Course 4 Week 3

Object localization

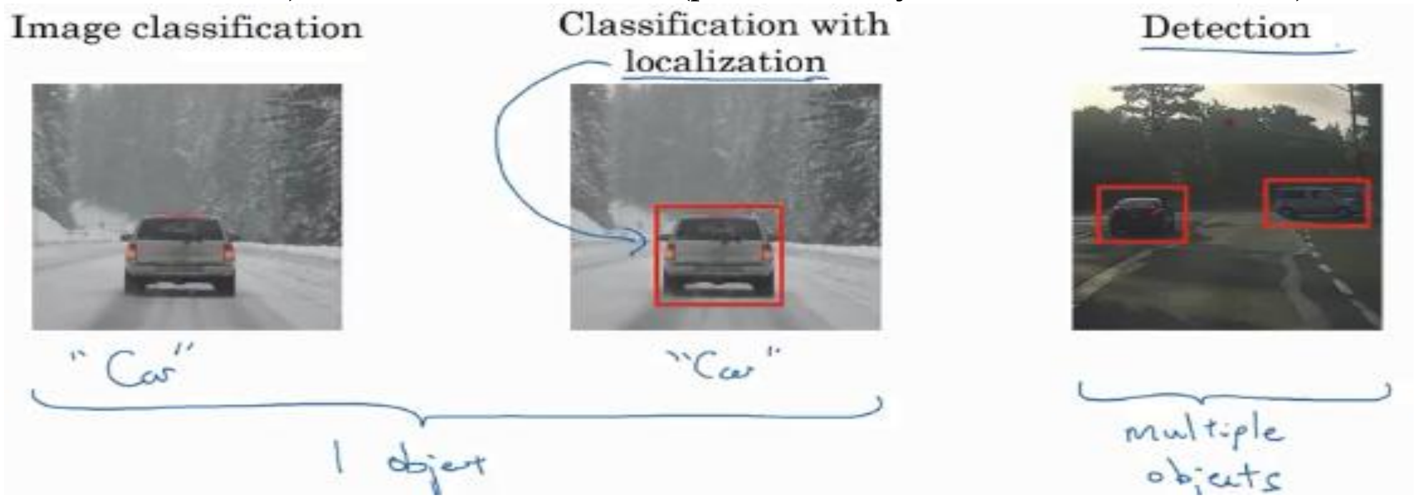
We have been seeing throughout all what we have taken that our main concentration was on image classification where we have c classes and our network is responsible to tell each input image belongs to which class. Localization on the other hand means to not only tell each input image belongs to which class but also tell the boundary box of the single object found in te image. There is also another task called detection where here we have multiple objects in each input image and these objects could belong to similar or different classes and so we have to detect and localize each object of them (as long as they belong to a pre-defined class).

To sum up:

1) Image classification: A single object to be classified (assign a class label to it).

2) Classification + Localization: A single object to be classified (assign a class label to it) and to be localized (put a boundary box around this object).

3) Detection: Multiple objects of similar or different classes each to be classified (assign a class label to each of them) and each to be localized (put a boundary box around each of them).



How to apply localization ?

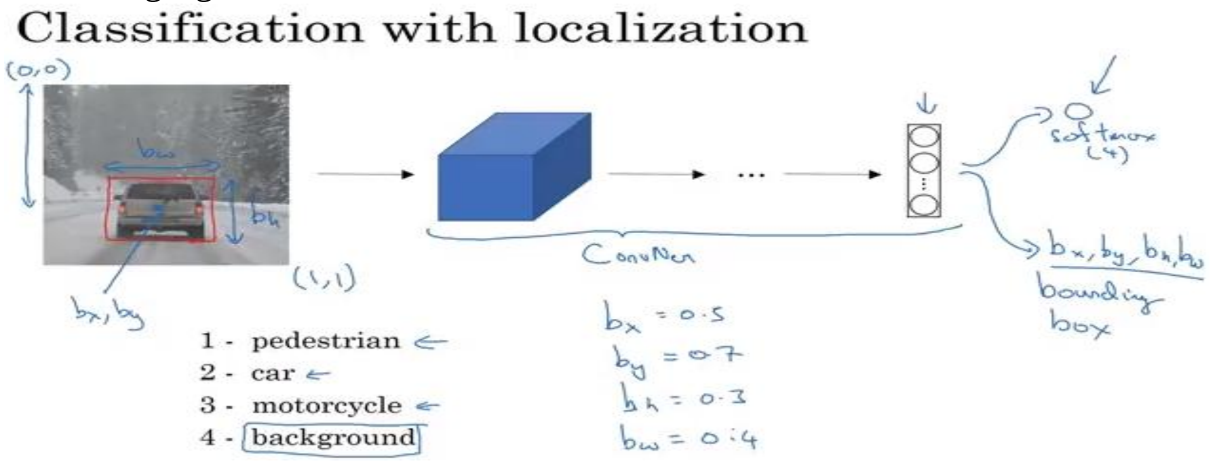
We have discussed before that to do classification tasks, we should go along the pipeline as follows : Input image → Convnet → Softmax layer with c classes → $\hat{y}$

This was the typical path to classify, so what modifications we do to apply localization as well?

Instead of making softmax outputs only predictions representing probabilities of each class, we will add few more outputs representing the bounding box (b_x, b_y, b_h & b_w) where b_x & b_y represents the co-ordinates of the center of the object and b_h & b_w represents the height and width of the bounding box respectively. Now instead of having dataset containing only class labels for each training example (supervised learning), we will also have 4 values representing the boundary box so now dataset will contain: [Image + Class label + boundary box of object].

Note: The notation convention to be used along the main course will be that the upper left corner has co-ordinate of (0,0) and the lower right corner has co-ordinate of (image_width, image_height) or (1,1) depending on the topic.

Check the following figure to conclude what said:



Defining target label y

$$y=\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ \vdots \\ C_c \end{bmatrix}, \text{ where:}$$

- P_c defines is there any object ➔=1 OR just a background (no object)➔=0?
- (b_x, b_y, b_h & b_w) defines the boundary box where (b_x, b_y) are the co-ordinates of the center of the box and (b_h & b_w) are the height and width of the box.
- C_1, C_2, …, C_c are the pre-defined classes to classify the object in the image by putting 1 on the right class and 0s on the remaining.

Notes:

1-There is a single object in the image as it is a localization problem.

2- In our example at the top of the page, Number of classes is 3 not 4 as the background is not class so we have C_1,C_2,C_3 while the background is identified by P_c to tell if there is an object or just a background.

On our example: $y=\begin{bmatrix} 1 \\ 0.5 \\ 0.7 \\ 0.3 \\ 0.4 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ given that C_1 is pedestrian, C_2 is car, C_3 is motorcycle.

Note: If the image is a background so P_c=0 and all the remaining will be don't cares (?).

Loss function

$$\ell\,(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2, (\hat{y}_2 - y_2)^2, (\hat{y}_3 - y_3)^2, \ldots, (\hat{y}_{5+c} - y_{5+c})^2, & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & , \text{if } y_1 = 0 \end{cases}$$

Note: 5+c represents the length or size of vector y so if we have 3 classes as our previous example so we have y_1=P_c & y_2,…, y_5=(b_x, b_y, b_h & b_w) & y_6, y_7, y_8 = C_1, C_2, C_3.

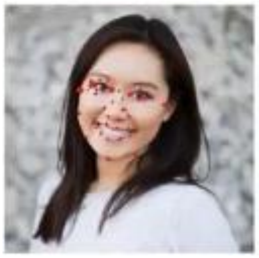
Landmark detection

Instead of having bounding box with (b_x, b_y, b_h & b_w) as an output to localize an object, here we can have some cases where we need just x & y co-ordinates of some important points in an image to be recognized, these points are called landmarks.

For example, assume you are having a face recognition task where you want to know where is the corner of the right eye so this in fact is a single co-ordinate (x,y) that represents the corner of the eye (red dot in the eye's corner)➔

So now our final layer will produce co-ordinates l_x, l_y to represent this landmark.

Similarly if you want 4 landmarks that represents the 2 corners of both eyes so now our final layer will have to produce (l_{1x}, l_{1y}), (l_{2x}, l_{2y}), (l_{3x}, l_{3y}), (l_{4x}, l_{4y}) to represent the 4 landmarks (2 corners of each eye). ➔



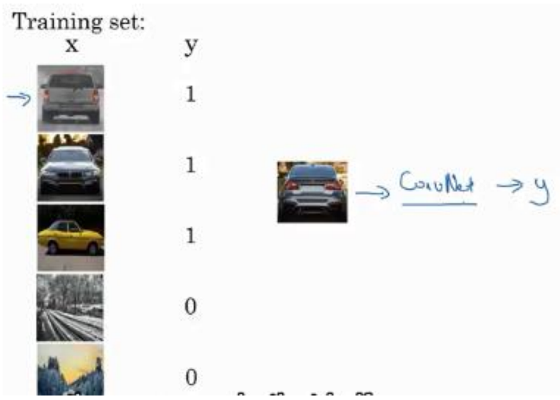
More generally for face recognition we will have several landmarks to represent everything in the face (eyes, nose, mouth, cheek points, face edges, …) so that these landmarks represent the face features so our final layer will produce (l_{1x}, l_{1y}), (l_{2x}, l_{2y}), …, (l_{nx}, l_{ny}) to represent position of each landmark assuming having n landmarks. ➔

Note: we have an additional output which tells whether there is a face in the image or not which is similar to P_c in localization. So our final layer output size will be (n×2)+1 assuming having n landmarks.

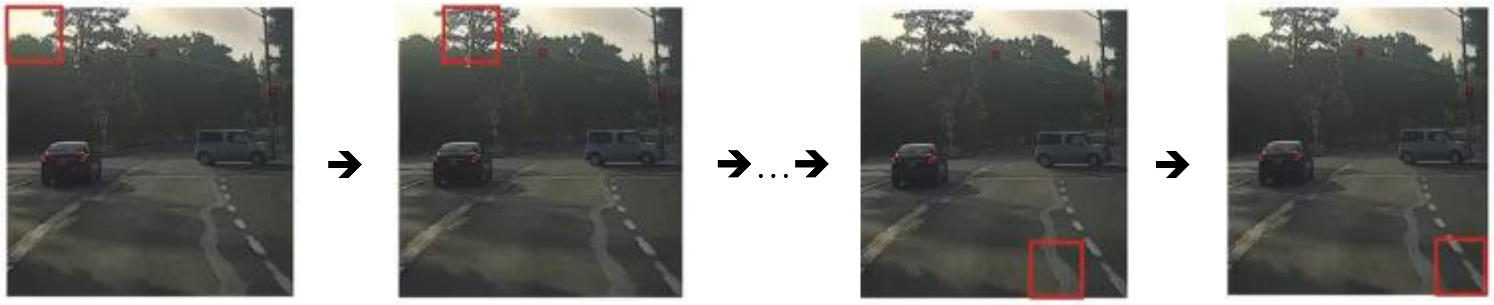
Of course landmark detection is not used only for face recognition but it has millions of other tasks as for example person's pose detection (walking, running, bending, kicking, …) ➔

Real-world application for landmark detection: From the famous real-world applications is the augmented reality used in snapchat filters that detects landmarks of the face to put the filter on your face, Also face unlock in the recent smart phones are using landmark detection.

Sliding window for object detection



We have said before that object detection is mainly having multiple objects of similar or different classes in an image and our task is to localize each object and assign class label to it. The basic idea to do that is to have a dataset of cropped images that is labeled to a specific class say we have cars or no cars as a simple example as shown ➔ then train our network on this dataset using convnet similarly to what we were doing in image classification tasks (now training phase is over) then apply sliding window of specific size over the test image and crop this window and pass it to the trained network to see whether there is a car or not in this cropped window as shown below in the test image:



As shown above we have slided a window over the test image and each time we crop this window and pass it to the convnet to say whether there is a car in the window or not and if there is a car it saves the boundary box position relatively to the test image as whole.

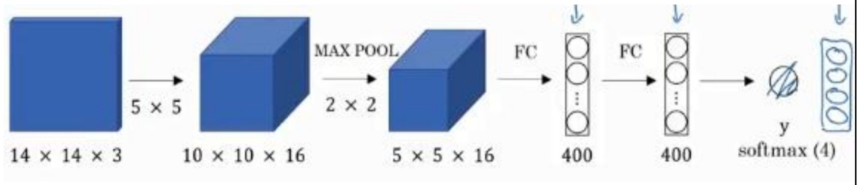
But may be the sliding window size was not right so that is why it couldn't detect the car so that is why we apply the sliding window several times each time with different window size:

This method has a problem of high computational cost as iteratively we are doing several sliding windows with different sizes and each sliding window is moving along the whole image where each time you apply the cropped part to a convnet which is an expensive task to be done.

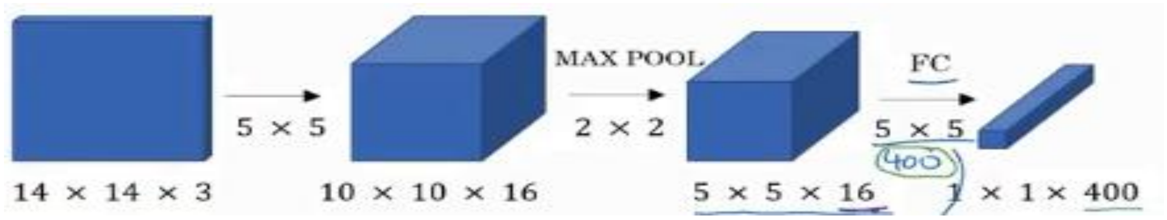
Before getting in how to solve this problem, let us conquer another idea then get back to the solution for this problem that this idea will be used in.

Turning Fully Connected (FC) layer into convolutional layer

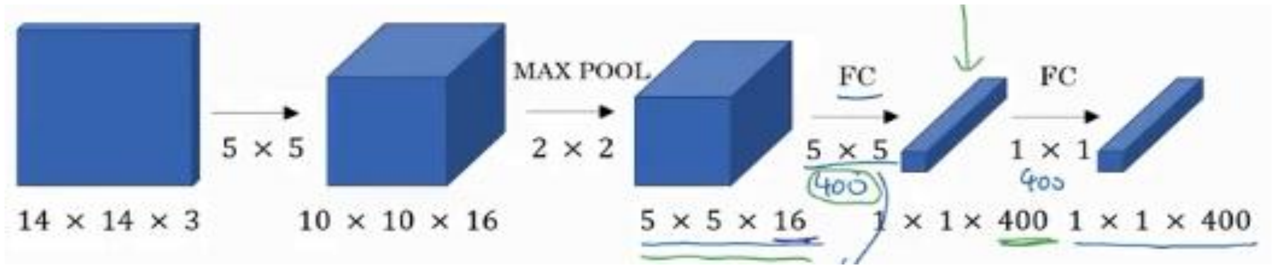
Assume having a network as shown➔



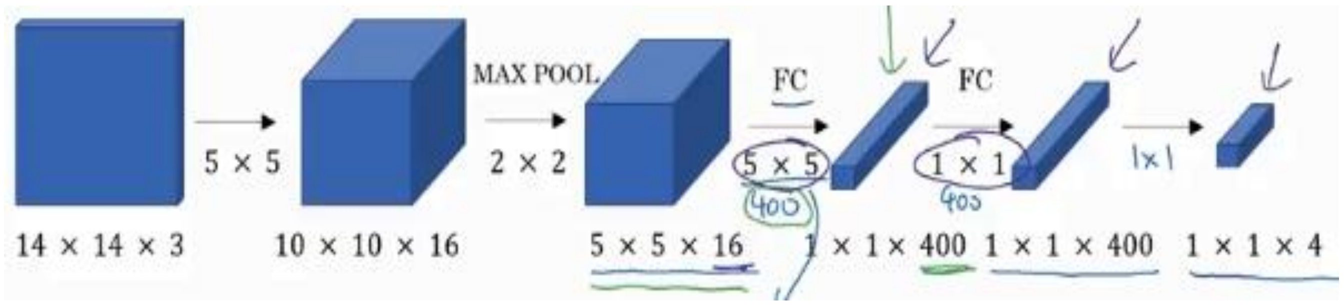
Now we want to have a fully connected layer but in form of convolutional layer, so what we do is that we will convolve the last layer before the required FC layer with n filters each of the same dimension (and channels of course) of this last layer before required FC layer where n is the length of FC layer so in our example the layer before the first FC is of dimension 5×5×16 so we will convolve it with 400 filters each filter is of dimension 5×5×16 so each convolution will result in a 1×1 result (single value) and since we have 400 filters so the final output will be of dimension 1×1×400 as shown:



What about the next FC layer ? Similarly, we will look at the layer before it so its size is 1×1×400 and the size needed for FC is also 400 so we will convolve this layer with 400 filters each of them is of size 1×1×400 so that the output now is of size 1×1×400 as shown:

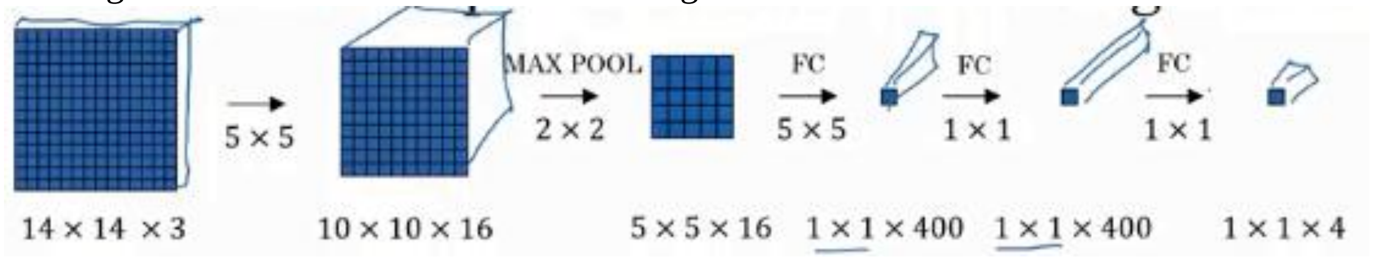


Similarly with the softmax layer which has 4 outputs (4 classes) so to form this layer we should convolve the layer before it with 4 filters each of dimension 1×1×400 to end up with an output of dimension 1×1×4 as shown:

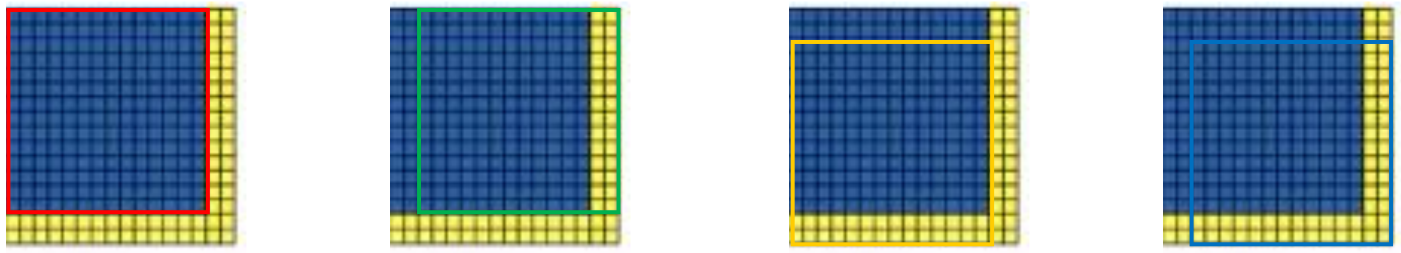


Now, How to solve the problem of high computational cost of sliding windows?

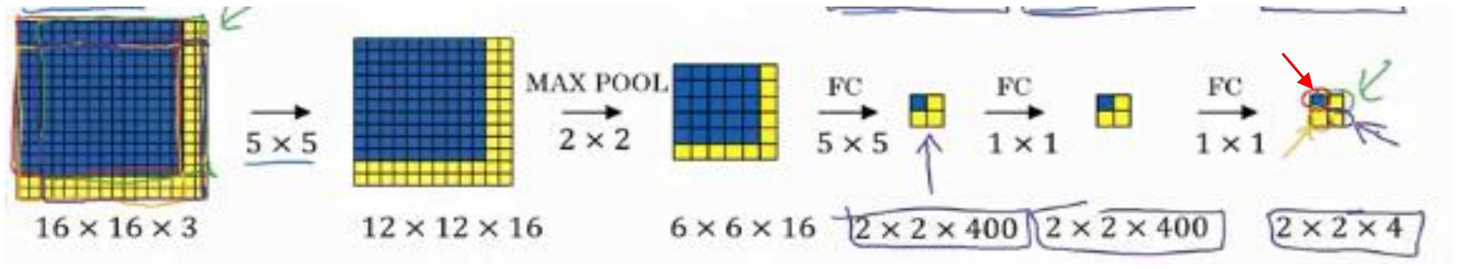
This is solved using overfeat method which depends on convolutions. Assume having input RGB images of size 14×14 and the same training network we used before as shown:



So now our window is of size 14×14 so if we assume that our test image is of size 16×16 and we want to apply this 14 by 14 sliding window with stride=2 (means that every time we slide the window over the testing image by shifting 2 pixels) so by doing that we will find that we have 4 possible windows to be applied as shown on the test image:

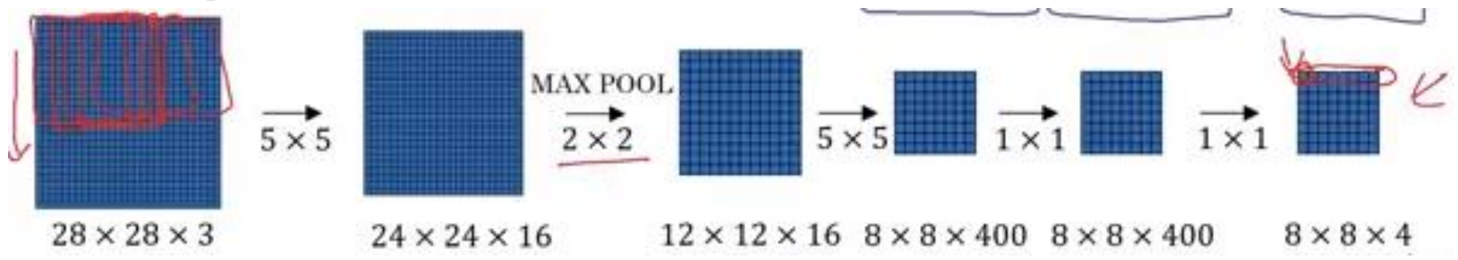


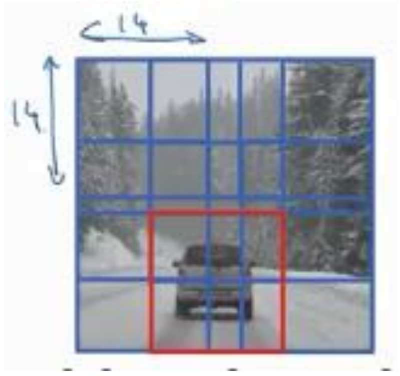
So what we were doing is that each window of this 4 windows will crop the test image and passes the cropped photo to the convnet to decide whether there is an object of the 4 classes or not but this is expensive computation to do as we will repeat this 4 times for each of the 4 windows taking into consideration that we are giving a simple example where there is only 4 possible windows but if stride=1 there will be 16 possible windows and if the image is larger and sliding window size is smaller there will be much much more so what overfeat method says is to apply the 16×16 test image as it is without any cropping to the same pre-trained network and the output will contain all the 4 possible outcomes of the 4 windows internally so the final output will be 2×2 instead of 1×1 so we will have the 4 vector output where each one represents the output of one of the windows as shown:



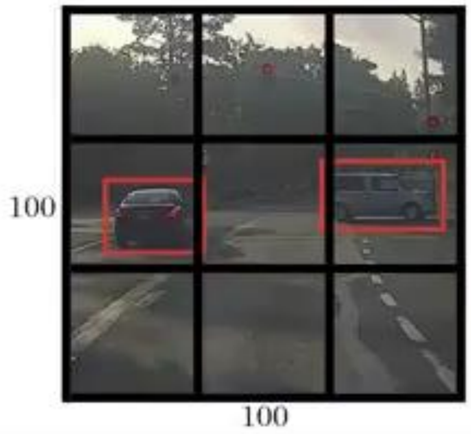
So the vector with red arrow is for the red window, the vector with green arrow is for the green window and so on … ➔ So now we have 4 vectors (representing the 4 windows) each of length 4 (representing the 4 classes) so instead of doing the same path 4 times for each window, now we do it just one time with the same result.

Similarly if the test images is 28×28 and we have the same sliding 14×14 window that is slided with stride=2 so now we have 64 possible windows so instead of doing 64 paths through the trained network each with the cropped window of the 64 windows now we pass it once and get the whole output at once as shown:



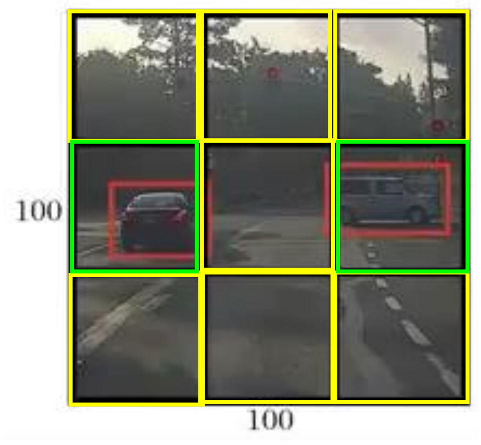


Bounding boxes predictions



Sliding windows is still a not very efficient way for predicting an accurate bounding box as there could be no bounding box that perfectly match the object so we will discuss now how to make these boxes more accurate and matching to the object.
What we will do is that we will apply grid cells to the image in our algorithm say for example we apply 3×3 grid cells to our 100×100 image as shown ➔
Now we will apply what we have done in classification + localization part for each grid cell so

for each grid cell we will have y for the training where y = $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ \vdots \\ C_c \end{bmatrix}$ and as we have 9 grid cells then we will have 9 y's each represents 1 grid cell.



Each of the 7 yellow grid cells has no objects (I mean here that the center of the object is not in these grid cells) and the 2 green grid cells are the ones having objects inside them (I mean here that the center of the object is in these grid cells) so the 7 yellow grid cells will have the following y $\rightarrow \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$ where (?) is don't care while the 2 green grid cells will have the following y's $\rightarrow \begin{bmatrix} 1 \\ b_{1x} \\ b_{1y} \\ b_{1h} \\ b_{1w} \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ b_{2x} \\ b_{2y} \\ b_{2h} \\ b_{2w} \\ 0 \\ 1 \\ 0 \end{bmatrix}$ assuming that we have 3 classes and class 2 is the cars classes. so now the output size is 3×3×8. So now during training we are passing an input image that a grid cell of any size (3×3 in our example) into the conv network then having an output as shown above including a more precise bounding box with any aspect ratio rather than like sliding window which has specific aspect ratio represented in the size of the sliding window all over the image.

Notes:

1-There is a problem with this implementation in the case that more than 1 object's center lies in the same grid so only 1 object will be identified and the remaining will be neglected.

2-As the grid size gets smaller (instead of 3×3 we can have 19×19) the chance that more than 1 object center lies in the same grid cell decreases.

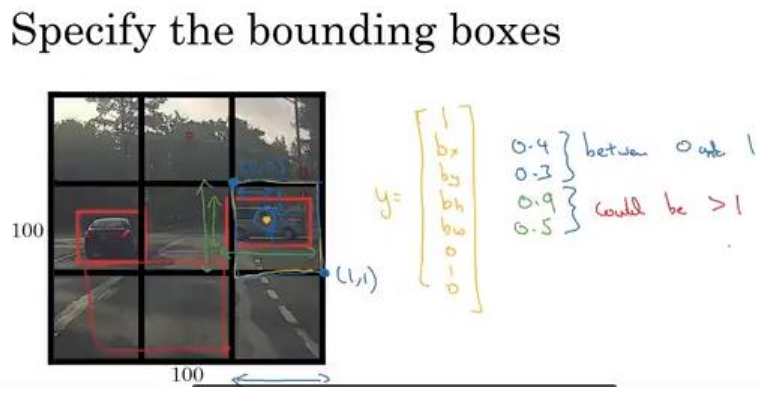
3- The process on one image is done at once which means that I'm not applying each grid cell to a convnet to apply classification and localization but the whole image is passed at once as we have done with the sliding window using convolutional layers.

How b_x, b_y, b_h, b_w are represented?

1-b_x and b_y are values between 0 and 1 that represents how far each of the co-ordinates from the upper left point relatively to the whole size of the grid (whole size is always equals 1). This means that if b_x=0.5 then the x co-ordinate of the center of the bounding box of the object in this grid cell is half the way from left side of grid cell right side of the grid cell and if b_y=0.7 then y co-ordinate of the center of the bounding box of the object in this grid cell is 7/10 the way from the upper side of the grid cell lower side of the grid cell

2- b_h and b_w can be any value larger than 0 so if b_h say equals 0.9 so this means that the height of the bounding box of the object in this grid cell is 0.9 the height of the grid cell (0.9×height of the grid cell) and similarly for the width.

So if b_w is larger than 1 (say 1.5) then this means that width of the bounding box is 1.5 the width of the grid cell (1.5× width of the grid cell).



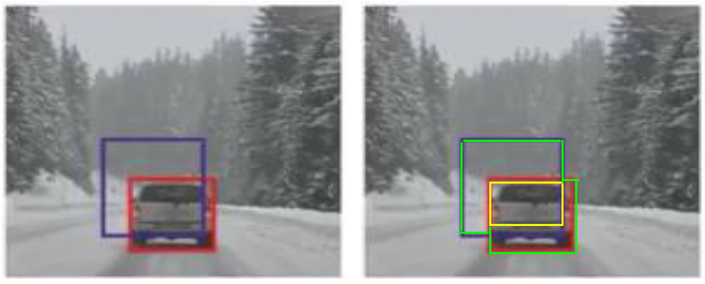
Note:

All what we have said in this section (Bounding box prediction) is introduced mainly in what is called by YOLO algorithm that will be discussed later after summing all the ideas introduced in it.

Intersection over union (IoU)

This is an evaluation metric for the accuracy of the predicted boundary box by comparing its overlapping with the actual box (aka ground truth box) during training, validation and testing phases.

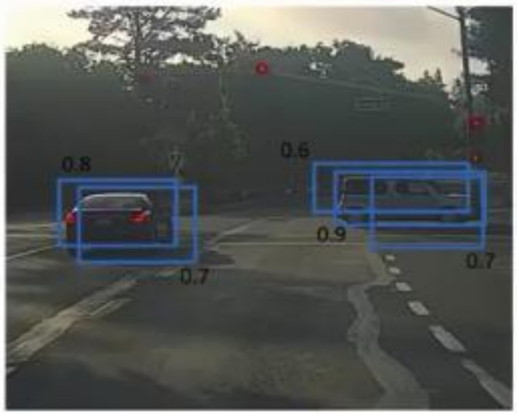
Assume having this image where the ground truth box is the red box and the predicted box from your network is the purple one, so how accurate is this predicted box?



$$\text{IoU} = \frac{\textit{Area of Intersection between boxes (yellow part)}}{\textit{Area of Union of boxes (green part)}}$$

So if IoU = 0 so there is no overlapping between the boxes and if IoU =1 so the predicted box is perfectly matching with the ground truth (actual) box. We use IoU to tell how much we would penalize our algorithm by comparing to a threshold (say 0.5) so if IoU is greater than the threshold then this is a good acceptable predicted boundary box and otherwise (<threshold) then we should penalize our algorithm for this bad prediction.

Non-Max suppression



Non-max suppression is used for finding only one box for each objects as we previously have said that we could use different sliding window to find the object so these sliding windows in fact would detect the same object within several sliding windows or more generally with any other technique to be used for detection there could be several boxes detecting the same object as shown in the figure ➔ so how could we make our algorithm only choose the best box for each object and reject the others? Noting that the values on the boxes is the highest probability among the classes of the network so if a network has 4 classes (cats, dogs, persons & cars) so each class has its own probability as said before in the softmax regression discussion and the highest probability (called P_c) is the predicted class by the network to this input test image so we take the highest probability and wrote it along with the boundary box. Let us get back to non-max suppression algorithm:

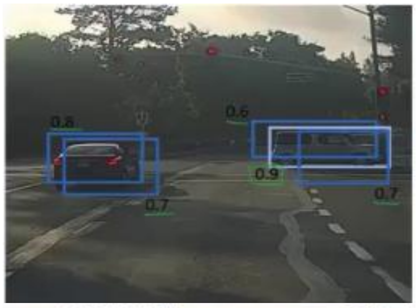
Algorithm

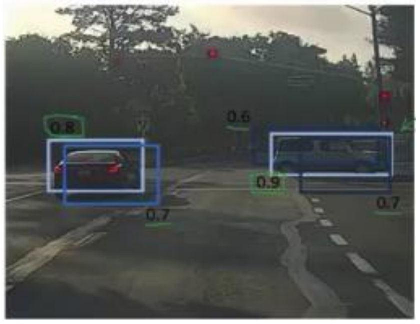
1-Discard all boxes associated with low probabilities (P_c < threshold).

2-If there are remaining boxes (boxes with $P_c > 0.6$) so loop over the following:

- Take the box with highest probability(P_c) among all the remaining boxes and keep it.
- Perform IoU calculation between this kept box and all the remaining boxes and discard any box of the remaining ones with IoU > threshold (say 0.5) because this means that most probably these boxes are detecting the same object of the kept box.
- Repeat the 2 previous steps again on the non-discarded boxes (IoU<0.5) because these boxes are most probably belonging to another object in the image.

Example

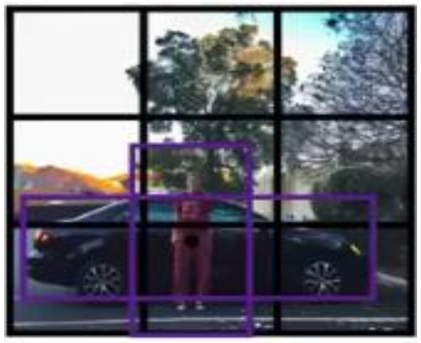




Now we will work on the example at the top of the page so firstly we will pick the highest probability among them all (0.9) and keep it (in white), then now we apply IoU with all the remaining 4 boxes so we found out that 2 of the remaining boxes(0.6 and 0.7 boxes) are having high IoU with this box in white so we will discard them. Now we have remaining 2 non-discarded boxes (on the left) so we will take the highest among these 2 boxes (0.8 box in white) then perform IoU with the remaining 1 box (0.7 box on the left) so we found out that it has high IoU so we will discard it. Eventually we have no remaining boxes so final output is the 2 white boxes with 0.9 and 0.8 class probability.

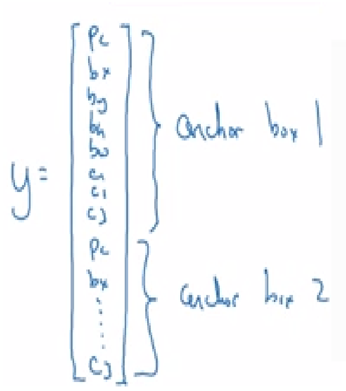
Anchor boxes

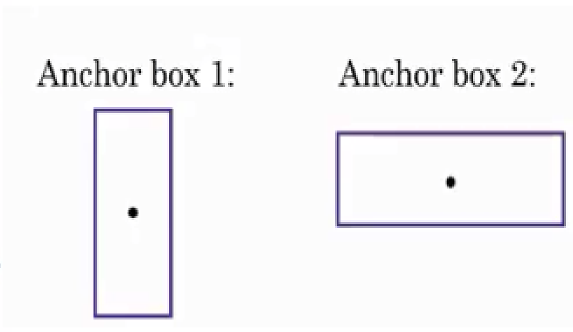
We have addressed a problem before that we skipped it which is that each grid cell in the image can only has 1 object so if there is more than 1 object so the remaining will be neglected and here comes the idea of anchor boxes to be able to get all the objects that their centers lie in the same grid cells.



Assuming having an image like this shown and there are 2 objects (a person and a car) where both the centers of the 2 objects are in the same grid cell. Anchor boxes are introduced so that each of them is of a different aspect ratio so if we assume we have 2 anchor boxes (could be more as you like) so y now is not only

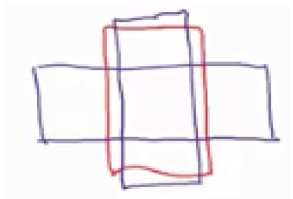
this ➔ $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ \vdots \\ C_c \end{bmatrix}$ but will be doubled ➔





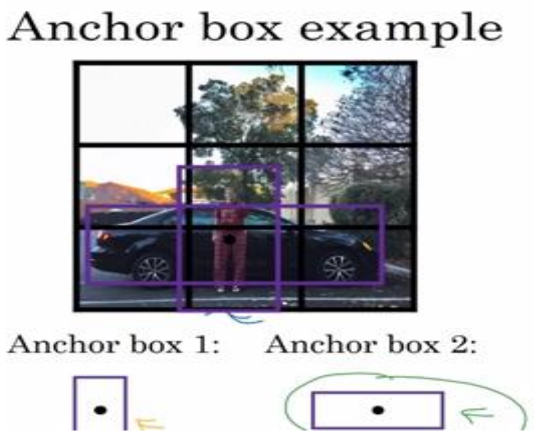
So now each part of the output is detecting objects that mostly can be represented by a box with the aspect ratio of those anchor boxes so for our example Anchor box 1 will mostly fit the person and anchor box 2 will mostly fit the car noting that in each grid cell we can't have more objects' centers than the number of anchor boxes (If no. of anchor boxes=2 therefore the maximum number of objects center to be detected within one grid cells is 2) so that is why we are using small grid size (19×19) with high number of anchor boxes (around 5) to be sure that almost no probability of missing any objects in the image.

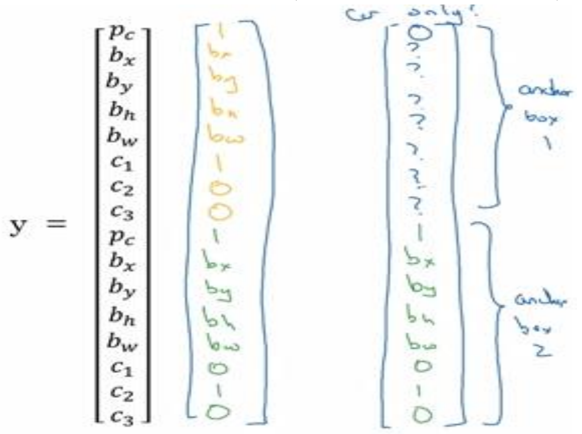
Note: we determine the best fit anchor box based on the highest IoU between the anchor boxes (purple) and the ground truth box (red) ➔



Output size now (assuming grid cells 3×3) is [3×3× no. of anchor boxes × (5 + no. of classes)]

Our Example:

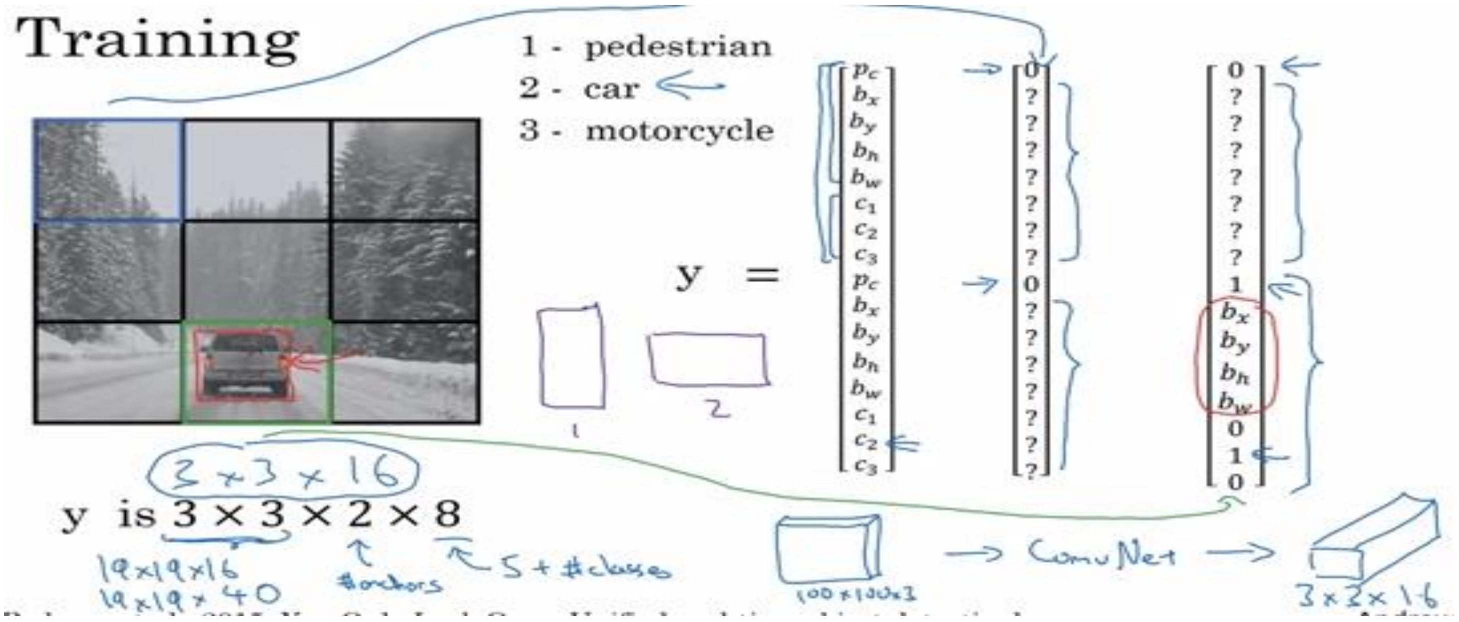




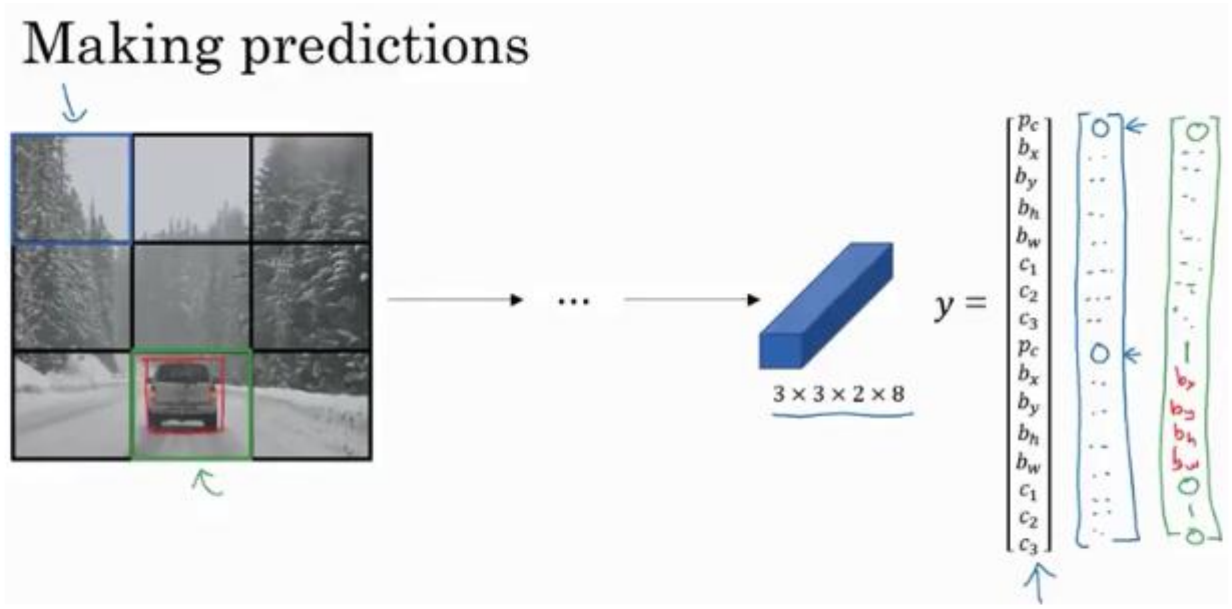
You Only Look Once (YOLO) Algorithm

In fact most of the topics discussed after the sliding window topic are introduced in the YOLO algorithm paper so this part we will gather all what we said in a single algorithm called YOLO.

For training:

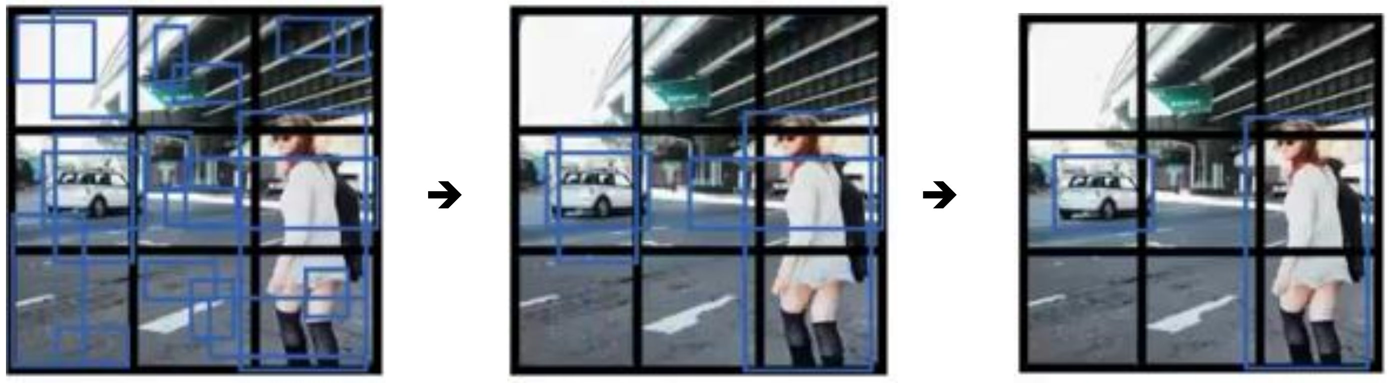


In training we have images labeled as shown as shown assuming having 3×3 grid cells, 2 anchor boxes and 3 classes so outputs are 9 vectors each of length 16 (2 anchor boxes × (5+3)) such that for the given training example there will be 8 vectors of just 0's and don't cares (?) while there will be 1 vector with 0 for the first anchor box and 1 for the second as the ground truth box is wider than being taller so IoU with anchor box 2 is greater.
You will notice in the bottom right of the image that we can use 19×19 grid cells and also we can use 5 anchor boxes (5 anchor boxes × (5+3) = 40).
After this image will pass through the network we will have output of (3×3×2×8) as shown



Don't forget that in the test phase we will do non-max suppression to discard boxes pointing at same object as well as discarding boxes with low probability by take into your consideration that all grid cells will provide boxes (their number is similar to number of anchor boxes) and non-max suppression is the technique that will take care of discarding all these boxes especially that most of them will have low probabilities and some will point to same object.

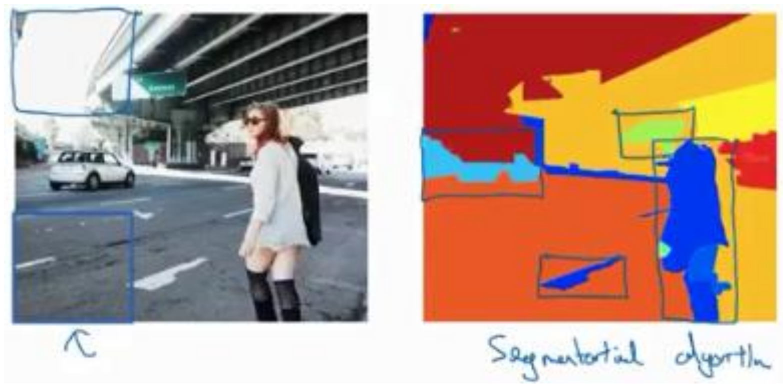
<u>Example:</u> Assuming we have 2 anchor boxes
First image with all boxes (2 boxes for each grid cells).
Second image is after discarding boxes with low probability using non-max suppression.
Third image is after discarding boxes pointing to the same object using non-max suppression.



<u>Note:</u>
YOLO algorithm is usually used for real-time application as it is a fast run-time algorithm compared to other algorithms.

Region Proposals

This idea is the main basis of some networks as R-CNN, Fast R-CNN, Faster R-CNN,…
As we said before that sliding windows is a little bit bad and time consuming as we need to slide the window over the whole image so most probably a lot of windows will be of background that doesn't contain any objects and this is a waste of times and resources. What R-CNN did is applying region proposal where we also having windows but more reasonable windows and to determine these reasonable windows, a segmentation algorithm is used (right photo) ➔
As we see here that segmentation gives some blobs with different colors so the windows generated are detecting those blobs to check if they contain object or not and the number of windows here of course is much less than that will be generated by normal sliding window algorithm.



R-CNN was too slow so fast and faster R-CNN was introduced.

Course 4 Week 4

This week will be concerned with 2 real-world applications which are face recognition and neural style transfer.

Face verification VS. Face recognition

Verification

- Input ➔ Image + Name or ID
- Output➔ Whether the image belongs to the claimed person or not.

It is one to one (1:1) problem which means we compare 1 image to 1 ID or name.

Recognition

We have a database of K persons

- Input ➔ Image
- Output ➔ ID if the image is any of the K persons

It is one to all (1:K) problem which means we compare 1 image to all K data in the database.

Face recognition system is harder than face verification because if you have a face verification system of 99%, it will be acceptable but if we use this system as a building block of the face recognition system having K=100 so this means that as we 1 % error for each person and we have K=100 then we may get wrong 100 times which is a high number so this means that we need the face verification method to be higher than 99% (may be 99.9% accuracy)

One-shot learning

One of the challenges that faces face recognition system is that we need to solve the one-shot learning problem which means that for most face recognition application you need to recognize the person given just 1 single image or in other words given 1 training example of that person's face which historically is almost an impossible task for deep learning to solve.

Example

Assume we have a database of 4 persons (K=4) as shown where each person has a photo of his face and now we entered 2 persons and want to check if those 2 persons are in the database or some intruders so we find that one of them is actually in the database and the other is not so here the case is that we are learning from one example to recognize a person again so learning process will be bad due to the small dataset.

Note: If we supposed that we will train this with convnet, softmax will be 5 not 4 because none of the above choice will be added.

This problem is solved using learning "Similarity" function.

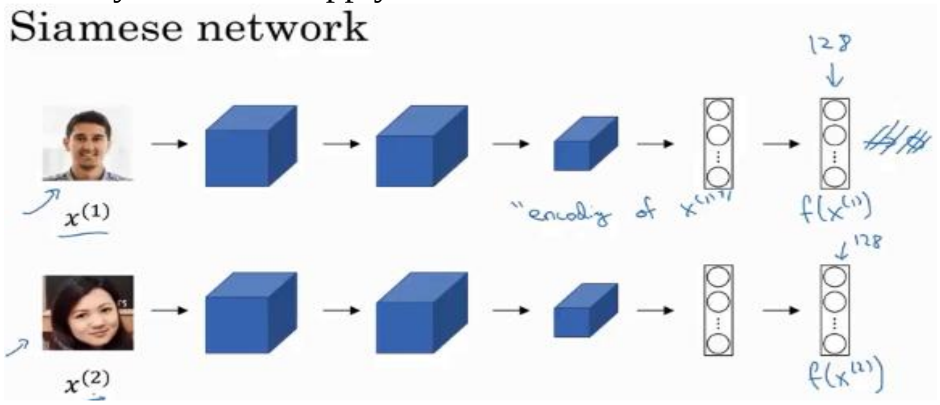
Learning "similarity" function
d(img1, img2)= degree of difference between images
so if img1 and img2 are of the same person then you need d(img1, img2) to be low (↓)
and if img1 and img2 are of different persons then you need d(img1, img2) to be high (↑)
then if d(img1, img2) < threshold (τ) ➔ same person
then if d(img1, img2) > threshold (τ) ➔ different person
Then in face recognition we just apply similarity function between the new photo and each of the photos in the database.

So here comes the question, how to apply this function ? using Siamese network

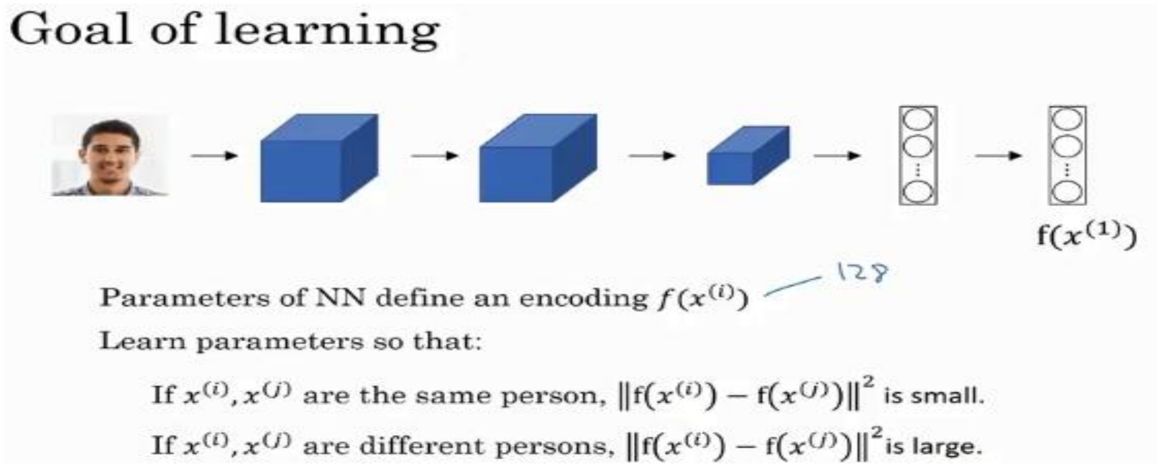
Siamese network

Usually we input image ($x^{(1)}$) to a network composed of convolutional, max pooling and fully connected layers and then passes the output from the last FC layer to softmax so we will do the same sequence except that we will remove the softmax layer (now the FC layer is the last layer). Assume that the last FC layer is composed of 128 nodes and we gonna call it f($x^{(1)}$) and we can think of this as an encoding of $x^{(1)}$. so now we have the image $x^{(1)}$ is represented by vector 128 values. So now if you want to build a face recognition system so you want to compare two images ($x^{(1)}$ & $x^{(2)}$) so what you do is to apply $x^{(2)}$ to the same network with the same parameters that $x^{(1)}$ was trained on.

So now as $x^{(2)}$ is a different person as shown in the figure ➔ so the resultant vector will be of different 128 values (call it f($x^{(2)}$)) that encodes the $x^{(2)}$.



Now we have two vectors each of length 128 which are f($x^{(1)}$) & f($x^{(2)}$) and to apply similarity function we do the following ➔ d($x^{(1)}$, $x^{(2)}$) = $\left\| f(x^{(1)}) - f(x^{(2)}) \right\|^2$



Goal of learning

Parameters of NN define an encoding $f(x^{(i)})$ — 128
Learn parameters so that:
If $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is small.
If $x^{(i)}, x^{(j)}$ are different persons, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is large.

Triplet loss

To get good parameters for the network to be able to get a good encoding of the image is to define and apply gradient descent on what is called by triplet loss so we need to define our learning objective of this system:

Our learning objective is to make the distance between the anchor image (original image) and the positive image to be small and the distance between the anchor image and the negative image to be as high as possible and from this learning objective the idea of triplet loss arises as we always look at the 3 images (anchor (A), positive (P) & negative (N)) at a time.

How to formalize the learning objective?

we want : $\underbrace{\|f(A)-f(P)\|^2}_{d(A,\,P)} \le \underbrace{\|f(A)-f(N)\|^2}_{d(A,\,N)}$ ➔ noting that d() refers to the distance.

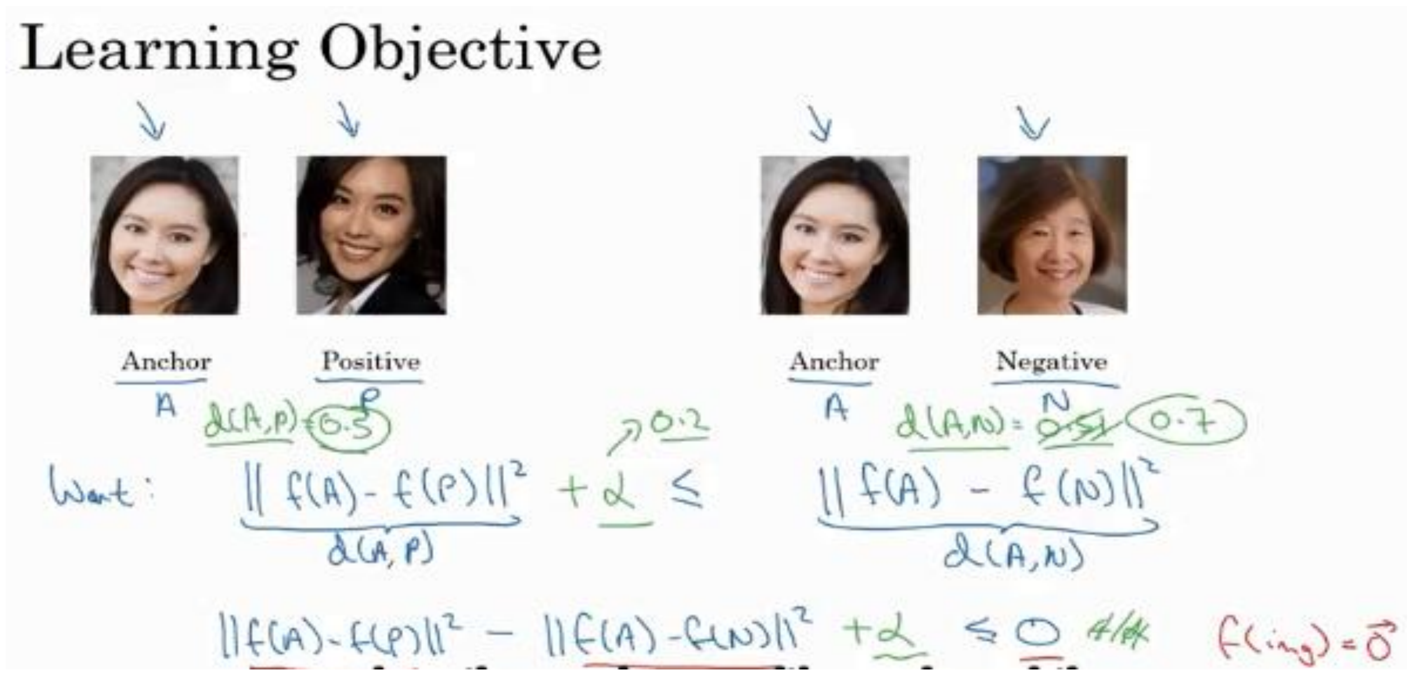
then: $\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 \le 0$

but there will be 2 trivial solution for this inequality:

- If network always gives encoding equals 0 (f(A)=f(P)=f(N)=0) so the network always gives 0 output.
- If the network gives equal encoding to all images (f(A)=f(P)=f(N)) so the network output will be always 0.

To avoid these 2 cases we will add a small positive term (+α) to L.H.S. Now α is a hyper-parameter to the network that prevents trivial solutions. α is called margin.

Finally ➔ $\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 + \alpha \le 0$



Triplet loss function

Given 3 images (A, P & N)

ℓ(A, P, N)= max (0 , $\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha$)

cost function J = $\sum_{i=1}^{m} \ell(A^{(i)}, P^{(i)}, N^{(i)})$

How to use the cost function?

Assume having 10,000 pictures of 1,000 persons so you have to start drawing(يسحب) from these 10,000 images sets of triplets images where 2 of them (A, P) of same person and 1 (N) for different and do gradient descent and repeat this process on different sets till you reach good parameters to encode every person's picture of the 1000 persons.

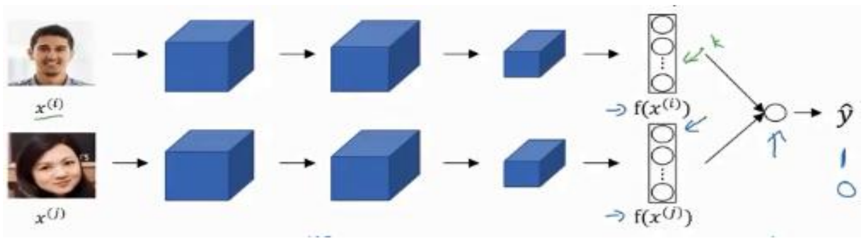
Note: You can notice here that every person is only having 10 images so we can train network with small dataset using triplet loss function and then use the final network for one-shot learning for your facial recognition system. ☺

How to choose your triplets ?

Don't choose A, P, N randomly because this would lead to an easy satisfaction of the learning objective and the gradient descent won't learn that much. You should choose triplets that are so hard to train on to make the gradient descent be able to learn and push parameters to make the two distances (d(A, P) , d(A, N)) as far as possible. Andrew NG hasn't explained how could we choose those hard trainable triplets from the dataset but said if you are keen on that you could check the paper of ➔ FaceNet: A unified embedding for face recognition and clustering.

Modification on Siamese network

Triplet loss is a good approach but not the only one to build a face recognition system. Siamese network could be modified to make learning similarity applied as a binary classification task by just adding a logistic regression that has the inputs from the outputs of the 2 Siamese networks subtracted from each other such that the output will be 1 if the 2 networks having the same person as inputs and 0 otherwise.



Note: Input here is a pair of images to learn from that could be of same of different persons.

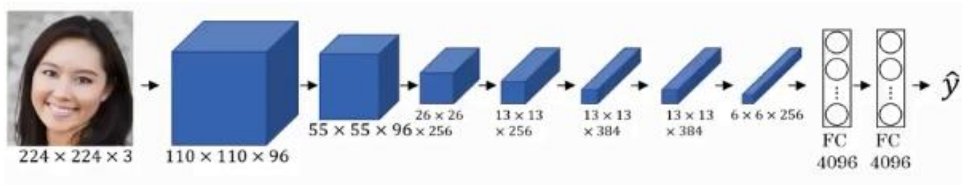
$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \left|f(x^{(i)})_k - f(x^{(j)})_k\right| + b\right)$$

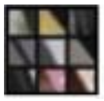
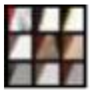
Another formula: $\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k} + b\right)$

Note: Both Siamese networks are having the same parameters.

What are deep convnets layers really learning?

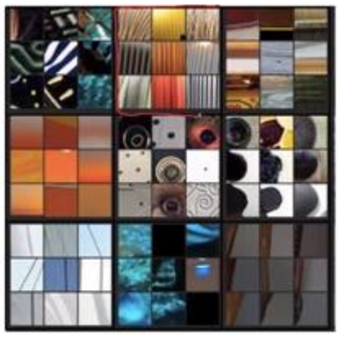
Assume we will go deeper in the layers of Alex convnet (↓) to see how layers and neurons see and deal with the input image.



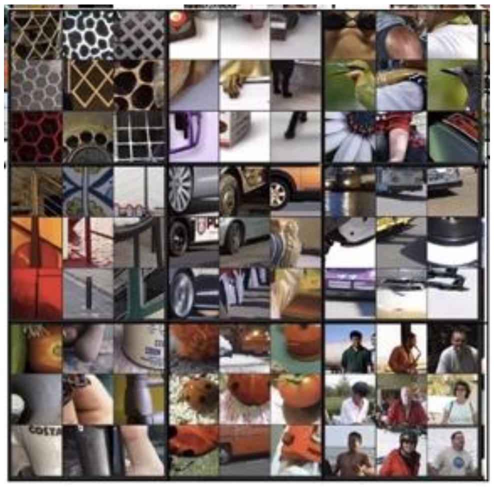
Each hidden unit is dealing with patches of the input image that maximizes its activation so assume each hidden unit is dealing with 9 patches of the input image. Now let us pick one hidden unit from layer 1 so what we have seen that those 9 patches  are the patches that maximizes this unit's activation which means that this neuron is dealing with those edges inclined to the right and when we picked another unit we found that those 9 patches  are the patches that maximizes the activation so this means that this neuron dealing with those edges inclined to the left and we started repeating picking different neurons so we found that each neuron is responsible of picking a low feature to cope with it as left side to be green or patches to be orange colored or patches to be green and so on …

So now we understand that hidden units in layer 1 are concerning low-level features as edges with different inclination, shades, colors and so on…

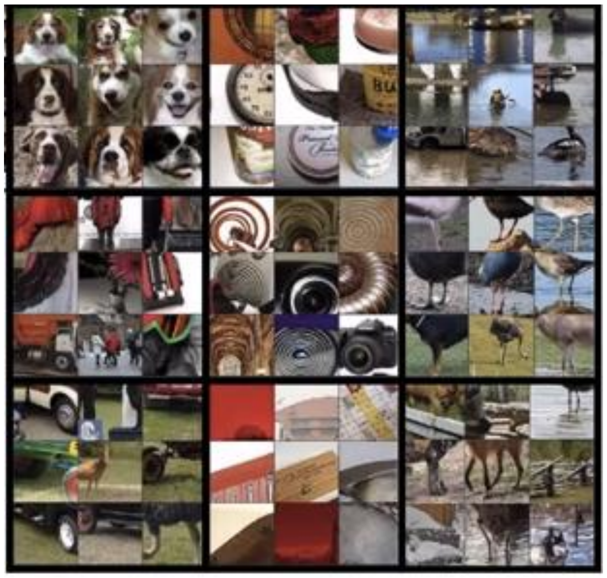
When we went further through the network for deeper layers we found out that hidden units now are dealing with larger patches sizes that could formulate higher level features as curvatures, geometrical shapes and more complex shapes as we go deeper.



Layer 2



Layer 3



Layer 4

<span style="color:red">Note:</span> As we go deeper, patches are bigger and concerned shapes are more complex.