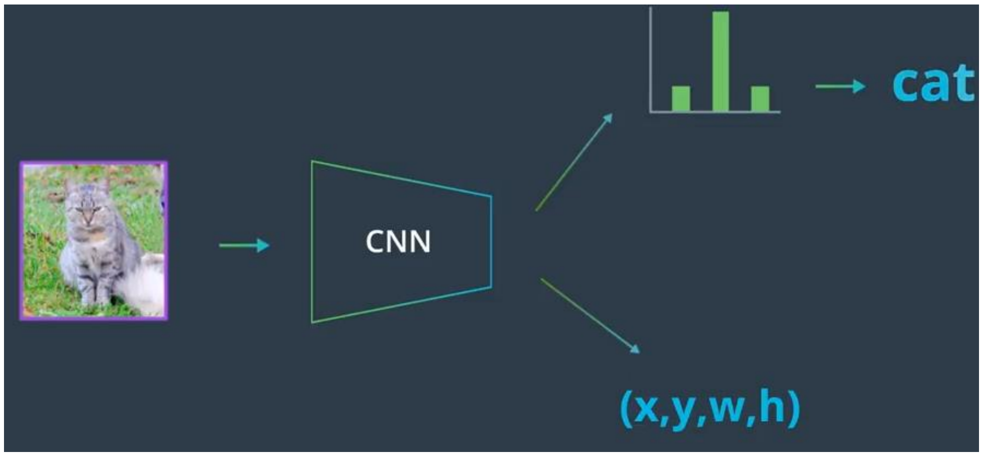


This model is called an R-CNN Which stands for convolutional neural network.

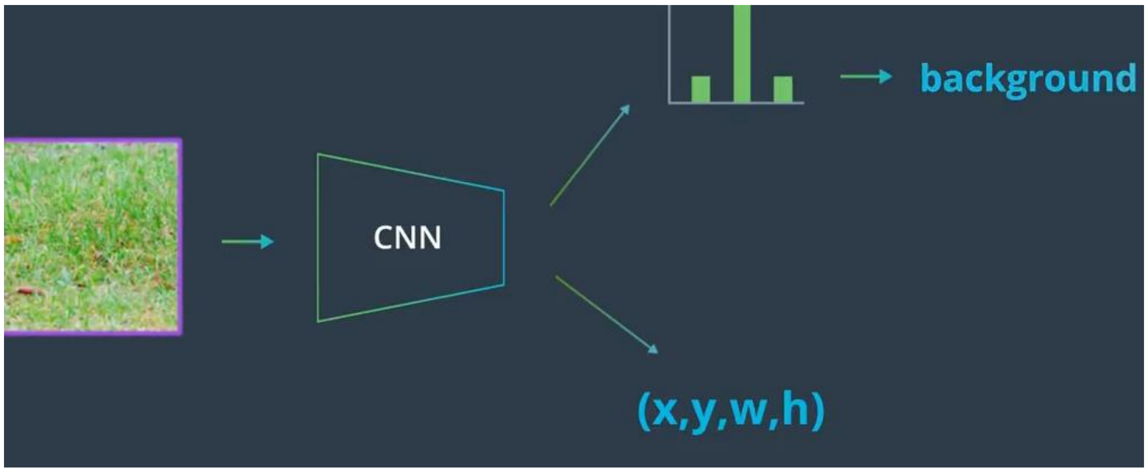
The R-CNN produces a class for each region of interest, and so it can identify the region that is a dog the region that is a cat in an image.



That is a cat in an image



In this case we also include a class called background, that`s meant capture any noise regions.



Since the regions are often different sizes they first need to **be transformed** and warped into a standard size that a CNN can accepts as input.

Know, The main shortcoming of this method is that it still time intensive because it requires that each cropped region go through an entire CNN before a class label to be produced.

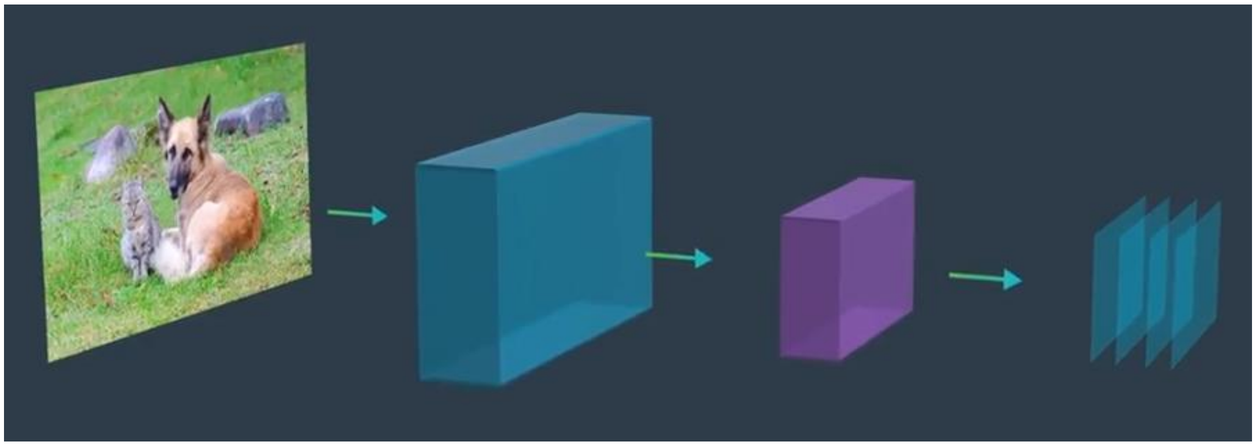
Next, we will see some examples of region based CNNs that aim to speed up this **process and efficiency classify multiple objects in an image**.

A **recurrent neural network** (**RNN**) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs

FAST R-CNN Architecture

Next advanced in region based in r-cnn is the fastR-CNN architecture.

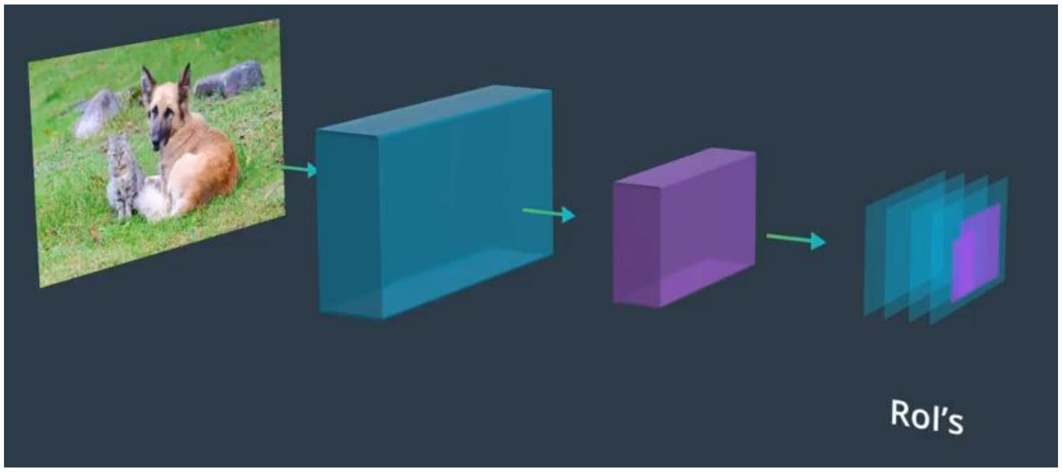
In steading of processing each region of interest individually through a classifications CNN,



This architecture runs the entire image through a classifications CNN only once

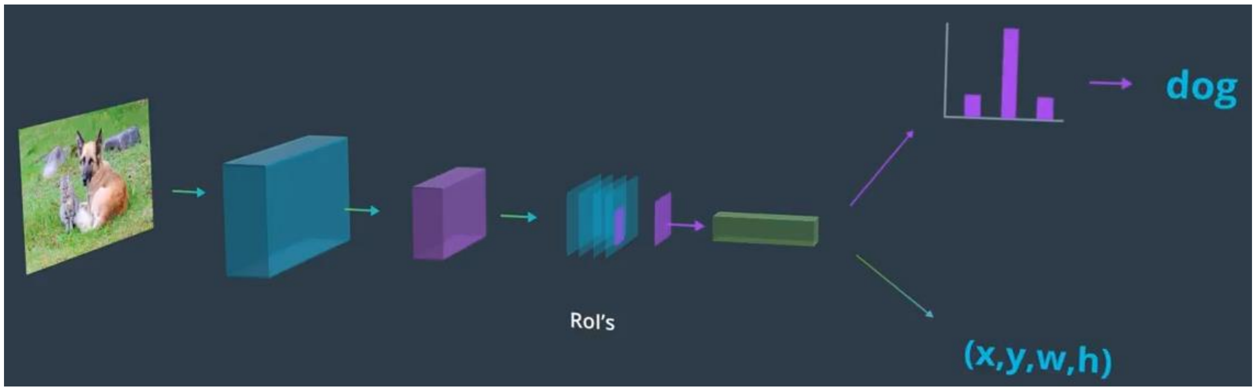
The image goes through a series of convolutional and pooling layers and at the end of these layers, we get a stack of feature maps

We still need to identify regions of interest but instead of cropping the original image, we project these proposals into smaller feature map layer.



Each region in the feature map corresponds to a larger region in the original image.

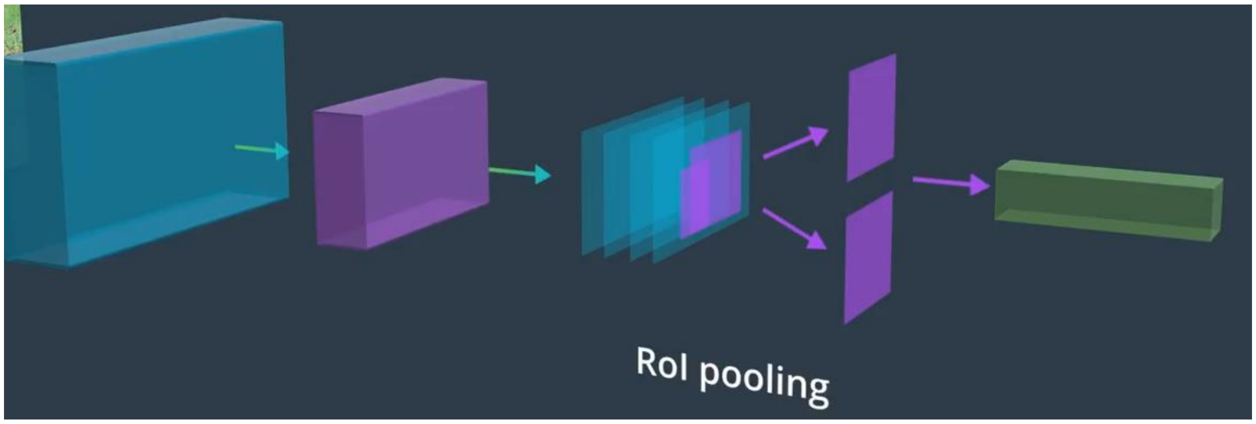
So, we grab a selected regions in the feature map and **feed** them one by one into a fully connected layer that generates a class for each of these different regions.



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 our desired outputs

Again, we have to handle the variable sizes and these protections 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.



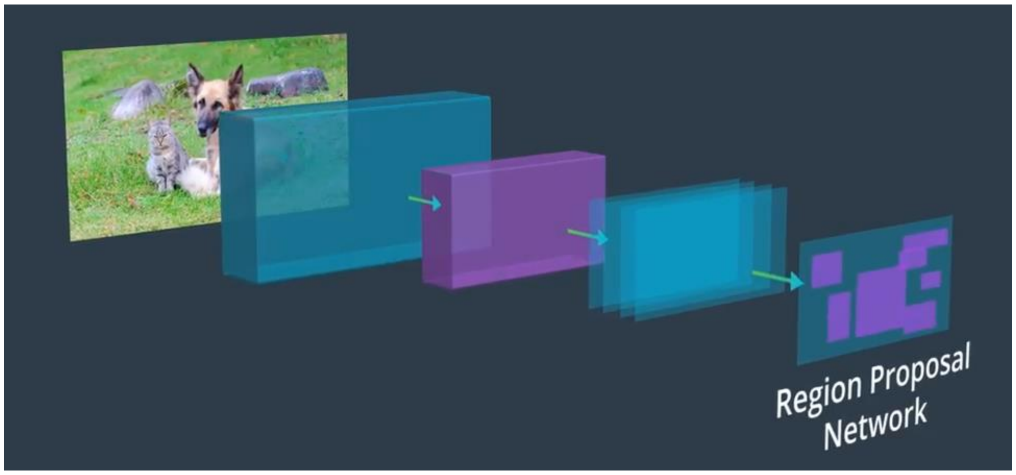
Now this network is faster than R-CNN but it`s still slow when faced with a test image for which it has to generate region proposals and it`s still looking at regions that do not contain object at all.

The next architecture we will look at aim to improve this region generation step.

To speed up time test on time through a network and detect all the object in it, we want to decreases the time it takes to from **region proposals** . for these we have the faster R-CNN architecture

Faster R-CNN learns to come up with its own region proposals.

It takes an input image, runs it **through** a CNN up until a certain convolutional layer just like Fast R-CNN but this time it uses the produced feature map as input a separate region proposals network.

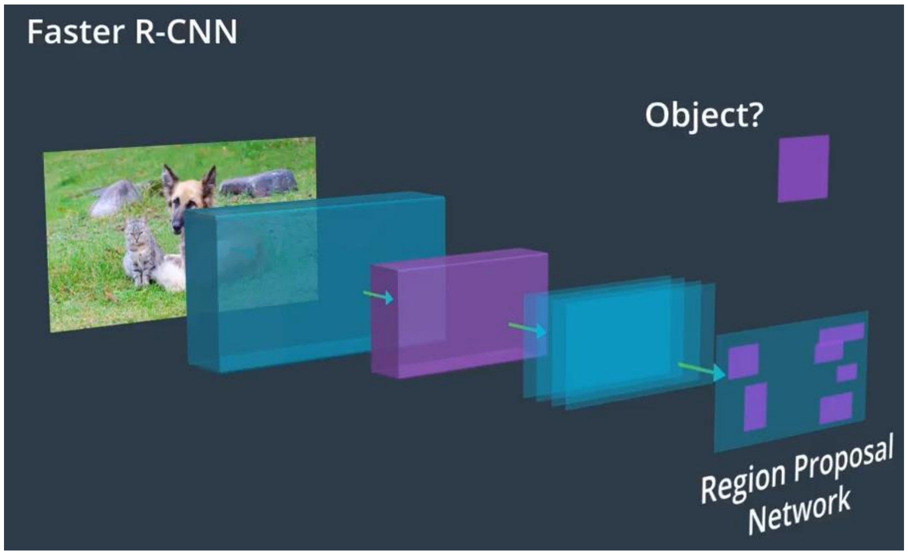


So it predicates its **own regions** from the feature produced inside the network.

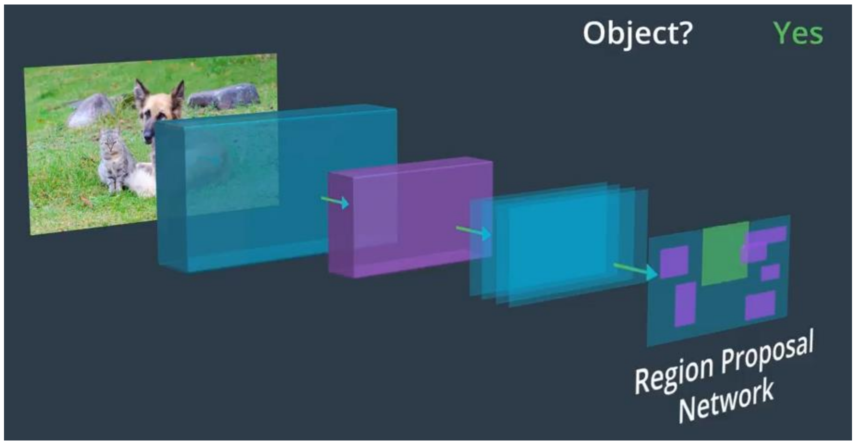
If an area in feature map is rich in detected edges or other features its **identified as a region of interest.**

Then this part of a network does quickly binary classification.

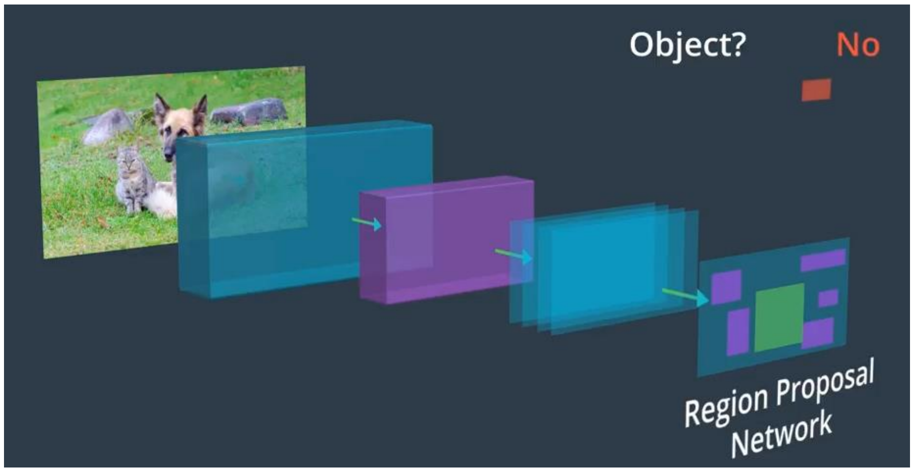
For each ROI it checks whether or not that region contains an object.



If it does then the region will continue on and go through the classifications steps.

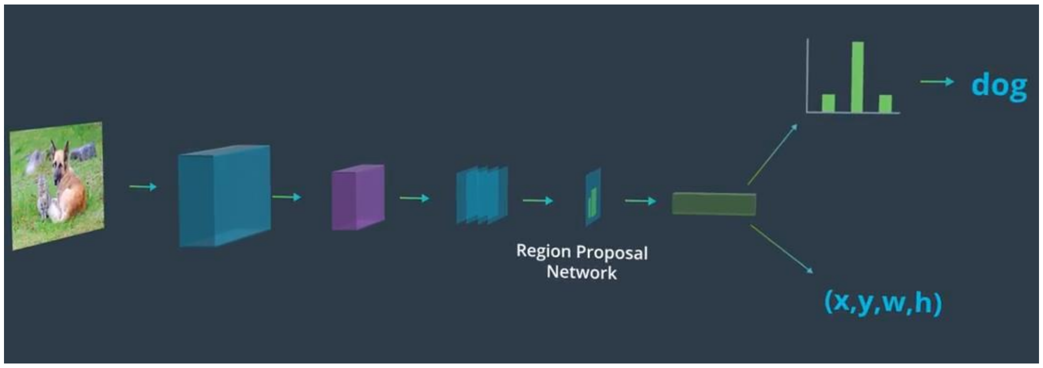


If it dose not then the proposals is discard.



Once we have the final region proposals, the rest of the network looks the same as FAST R-CNN.

It takes cropped regions from the feature map and learns to classify those regions.



By eliminating the analysis of non-object regions, this model is he fasted of all region based CNNs 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**? 1- **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 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 i**t 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**!

Why is speed important?

Multi-object identification consists of **locating** and **classifying** different objects in an image. Some models trade off accuracy for speed. What kinds of applications require speed in object recognition? Check all that apply.

☐

Accurately classifying images of cancerous vs. non-cancerous tissue.

[✓]

Pedestrian detection for an autonomous vehicles.

☐

Identifying people's faces in a set of images.

[✓]

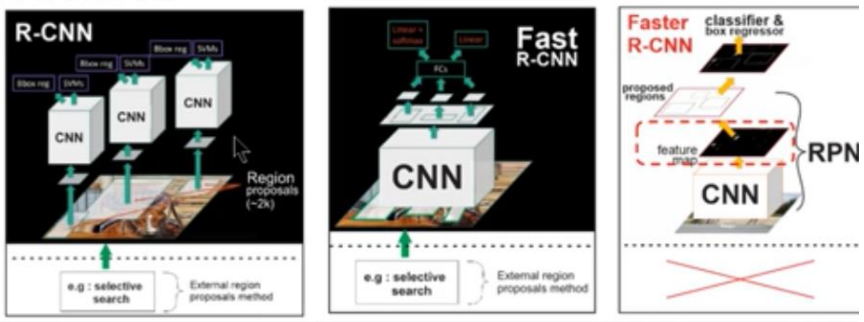
Tracking people's faces so that a camera can focus on them.

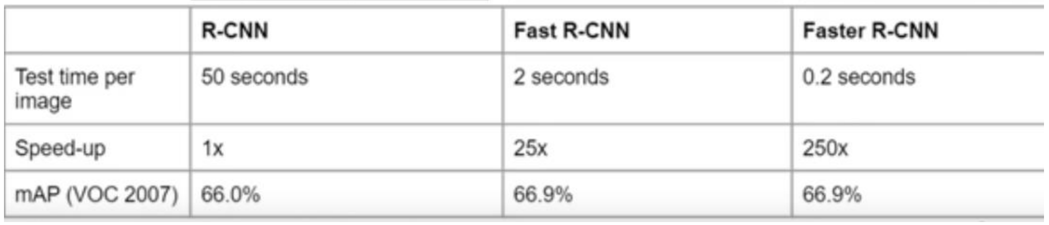
SOLUTION:

- Pedestrian detection for an autonomous vehicles.
- Tracking people's faces so that a camera can focus on them

Faster R-CNN Implementation

If you'd like to look at an implementation of this network in code, you can find a peer-reviewed version, at [this Github repo](#).





	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x
mAP (VOC 2007)	66.0%	66.9%	66.9%

Summary:

Really learn about revolution of CNN architecture in this lesson especially about how to detect multiple objects in an image using region proposals.

As we mentioned at to start, simple image classification problems are really the most basic applications of CNNs And if we ‘re aiming to reach levels of human understanding, we have to add complexity to these networks so that they ‘re able to recognize multiple objects and their relations to one another.

You may also able to wondering if there `s a way to perform object recognition without region proposals and there is.

There‘s whole other family of object detections network that don‘t use region proposals.

One of this YOLO.

YOLO stand for you only look once, and another is

SSD Stand for single shot detection.

These both came out round the same time, and the next lesson you will learn all about YOLO and get chance to work with the code implementation of YOLO Detection Network.