


**Deep Learning for Computer Vision**  
**Professor. Vineeth N Balasubramanian**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Lecture No. 44**  
**CNNs for Object Detection-I**

We have so far seen neural networks in its most simplest form, feed forward neural networks, learnt back propagation, seen how different variants of gradient descent can be used. Learnt a few practical tricks while training neural networks, we then moved on to convolutional neural networks, the building blocks of computer vision. We saw how they can be trained using back propagation, we saw various kinds of architectures in CNNs.


And then we also saw last week on how one can visualize and understand what CNNs are learning as well as explain their predictions. But in all these lectures so far, we focused on the task being classification, given an image our objective was to classify what object was in the image. You could extend that to regression where you predict a real value by changing the cross entropy loss to a mean square error.

However, there are many other kinds of vision tasks that are possible and we will now move on to those tasks in this week. We will start with one of the most used tasks today which is object detection.

(Refer Slide Time: 1:38)



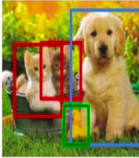



NPTEL




Indian Institute of Technology Hyderabad

Classification vs Localization vs Detection

Classification	Classification + Localization	Object Detection	Instance Segmentation
			
CAT	CAT	CAT, DOG, DUCK	CAT, DOG, DUCK
Single object		Multiple objects	

Credit: Mike Tamir, SIG & UC Berkeley





Vineeth N B (IIT-H)
§7.1 CNNs for Detection
2 / 44

So the difference between classification, localization and detection is as follows. In classification, given a cat in an image, the objective is to say that there is a cat that

corresponds to the image. In localization, in addition to saying that there is a cat you also have a bounding box around the cat to say where in the image the cat was located. In object detection, you go one step further and you detect all possible occurrences of the objects in your set of classes in a given image as well as localize them. In the localization task, there is only one object which you localize whereas in object detection there could be multiple objects, multiple instances of the same object, you could have a cat and a dog or two dogs and one cat all of these possibilities.


The objective here is to recognize each of these objects as well as localize them using these kinds of bounding boxes. Finally, the task of segmentation where the job is to label each pixel as belonging to one of say  $c$  classes depending on the number of classes we have. So, we will start this lecture with detection and then move on to segmentation as well as other tasks later this week.

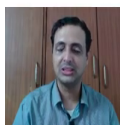
(Refer Slide Time: 3:17)

### Detection in the Pre-Deep Learning Era: Viola-Jones Algorithm<sup>1</sup>

- A framework to perform object detection in real-time
- Used predominantly for detecting frontal upright faces.
- Consists of three main components:
  - Weak classification using **Haar-like features** and **Integral Images**
  - Adaboost to build strong classifier
  - Classifier cascades





<sup>1</sup>Viola, Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, CVPR 2001



Vineeth N B. (IIT-H) [§7.1 CNNs for Detection](#) 3 / 44

Before deep learning came into the picture, there were other methods that were used to deploy computer vision for object detection on various devices. So, object detection and face detection were even available in say point and shoot cameras 15 years ago when deep learning methods had still not matured. Most of those methods used a variant of what is known as the Viola-Jones Algorithm, this was developed by Viola and Jones and that is the reason for the name and this work was published in CVPR of 2001.

And this is a framework to perform object detection in real time. It was primarily used for detecting frontal upright faces although it was adapted to be able to detect other kinds of objects or even detect parts of objects such as say eye detection on a face. The main contributions of this work was they performed weak classification using what are known as Haar-like features. We will see what these are in the next slide and they also introduced a very powerful idea known as integral images which helps make computations of such haar-like features significantly faster as we will see.



Then they used Adaboost as the classifier of choice but how they used Adaboost was slightly different from the traditional formulation in machine learning and they also used this as a cascade of classifiers as we will see over the next few slides.





(Refer Slide Time: 5:07)







### Haar-like Features

- Rectangular features based on Haar wavelets
- Feature value given by sum of pixels in white rectangles subtracted from sum in black rectangles, i.e.
 
$$f = \Sigma(\text{pixels inside black rectangle}) - \Sigma(\text{pixels inside white rectangle})$$
- To normalize for images of all scales, features are scaled through height and width



(a) Edge Features





(b) Line Features



(c) Four-rectangle features



Vineth N B (IIT-H)
§7.1 CNNs for Detection
4 / 44

Haar features are rectangular features based on haar wavelets, so these are, this is how the haar features look like. So, you can see here that there are generally a region of black and a region of white but there could be different combinations of those you could have just two regions white and black horizontally aligned vertically arranged or you could have a black sandwiched between a white vertically or horizontally or you could have checkerboard patterns and you can keep going forward you will get various kinds of patterns as you use the same idea.

So, the idea behind the haar filter is, the feature value is given by the sum of the pixels in the white rectangles subtracted from sum of the pixels in black rectangles. Remember this is

convolution, so things would get flipped which is why you have the output of convolving with a Haar filter would be the sum of the pixels inside the black rectangle minus the sum of the pixels inside the white rectangle, remember black is 0 white is 1 but because of the flipping this is what the output would turn out to be.


And to normalize for images of different scales, these features could be scaled through height, through width or even different kinds of patterns as we will see. So, you could have a  $6 \times 4$  filter, a  $10 \times 6$  filter. You could have a  $10 \times 24$  filter so on and so forth. You could have them of various sizes depending on what you are applying these haar wavelets for.

(Refer Slide Time: 6:52)

The slide, titled "Haar Features: Intuition", illustrates the concept of Haar wavelets. It features the NPTEL logo and the IIT Bombay logo. The main content shows a grid of images: a grayscale face on the left, and a 2x2 grid of images on the right. The top row of the grid shows two Haar filters: a white horizontal bar on a black background (representing a top edge detector) and a black vertical bar on a white background (representing a left edge detector). The bottom row shows the same face image with these filters applied, highlighting the detected edges in black and white. At the bottom of the slide, there is a small video inset of the presenter, Vineeth N B (IIT-H), and a footer indicating the slide is 5 out of 44 in the section "§7.1 CNNs for Detection".

So, the intuition of using haar features for object detection or face detection in particular is that a lot of the features of the human face are about contrast between a certain region and the adjacent region. So, for example, if you looked at the eye region as you can also see here, it is likely to have a black area below and a reasonably brighter intensity area right above that. So, Haar features such as this would capture that difference. You could also look at eyes the other way and represent them and capture eyes using a white block sandwiched by two black blocks in a Haar wavelet. So, these are likely to be good feature detectors of parts of the face.

(Refer Slide Time: 7:44)



### Integral Image

- Reduces computational complexity caused while adding pixel intensities for any image operation
- For image  $i$ , Integral image  $ii$  is given by:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

- Recurrent definition of  $ii(x, y)$ :
 
$$s(x, y) = s(x, y - 1) + i(x, y)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y)$$
 where  $s(x, y)$  is cumulative row sum  
 $s(x, -1) = 0, ii(-1, y) = 0$

38	66	17	57	73	2
89	97	98	30	2	89
15	37	75	81	23	50
96	70	63	84	41	29
93	84	89	46	28	26
74	74	51	96	64	39

Image

38	104	121	178	251	253
127	290	405	492	567	658
142	342	700	1013	1152	939
238	506	761	1013	1152	1322
331	685	1024	1325	1492	1688
405	833	1226	1620	1851	2086

Integral Image

As I just mentioned, this work also introduced the idea of what is known as an integral image this is a fundamental concept not necessarily anything to do with face detection, it is a way of making computations faster especially when you use haar-like features but this was introduced in this particular work by Viola and Jones for doing object detection.

So, let us see what an integral image is? So, the idea is to reduce computational complexity when you are adding pixel intensities for whatever option and the integral image is defined as, for any pixel in the integral image you sum up all the pixel values up till that point starting from the origin on the top left and the summation of all the pixel intensities is what is filled at a particular location.

So mathematically, you could write it as, integral image at x,y location would be the

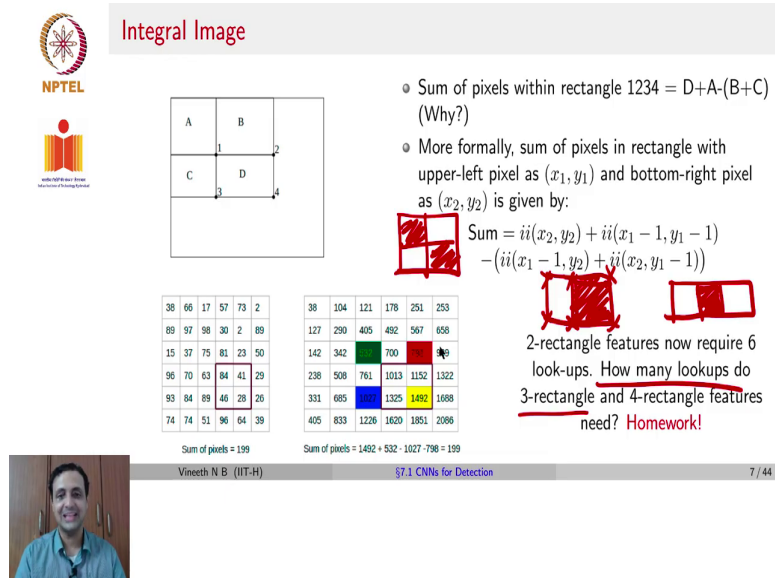
$\sum_{x' \leq x, y' \leq y} i(x', y')$  where x is the current x location, y being the current y location. So, you add

up all the pixel values until that particular pixel that you are looking at and the sum of all of those intensities is what you fill in at that pixel in the integral image. So, given this kind of an image, the integral image would look like this. You can see that for example the  $2 \times 2$  window 290 is the value given and 290 is given by  $38 + 66 + 89 + 97$  and so on and so forth.

So, the recurrent definition for an integral image would be given by  $s(x, y) = s(x, y - 1) + i(x, y)$  the above column plus current pixel location and the integral

image at  $x, y$  would be  $s(x, y) + ii(x - 1, y)$ . You can work this out and see that this is correct. And  $s(x, y)$  is the cumulative row sum and to make this recurrent definition work you have to define  $s(x, - 1) = 0$  and  $ii(- 1, y) = 0$ .

(Refer Slide Time: 10:11)



**Integral Image**

- Sum of pixels within rectangle 1234 =  $D+A-(B+C)$  (Why?)
- More formally, sum of pixels in rectangle with upper-left pixel as  $(x_1, y_1)$  and bottom-right pixel as  $(x_2, y_2)$  is given by:
 
$$\text{Sum} = ii(x_2, y_2) + ii(x_1 - 1, y_1 - 1) - (ii(x_1 - 1, y_2) + ii(x_2, y_1 - 1))$$

2-rectangle features now require 6 look-ups. How many lookups do 3-rectangle and 4-rectangle features need? Homework!

A	B	1	2
C	D	3	4

38	66	17	57	73	2
89	97	98	30	2	89
15	37	75	81	23	50
96	70	63	84	41	29
93	84	89	46	28	26
74	74	51	96	64	39

38	104	121	178	251	253
127	290	405	492	567	658
142	342	761	1013	1152	1322
238	508	761	1325	1492	1688
331	685	1226	1620	1851	2086
405	833	1226	1620	1851	2086

Sum of pixels = 199      Sum of pixels =  $1492 + 532 - 1027 - 798 = 199$

Vineth N B (IIT-H)      §7.1 CNNs for Detection      7 / 44

Let us actually now see how the integral image can be used in particular for haar filters. Let us assume now that we want to find the sum of the pixel intensities inside this rectangle defined by 1, 2, 3, 4 A is this top left rectangle B is this entire rectangle including a and including a part of it is A, C is this entire rectangle and D is this full rectangle. So, the sum of the pixels within the rectangle 1, 2, 3, 4 would be given by  $D + A - (B + C)$ . You can work this out and see that it would be quite true.

So, more formally you can say that sum of pixels inside a rectangle with an upper left pixel as  $x_1, y_1$  and bottom right pixel as  $x_2, y_2$  could be given by the  $ii(x_2, y_2) + ii(x_1 - 1, y_1 - 1) - (ii(x_1 - 1, y_2) + ii(x_2, y_1 - 1))$ , so you can work this out to see that this is actually true.


So, you would notice now that in this particular case, let us say you wanted to find out the sum of pixel intensities in this particular square block, so then what you could do is you could you already have the integral image computed, so you only have to look up a certain value certain set of values here and say  $1492 + 532 - 798 - 1827$ . Why is this useful?

When you want to use a convolution such as a Haar filter for these kinds of images, if you had a 2 rectangle feature, what we mean by a 2 rectangle feature is let us assume that you have a Haar filter which is given by just a black region followed by a white region. Remember we said that this is the sum of pixel intensities in the black region minus the sum of pixel intensities in the white region, such a convolution for filter would only need 6 lookups in your integral image, why so?

You would need one lookup for all of these corners, so there are six corners here including the middle ones you will need those 6 lookups and you will be able to compute what would be the output of the haar filter without actually doing any kind of a convolution, you can just look up the integral image and be able to get these values. As an exercise, try to think how many lookups you would need for a 3 rectangle feature. What do we mean by a 3 rectangle feature. Consider a haar filter with say black in the center and white on both sides.

So, this would be a 3 rectangle feature, how many lookups would this need? It is simple but work it out later. Similarly, what if you had a 4 rectangle feature something like a checkerboard something like this. So, then how many lookups do you need? Think about it as homework.

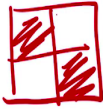
(Refer Slide Time: 13:32)



### Weak Classifiers


- Each feature is considered as a weak classifier
- Given feature  $f_j$ , a threshold  $\theta_j$  and a parity  $p_j$  indicating the direction of the inequality sign, classifier can be defined as:
 

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$
- Each classifier weak on its own, but combination gives strong classifiers



Do you see any problem here?

For a sliding window base detector of size  $24 \times 24$ , there exist over 160k features. How to identify the best ones?



Vineeth N B (IIT-H)
§7.1 CNFs for Detection
8 / 44

So, once you have the outputs of these Haar filters, the algorithm of Viola and Jones suggested that each feature can be now considered as a weak classifier. Given a feature  $f_j$  which is one of those Haar wavelets could be a 2 rectangle feature, a 3 rectangle feature, 4


rectangular feature one of those you can have all possible variants. A threshold  $\theta_j$  and a parity  $p_j$  which only indicates the sign. The classifier is defined as, the weak classifier is defined as  $h_j(x) = 1$  if  $p_j f_j(x) < p_j \theta_j$  and 0 otherwise.

So, you are saying that if that feature value is less than a threshold, you are going to call it 1, 0 otherwise, of course you could define the parity to use this otherwise if you like. So now, using this you are going to get one classifier for every Haar feature. We know that a combination of weak classifiers can give strong classifiers but do you see a problem with this approach so far?

If you thought carefully, even if you had a simple sliding window of  $24 \times 24$  and let us say that is the window you are going to slide across the image and then inside each of those sliding windows you are going to compute various Haar features and use this thresholding to be able to detect whether Haar feature says there is a face or not a face. But the problem is if you had even just a  $24 \times 24$  window there are over 160,000 features possible. By that we mean, so within a  $24 \times 24$  window you could have a very large checkerboard then you could imagine all possible combinations of arranging black blocks and white blocks adjacent to each other inside a  $24 \times 24$  window, you will have a huge number of features.

Now to be able to compute 160,000 weak classifiers inside each sliding window seems tedious, what do we do?

(Refer Slide Time: 15:51)



### Classifier Training using AdaBoost

Recall: AdaBoost learns a strong classifier as a linear combination of weak classifiers


- Given example images  $(x_1, y_1), \dots, (x_n, y_n)$  where  $y_i = 0, 1$  for negative and positive examples respectively.
- Initialize weights  $w_{1,j} = \frac{1}{m} \frac{1}{l}$  for  $y_i = 0, 1$  respectively, where  $m$  and  $l$  are the number of negatives and positives respectively.
- For  $t = 1, \dots, T$ :
  1. Normalize the weights,
 
$$w_{t,j} \leftarrow \frac{w_{t,j}}{\sum_{j=1}^l w_{t,j}}$$
 so that  $w_t$  is a probability distribution.
  2. For each feature,  $j$ , train a classifier  $h_j$  which is restricted to using a single feature. The error is evaluated with respect to  $w_t$ ,  $\epsilon_j = \sum_i w_t |h_j(x_i) - y_i|$ .
- 3. Choose the classifier,  $h_t$ , with the lowest error  $\epsilon_t$ .
- 4. Update the weights:
 
$$w_{t+1,i} = w_{t,i} \beta_t^{1-\epsilon_i}$$
 where  $\epsilon_i = 0$  if example  $x_i$  is classified correctly,  $\epsilon_i = 1$  otherwise, and  $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ .

• The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where  $\alpha_t = \log \frac{1}{\beta_t}$

*Viola, Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, CVPR 2001*



Vineeth N B (IIT-H)
§7.1 CNNs for Detection
9 / 44



That is where AdaBoost comes into the picture. So, the way Adaboost was used in the Viola Jones algorithm is, the algorithm given here this is from the paper, we will only walk over the salient parts of this, this is the standard Adaboost algorithm where given example images  $x_1, y_1$  till  $x_n, y_n$  you initially start with uniformly weighting all your training samples and you normalize those weights across all of your training samples.

Now, for each feature  $j$  which is one of your Haar features among the all possible combinations that you could have, you train a classifier  $j$  which is a  $h_j$  which is a weak classifier which is restricted to using a single feature and the error with respect to this weak classifier is given by  $\epsilon_j$  which is given by  $w_i \times |h_j(x_i) - y_i|$ , rather  $h_j(x_i)$  is the output of this weak classifier  $y_i$  was the expected output, so how close this weak classifier was to the expected output multiplied by the weight of that sample that we are evaluating, that is the standard Adaboost approach to weight more erroneous samples a little higher and less erroneous samples a little lower.

But now, the Viola Jones approach to using Adaboost says, among all those possible features which would have given you several different classifiers, choose the classifier  $h_t$  which with the lowest error  $\epsilon_t$  is among all your possible classifiers choose the one that gives you the least error. Based on the error of this classifier you re-weight all your samples as you see here, this is again the standard reweighting step in AdaBoost and you repeat this over and over again and your final strong classifier after a certain number of iterations, remember now you would not be using all features you are only using the  $T$  number of features where capital  $T$  is a parameter that you can give.

In each iteration you pick one of the features that is really strongest and at the end you only use a weighted combination of capital  $T$  different features. And once again this final strong classifier rule is the standard AdaBoost algorithm.

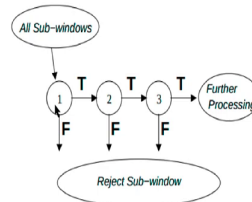
(Refer Slide Time: 18:19)



## Classifier Cascade

### Intuition:

- Very less number of sub-windows actually have objects (False positive rate very high)
- But computation is equal across all sub-windows
- Eliminate negative windows earlier thereby, reducing computation time spent on them



Vineeth N B (IIT-H)

§7.1 CNNs for Detection



10 / 44

A third component of the Viola Jones algorithm was known as a classifier cascade, so this came from the motivation that very few sub windows of an image actually have objects or faces. Remember you could have 100s of 1000s of sub windows in a given image but you are not going to have that many faces in a given image, so there it is very likely that the false positive rate may be very high if you evaluate every possible window. So, how do we overcome this issue?

So, what this classifier cascade idea suggests is eliminate negative windows earlier therefore reducing computation time spent on them, so if there is a sub window that is negative ignore it and never include any further sub windows inside that sub window for any further processing. Speaking at a very high level, so you have all possible sub windows that goes to the first stage and you rule out a certain number of them and assume that they will never have a face at all then only those sub windows where a face is likely it goes to a next stage in the classifier which again checks for faces again and if there are sub windows rejected at that stage, that again comes out. Only the remaining sub windows go through the next stage so on and so forth.

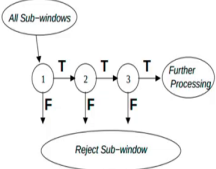
And in each of these stages, you have a set of weak classifiers. You can constrain that using the number in capital T, but now you can ensure that you can use a stage of different classifiers to keep refining your performance over time.

(Refer Slide Time: 20:05)

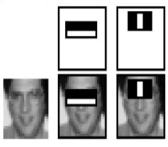


### Classifier Cascade: Why does it matter?


- A 1 MP has  $\sim 10^6$  pixels and a comparable number of candidate face locations  $\Rightarrow$  our ideal false positive (FP) rate has to be less than  $10^{-6}$



- This feature combination can yield 100% detection rate and 50% FP rate
- A 200-feature classifier can yield 95% detection rate and FP rate of 1 in 14084. Not enough!



- Detection rate and FP rate of cascade are found by multiplying respective rates of individual stages
- Detection rate of 0.9 and FP rate on the order of  $10^{-6}$  can be achieved by a 10-stage cascade if each stage has detection rate of 0.99 ( $0.99^{10} \approx 0.9$ ) and FP rate of about 0.30 ( $0.3^{10} \approx 6 \times 10^{-6}$ )



Vineeth N B (IIT-H) 37.1 CNNs for Detection 11 / 44

Why does this classifier cascade matter? We just mentioned that for a problem like phase detection or object detection, false positive rate is very important. So, if you had a one megapixel image, remember you are dealing with  $10^6$  pixels and at least a comparable number of candidate phase locations, even as if you assumed that a face could be at each of these locations you are at least looking at  $10^6$  probabilities. Once again, you could have different sized faces which would even further expand the possible candidate phase regions but for the moment, let us assume that you have at least  $10^6$  candidate locations.

So, your ideal false positive rate has to be less than  $10^{-6}$  assuming there is only 1 face or of the order of a few faces. So, you want your false positive rate to be about  $10^{-6}$ . If you took these basic Haar filters that we spoke about a few slides ago, this combination would yield 100 percent detection rate would give you a 50 percent false positive rate and it gives you a lot of possibilities of faces. Not all of them would actually turn out to be faces.

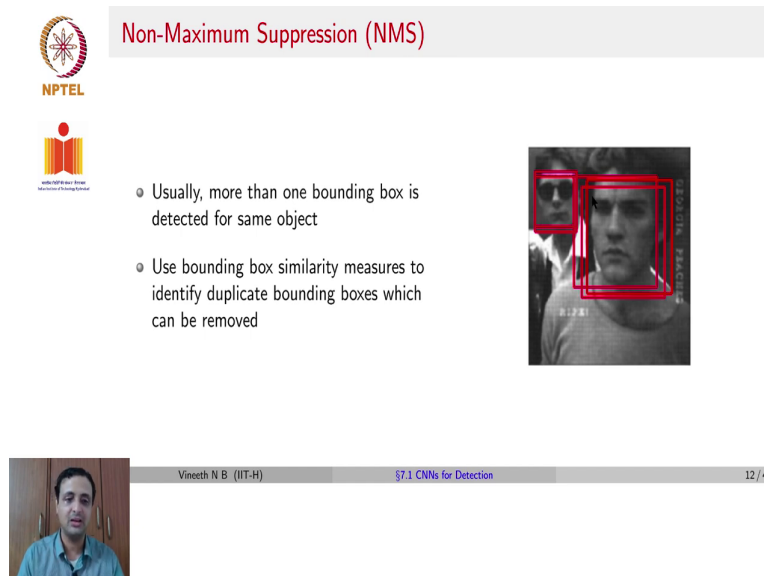
If you went a little further and included 200 features, remember the above example we had only two kinds of Haar filters, if you included 200 kinds of Haar filters, when we say 200 kinds you arrange those black rectangles and white rectangles in several ways you could have them as a checkerboard  $2 \times 2$  checker board, a  $5 \times 5$  checker board, you could have 3 rectangles you could have 5 rectangles 7 rectangles all those possibilities are open. You have 200 features that yields a 95 percent detection rate and a false positive rate of 1 in roughly

about 14, 000. It is better but it is still not enough to what we want which is  $10^{-6}$  false positive rate.

What do we do? The classifier cascade comes to our rescue. So remember, that when you have a cascade the final detection rate and the false positive rate are found by multiplying the false positive rate and reduction rate of each of these stages. So, if each of these stages was say a 200 feature classifier which means your capital T was 200, then you may have a detection rate of say approximately 0.9. You could then achieve a false positive rate of  $10^{-6}$  across a 10 stage cascade if each stage had a detection rate of 0.99 and a false positive rate of 0.3.

Because if you had a false positive rate of 0.3 in each stage of the cascade, you would then have  $0.3^{10}$  because you are talking about 10 stage cascade so  $0.3 \times 0.3 \times 0.3$  and so on, we will have  $0.3^{10}$  which is roughly about the order of  $10^{-6}$ . So the idea of classifier cascade allows us for each stage in the cascade to have a reasonable detection rate of 0.99 and a false positive rate of 0.3 which as we saw is possible with just a few features and we just keep refining these over stages of the cascade and obtain the false positive rate we ideally wanted.



(Refer Slide Time: 23:39)



The slide is titled "Non-Maximum Suppression (NMS)". It features the NPTEL logo and the IIT-H logo. The main content consists of two bullet points: "Usually, more than one bounding box is detected for same object" and "Use bounding box similarity measures to identify duplicate bounding boxes which can be removed". To the right of the text is a grayscale image of a person's face with several overlapping red bounding boxes. At the bottom left is a small video feed of a man, and at the bottom right is a footer with the text "Vineeth N B (IIT-H) 7.1 CNNs for Detection 12 / 44".

Another aspect of this method, for that matter any detection method, is the notion of non-maximum suppression. So, it is likely that many windows around an object may be classified as containing an object. So, how do you choose which one of them is the right object?

(Refer Slide Time: 24:06)




### Intersection over Union (IoU)

**Intersection**      **Union**      **Intersection over Union**

$B_1$        $B_2$        $B_1$        $B_2$

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2}$$



*Credit: Hands-On Convolutional Neural Networks with TensorFlow, Iffat Zafar*



Vineeth N B (IIT-H)      §7.1 CNNs for Detection      13 / 44


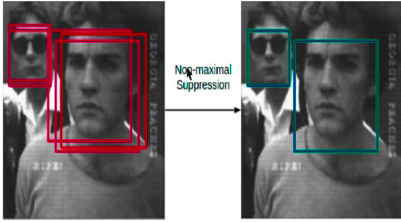
So, we have to use some kind of bounding box similarity measures and the most popular one is known as intersection over union. Intersection over union states that if you have two bounding boxes  $B_1$  and  $B_2$  you take their intersection and you take their union. The ratio of the intersection to the union gives you a sense of how close these bounding boxes are. If the ratio is 1 which means the bounding boxes  $B_1$  and  $B_2$  are exactly the same or as they get close to 1 you are going to have that these bounding boxes are fairly overlapping with each other. How do we use this IoU?

(Refer Slide Time: 24:45)



### Non-Maximum Suppression (NMS)

- Select a random box from a bounding box proposal list
- Compare it with rest of the boxes; if  $IoU > 0.5$ , remove box from list
- Repeat until boxes left are exclusive



Vineeth N B (IIT-H)      §7.1 CNNs for Detection      14 / 44

We can now use it for non-maximum suppression. So, if you had a set of possible bounding boxes which contain an object you can select a random box from them. Then you compare that box with the rest of the boxes. If IoU is greater than 0.5, you remove that box from the list. So, you are ensuring that if there are multiple boxes that have a 50 percent overlap with each other, you retain only one of them.

We will see a little later that we would not just follow this, we will also follow trying to find which of these boxes has the highest confidence on the object being in the box, we will see that in a few slides from now. So, this process is known as non-maximal suppression NMS, which is extensively used in detection today as we will see.

(Refer Slide Time: 25:39)

**Histograms of Oriented Gradients<sup>2</sup>**

NPTEL  
National Programme on Technology Enhanced Learning

detection window slides over an image

at each location where the window is applied, gradients are computed

window is evenly partitioned into cells and each pixel of the cell contributes to cell gradient orientation histogram

orientation histograms for overlapping 2x2 blocks of cells are normalized and collected to form the final descriptor  
This is called Contrast Normalization

Final descriptor

SVM

Yes

No

Classification

<sup>2</sup>Dalal and Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

Vineeth N B. (IIT-H) | 7.1 CNNs for Detection | 15 / 44

That was the Viola Jones algorithm for phase detection. Very popular. Was used in several technologies. Another popular approach at that time was known as the Histogram of Oriented Gradients. We have seen this when we talked about extracting handcrafted features from images. But we will now talk about it as to how they are used for the task of pedestrian detection.

So, if you have say a pedestrian given in an image, you have a detection window that slides over an image and gets all your gradients. Could be a sobel filter, an elegy filter so on, so you have all of your gradients. Now you divide the entire image into cells and in each cell you draw out a histogram of orientations of gradients. We have done this before with sift so you

perhaps know how to do it. Once you get these histograms of oriented gradients in each of these cells, then you consider overlapping  $2 \times 2$  blocks of cells.

So you see here that you have 4 different  $2 \times 2$  cells. So they could be overlapping. But you take a set of  $2 \times 2$  cells, normalize the histograms in all of those 4 blocks and that is what you define as the final histogram for the center pixel. When you do this for all the cells in your image, you would get a final descriptor which looks like this where you have a set of histograms of oriented gradients across different locations in the image. How do you use this? Once you get this final descriptor you then train a support vector machine to say whether this is a pedestrian or not a pedestrian. So, this was introduced in this work in 2005 by Dalal and Triggs.

(Refer Slide Time: 27:38)

The slide, titled "HOG: SVM Weights", illustrates the process of generating HOG descriptors and applying SVM weights. It features the NPTEL logo and a small video inset of the presenter, Vineth N B. The main content is a sequence of seven images:

- Average gradient image over training examples:** A grayscale image showing the average gradient of training examples, appearing as a blurry, light-colored shape.
- Maximum positive and negative SVM weights respectively centred on the block:** Two grayscale images showing the maximum positive and negative SVM weights, appearing as dark, pixelated shapes.
- Test Image:** A color image of a person walking on a path.
- HOG descriptor for test image:** A grayscale image showing the HOG descriptor for the test image, appearing as a grid of small, dark, oriented gradient histograms.
- HOG descriptor weighted by positive and negative SVM weights respectively:** Two grayscale images showing the HOG descriptor weighted by positive and negative SVM weights, appearing as dark, pixelated shapes.

At the bottom of the slide, there is a small video inset of the presenter, Vineth N B (IIT-H), and a footer indicating the slide number 16 / 44 and the topic 7.1 CNNs for Detection.

So, you can also visualize as to how the weights of the support vectors look in this particular scenario. So, if you took the average gradient image over all your training examples, you can visualize what are the positive and negative SVM weights. Remember these are your different dimensions here because the cells are your dimensions of your entire descriptor vector. So, you can now imagine positive and negative SVM weights after you learn a support vector machine.

Given a test image and its corresponding histogram of oriented gradient descriptor, you can multiply these descriptors by the positive and negative weights and you would get something

like this and you notice here that looking at the positive weights it would be easy for the SVM to classify this as a pedestrian being in an image.



(Refer Slide Time: 28:31)

**Image Pyramid**

To identify images of all sizes, repeat the process across multiple image scales

$score(l, p) = w \cdot \phi(l, p)$

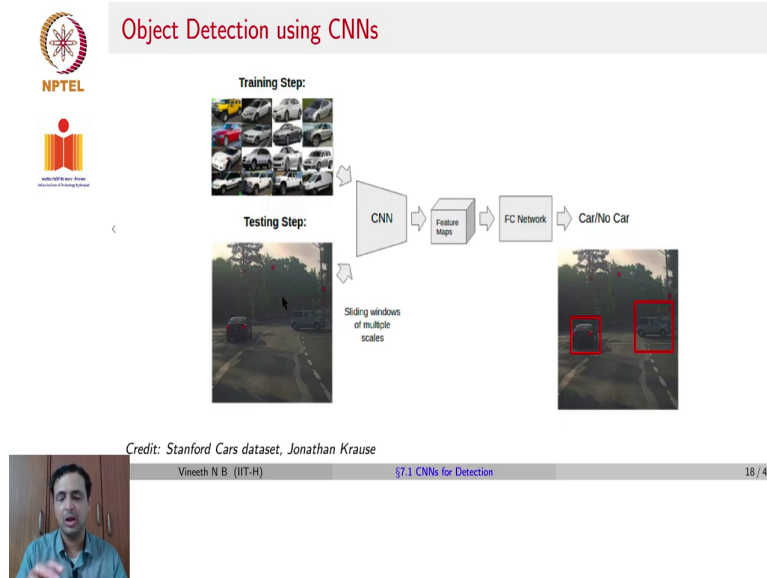
Image pyramid      HOG feature pyramid

Vineeth N B (IIT-H)      37.1 CNNs for Detection      17 / 44

This was also extended to be done at a multi-resolution level, where you have an image pyramid and you perform the same approach at each stage of the pyramid. So, for example, at the lowest resolution you have one specific block and you would get a score using a support vector machine for that particular window in the lowest resolution image. You get a certain score for a pedestrian being there. You would do this similarly for the higher resolution and the highest resolution images and then you can use various techniques to integrate the scores. You could integrate the scores the way we did it for pyramid matching by giving a high score for the highest resolution and a low score for the lowest resolution.

Or you could take a vote or you could use other kinds of heuristics to combine these scores to find out whether a person was in an image. These were the different approaches that were used for detection before deep learning came into the picture.



(Refer Slide Time: 29:38)



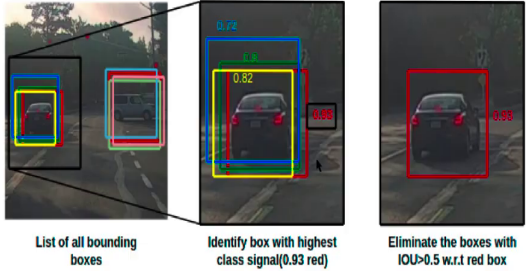
Now let us try to see how one would do object detection using CNNs in the simplest possible manner. If you had say images of cars during training, you would train a CNN on various images of cars to learn a car classifier. So you would use a standard cross entropy loss. CNN, a simple CNN. At test time, when you get an image where a car could be located anywhere in the image, you take different sliding windows of multiple scales.

So, you may have to take say 100, 000 sliding windows potentially from this image and each of those sliding windows you give as input to the CNN assuming that they all are of the same size or you can bring them to the same size. You give that as input to the CNN and that CNN now classifies each window in the original image as belonging to a car or not a car. So, based on that you would detect a car here and here.

(Refer Slide Time: 30:42)


  


### Non-Maximum Suppression using Class Scores





List of all bounding boxes      Identify box with highest class signal(0.93 red)      Eliminate the boxes with IOU>0.5 w.r.t red box

Vineeth N B (IIT-H)      §7.1 CNNs for Detection      19 / 44



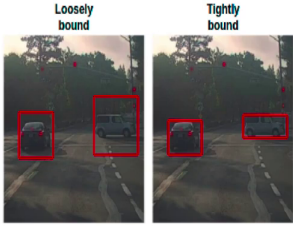
Once again in this approach you can include non-maximum suppression to improve the performance but here as we said before once you get the list of all possible bounding boxes where a car could be present, you consider a bounding box with the highest class signal. So, in this case, the red box would be the box with the highest class signal 0.93. Now you take the IoU of all the other bounding boxes with respect to this bounding box, this red bounding box. If any of them have a high overlap or a high IoU, with this red bounding box, you eliminate them and retain only the red bounding box because it had the highest confidence of a car present in that bounding box.

(Refer Slide Time: 31:34)


### Object Detection using CNNs and Sliding Windows: Disadvantages

- Bounding boxes may not be tightly around the object



- Evaluating a CNN network on all sliding windows is computationally intensive and time consuming

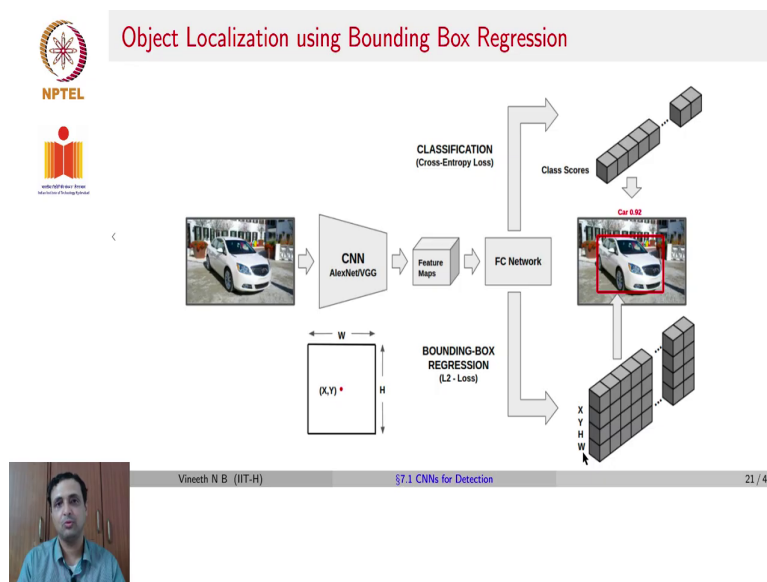
Vineeth N B (IIT-H)      §7.1 CNNs for Detection      20 / 44



Do you see problems with this approach? This is the simplest way of adapting CNNs for object detection but do you see any issues? There are actually a few of them. Firstly, the bounding boxes may not be tightly around the object. You can see here that some of these bounding boxes are fairly loose. We ideally want a tight bounding box so that we exactly focus on only that part of the object, that part of the image that contains the object. That could happen in this kind of an approach.

The other problem here is obviously that you have to evaluate thousands of sliding windows through the CNN to be able to find which of them contain your object of choice. We are going to talk about methods that overcome this issue.

(Refer Slide Time: 32:33)



But before we go there, we will talk about object localization which as we said was a precursor to detection. So, object localization can be achieved by, if you had only a single object in an image, you can then give that as input to a CNN, you get your feature maps, you get your fully connected network. Now, you branch out your CNN into two parts, one part that gives you a classification score which can be learnt using a cross entropy loss, that is the standard CNN for classification that we have spoken about so far, but we could also do a bounding box regression.

So, you could now try to see what are the exact coordinates of the bounding box say with respect to the entire image or with respect to any other box in the image. You could see what is the offset and you also learn that as part of your learning procedure. So, your fully

connected network at the end would have two heads, a classification head and a regression head. The classification head would require clock cross entropy loss, the regression head which will learn in x, y the top left corner of the box, a height and the width. 4 coordinates, 4 values.

And these values can be learnt using a normal L2 or mean square loss, so the entire network would be trained by the sum of these two losses to not only classify the image but also localize the image.

(Refer Slide Time: 34:16)

The slide features the NPTEL logo on the left. The main title is "OverFeat: Integrated Recognition, Localization and Detection<sup>3</sup>". Below the title, it lists:
 

- Winner of ILSVRC 2013 Localization Challenge
- Intuition:** Avoid computation time over sub-windows by applying filter directly to image

 To the right, a diagram illustrates the process. It shows an "Image" with a 3x3 filter (yellow and green) applied to it, resulting in a localized feature map. Below this, "Sub-window 1" and "Sub-window 2" are shown, each with the same 3x3 filter applied to a specific region of the image, demonstrating that the filter is applied directly to the sub-window rather than the entire image for each location.

<sup>3</sup>Sermanet et al, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks, ICLR 2014

Vineeth N B (IIT-H) 57.1 CNNs for Detection 22 / 44

One of the first successes of a good localizer in CNN was OverFeat. OverFeat was the winner of the ImageNet localization challenge. Every year, ImageNet, in addition to having the classification challenge also had a localization challenge and OverFeat was the winner of this challenge in 2013 and the way OverFeat did localization was to ensure that your entire network is convolutional instead of having fully connected layers. So, this avoids computation time over sub windows by applying filter directly to the image, let us see how.

(Refer Slide Time: 35:02)

**OverFeat: Integrated Recognition, Localization and Detection<sup>4</sup>**

<sup>4</sup>Sermanet et al, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks, ICLR 2014

Vineeth N B (IIT-H) §7.1 CNNs for Detection 23 / 44

So, given an input which is say  $14 \times 14$  or you pad and make it  $16 \times 16$  you perform  $5 \times 5$  convolution this is only an example for visualization you get a  $10 \times 10$  output assume if you did not do padding and if you do 2 pixel padding it becomes  $12 \times 12$  then a  $2 \times 2$  pooling makes it  $5 \times 5$ . A  $5 \times 5$  convolution makes it  $1 \times 1$  and now using multiple one cross one convolutions you can get an output directly without having to perform any fully connected layer operations.

(Refer Slide Time: 35:45)

**OverFeat: Integrated Recognition, Localization and Detection<sup>5</sup>**

The network used is fully convolutional, replacing fully connected layers with convolutional layers. Why?

<sup>5</sup>Sermanet et al, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks, ICLR 2014

Vineeth N B (IIT-H) §7.1 CNNs for Detection 24 / 44

So, this idea of OverFeat made computations significantly faster as you can see now, instead of using a fully connected layer OverFeat replaces it with a fully convolutional layer which

finally has  $1 \times 1$  convolution similar to what we saw with InceptionNet if you recall or GoogleNet and the this OverFeat approach uses that to get your final performance. Why is this useful? Because we remember we said that fully connected layers use up a lot of parameters. That is what we said and using a fully convolutional approach overcomes this and hence can speed up performance.