# Convolutional Neural Networks (CNNs)

**Assignment # 4 Resnet assignment……**

- **ReseNet for SIGN dataset.**
- **Transfer learning….**

**https://github.com/abdur75648/Deep-Learning-Specialization-Coursera/tree/8da5959ad709d0859de42cb18931e409e98cbc8c/Convolutional%20Neural%20Networks/week2**

**Assignment # 5 (Object detection, Submission is up to you)**

**https://github.com/abdur75648/Deep-Learning-Specialization-Coursera/blob/main/Convolutional%20Neural%20Networks/week3/1%20Car_detection%20(Autonomous_driving)/Autonomous_driving_application_Car_detection.ipynb**

**https://www.v7labs.com/blog/yolo-object-detection**

**Notes and for details of YOLO Network architecture:**

**https://github.com/mbadry1/DeepLearning.ai-Summary/tree/master/4-%20Convolutional%20Neural%20Networks**

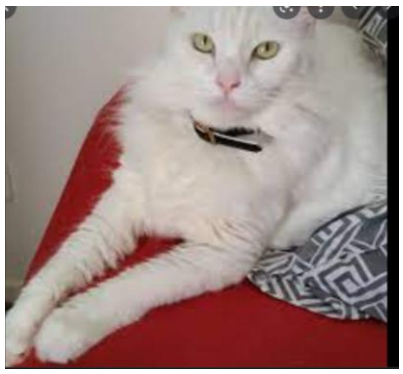
**Useful tips:**

## 1. Using Open-Source Implementation

- We have learned a lot of NNs and ConvNets architectures.
- It turns out that a lot of these **NN are difficult to replicated**. because there are some details that may not presented on its papers. There are some other reasons like:
    - Learning decay.
    - Parameter tuning.
- A lot of deep learning **researchers are opening sourcing** their code into Internet on sites like Github.
    - [c:\>**git clone https://github.com/KaimingHe/deep-residual-networks.git** ]
- If you see a research paper and you want to build over it, the first thing you should do is to **look for an open source implementation** for this paper.
- Some **advantage** of doing this is that you might **download the network implementation along with its parameters/weights**. The author might have used multiple GPUs and spent some weeks to reach this result and its right in front of you after you download it.

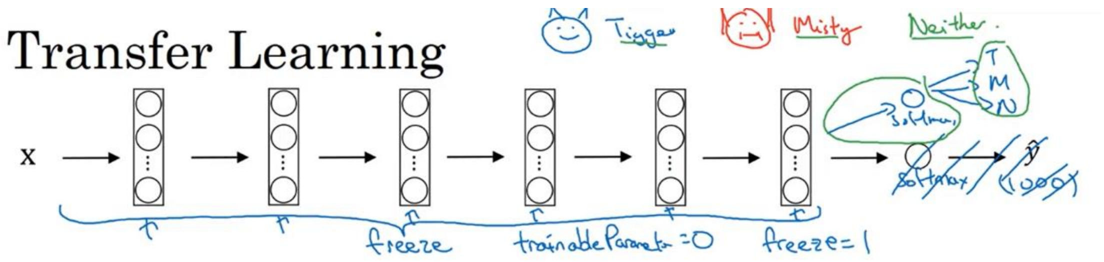
## 2. Transfer learning:

- **Your classification tasks: Tigger cat or Misty cat or neither:**



**Case1:** Small dataset < 10000

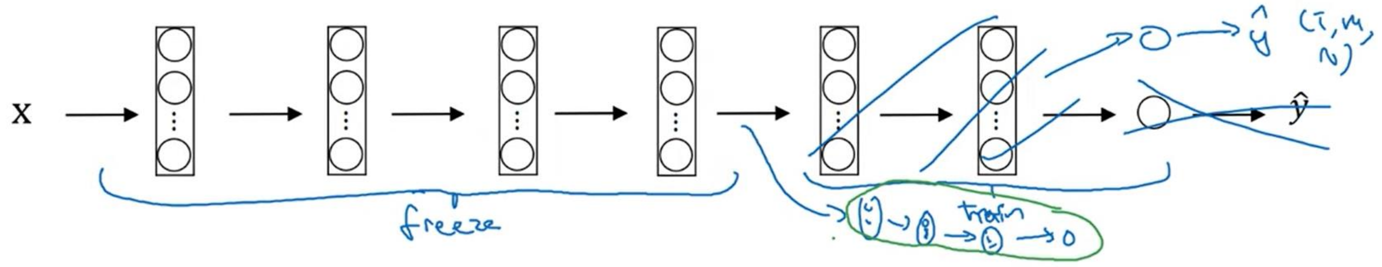
- Use the pretrained model (use pretrained weights/bias instead initialized with random values) on ImageNet dataset (**14 million with 1000 classes**)



```
resnet_model = models.resnet34(pretrained=True)
for param in resnet_model.parameters():
    param.requires_grad = False
num_features = resnet_model.fc.in_features

resnet_model.fc = nn.Linear(num_features, 3)  # 3 output classes
```

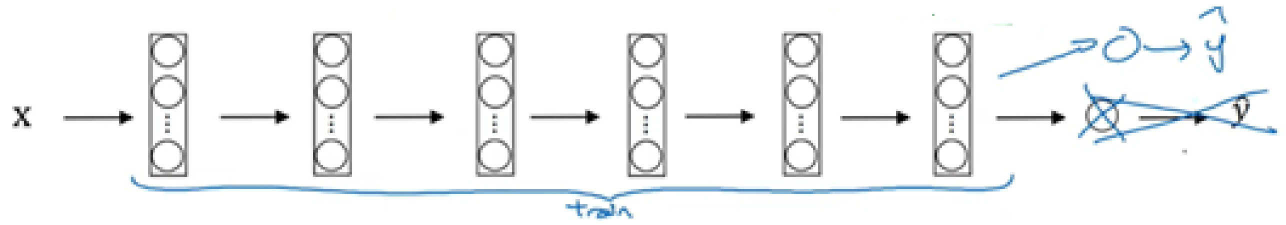
## Case2: Medium dataset: let’s say >10000



```
resnet_model = models.resnet34(pretrained=True)
# Get a list of all the layers in the model
layers = list(resnet_model.children())
# Determine the halfway point
halfway = len(layers) // 2
# Freeze the first half of the layers
for layer in layers[:halfway]:
  for param in layer.parameters():
    param.requires_grad = False
# Leave the second half of the layers unfrozen for fine-tuning
for layer in layers[halfway:]:
  for param in layer.parameters():
    param.requires_grad = True

# Replace the last fully connected layer (fc) with a new one for 3-class classification
num_features = resnet_model.fc.in_features
resnet_model.fc = nn.Linear(num_features, 3)  # 3 output classes
```

## Case3: Larger dataset (1million)



```
resnet_model = models.resnet34(pretrained=True)

for param in resnet_model.parameters():

    param.requires_grad = True

num_features = resnet_model.fc.in_features
resnet_model.fc = nn.Linear(num_features, 3)  # 3 output classes
optimizer = torch.optim.Adam(resnet_model.parameters(), lr=1e-4)
```

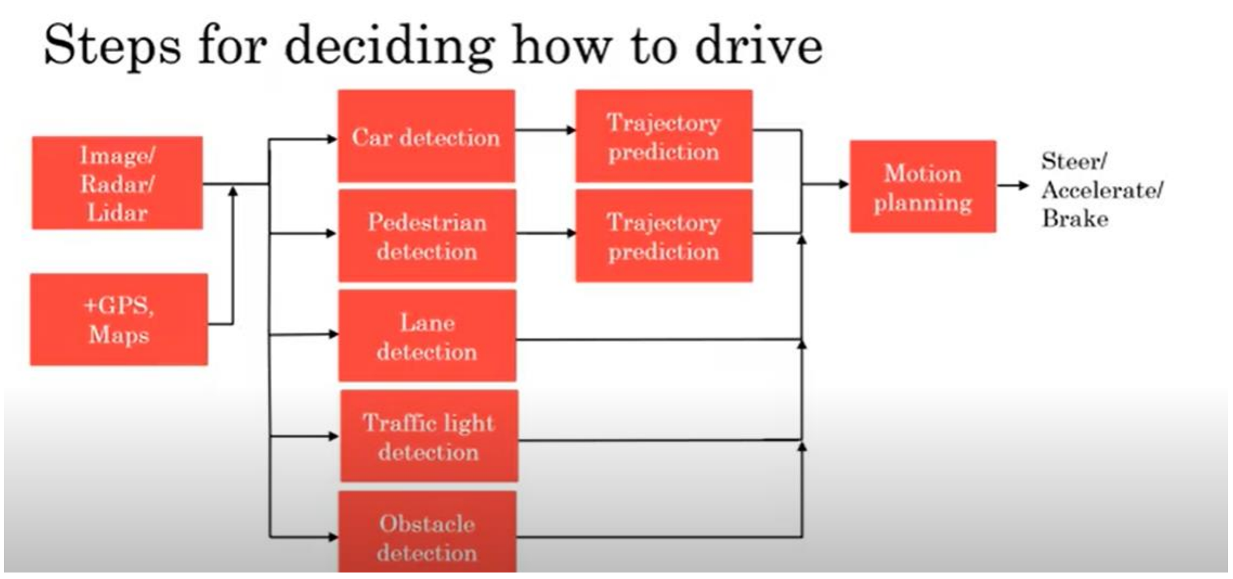
Transfer Learning

- If you are using a specific NN architecture that has been trained before, you can use this pretrained parameters/weights instead of random initialization to solve your problem.
- It can help you boost the performance of the NN.
- The pretrained models might have trained on a large datasets like ImageNet (**14 million with 1000 classes**), Ms COCO (**328,000 images**), or pascal and took a lot of time to learn those parameters/weights with optimized hyperparameters. This can save you a lot of time.
- Lets see an example:
    - Lets say you have a cat classification problem which contains 3 classes Tigger, Misty and neither.
    - You don't have much a lot of data to train a NN on these images.
    - Andrew recommends to go online and download a good NN with its weights, remove the softmax activation layer and put your own one and make the network learn only the new layer while other layer weights are fixed/frozen.
    - Frameworks have options to make the parameters frozen in some layers using trainable = 0 or freeze = 0
- Another example:
    - What if in the last example you have a lot of pictures for your cats.
    - One thing you can do is to freeze few layers from the beginning of the pretrained network and learn the other weights in the network.
    - Some other idea is to throw away the layers that aren't frozen and put your own layers there.
- Another example:
    - If you have enough data, you can fine tune all the layers in your pretrained network but don't random initialize the parameters, leave the learned parameters as it is and learn from there.

**Object detection:** Learn how to apply your knowledge of CNNs to one of the toughest but hottest field of computer vision: Object detection. For this purpose, will discuss YOLO algorithms. Application of YOLO:

- self-driving car,
- wildlife (animal detection),
- security (restrict the people to pass through a certain area for security reason)

## Self driving car basic modules:



## Deep learnings algorithms for Computer vision problems:

1) **Image Classification:**

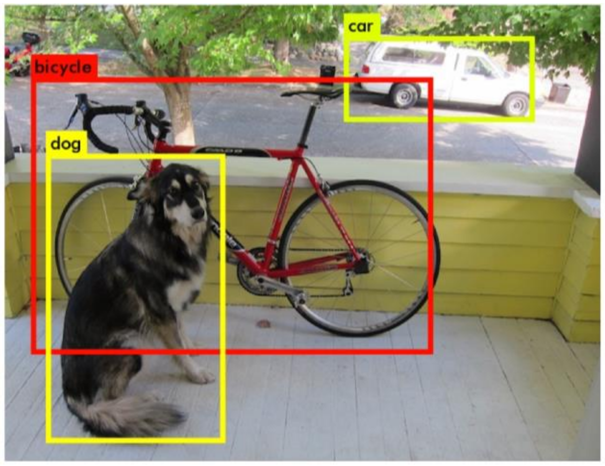
- Classify an image to a specific class.
- The whole image represents one class.
- We don't want to know exactly where the object are.
- Usually only one object is presented.

2) **Classification with localization:**

- Given an image we want to learn the class of the image and where are the class location in the image.
- We need to detect a class and a rectangle of where that object is.
- Usually only one object is presented.

**3) Object detection: (Self Driving Car)**

- Given an image we want to detect all the object in the image that belong to a specific classes and give their location.
- An image can contain more than one object with different classes.


car
bicycle
dog

**Summary:**

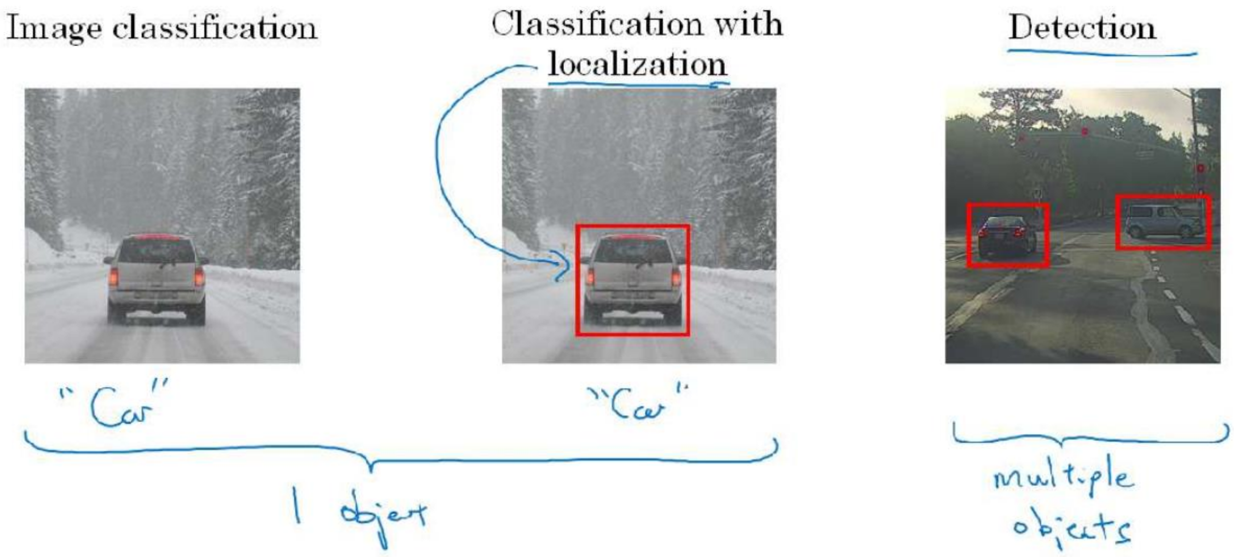
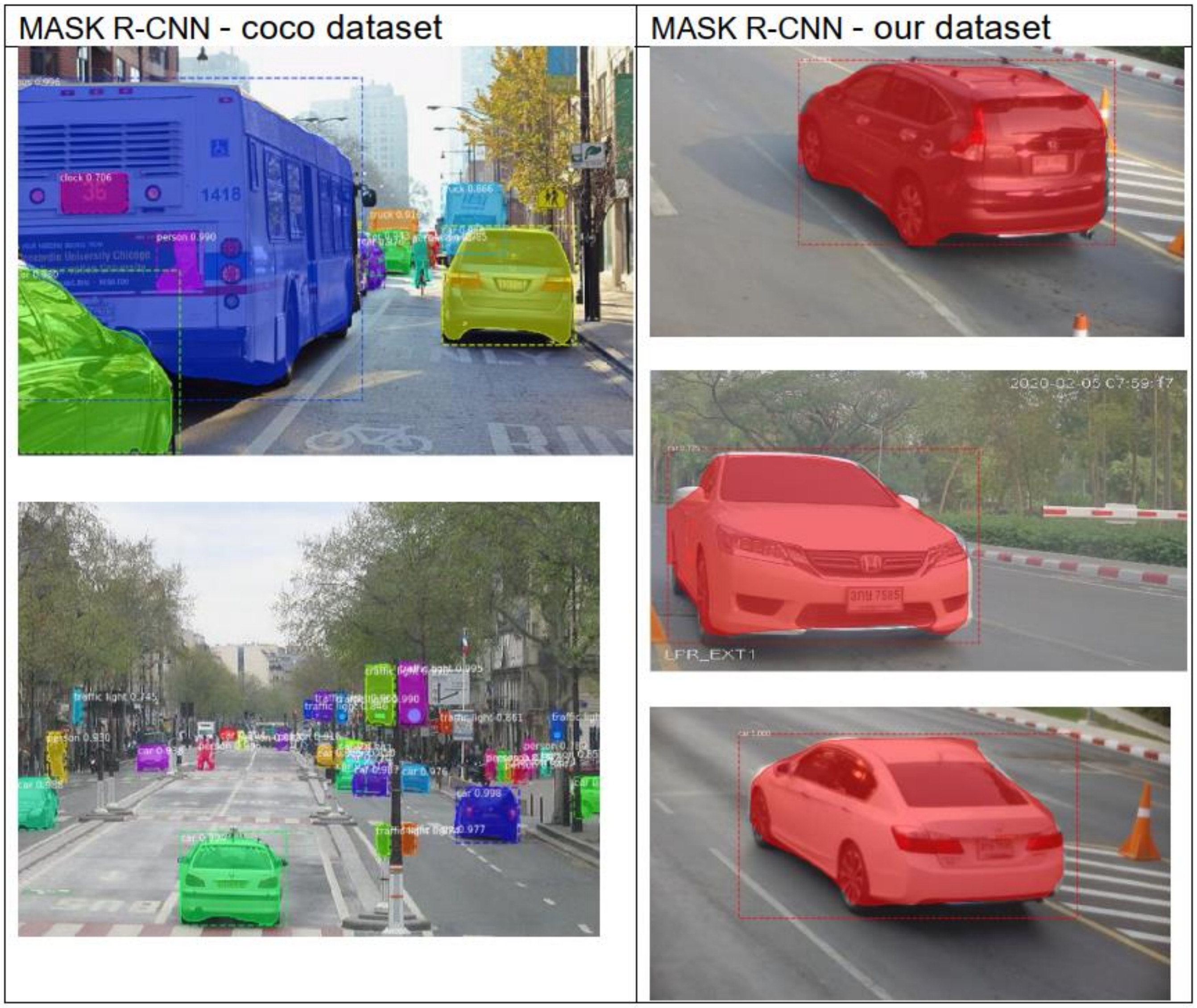


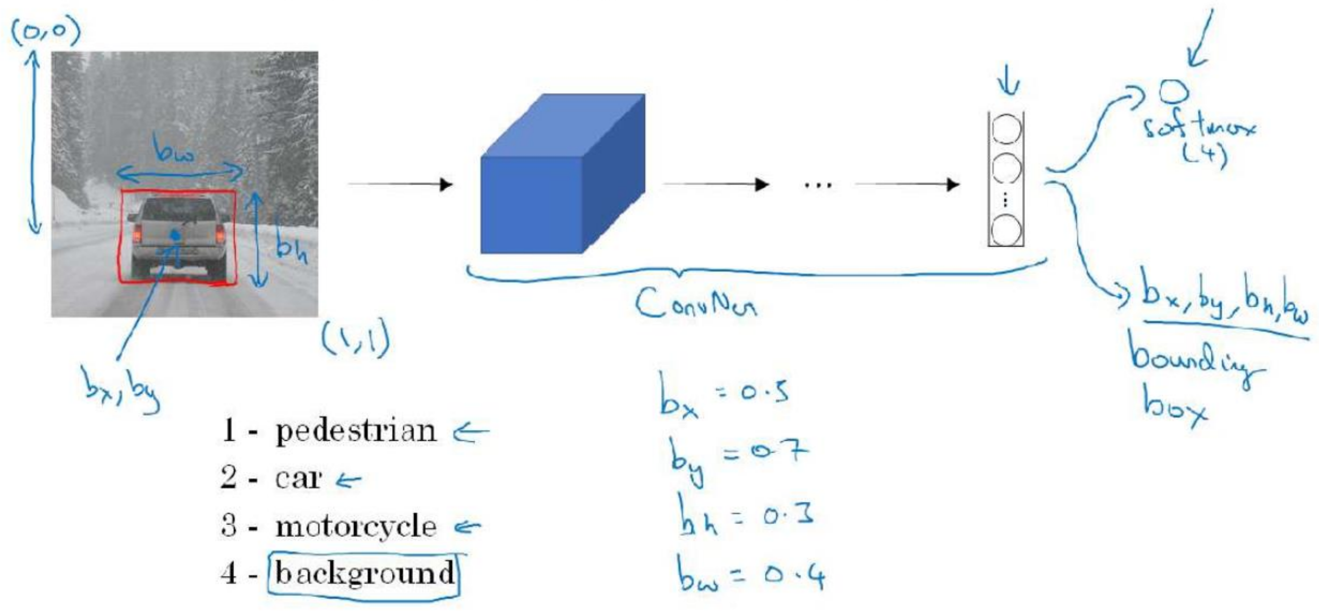
Image classification
Classification with localization
Detection
"Car"
"Car"
1 object
multiple objects

4) **Segmentation algorithm (Instance segmentation):**

- 3 tasks done in parallel
  - Classification
  - Bounding box regression
  - Instance segmentation (mask)
- Kisming et.at., 2017 MASK R-CNN:



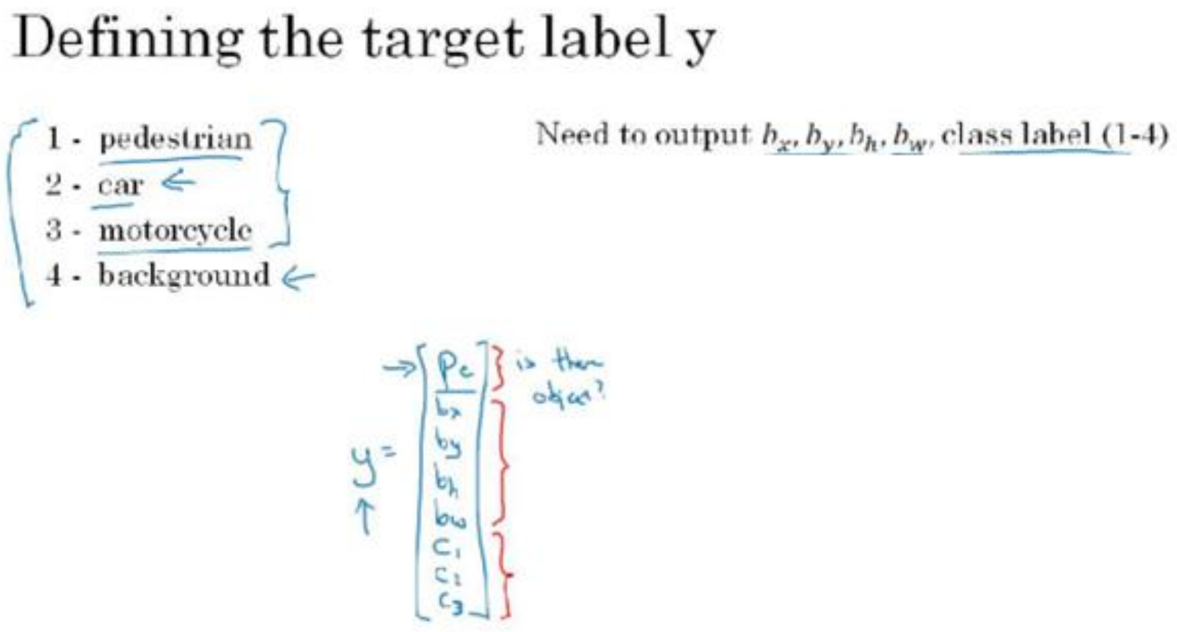
## Classification with localization



To make **image classification** we use a **Conv Net with a Softmax** (Network architecture) attached to the end of it. X is input image and Y is hot encoded vector [0; 1; 0; 0] (dataset label)

1) To make **classification with localization**
   a. We use a **ConvNet** with a softmax attached to the end of it and also regress four numbers **`bx`, `by`, `bh`, and `bw`** to tell us the location of the class in the image. **(Network architecture change)**
   b. The **Y** label of **dataset** should contain these four numbers with the class information. **(Dataset label change)**

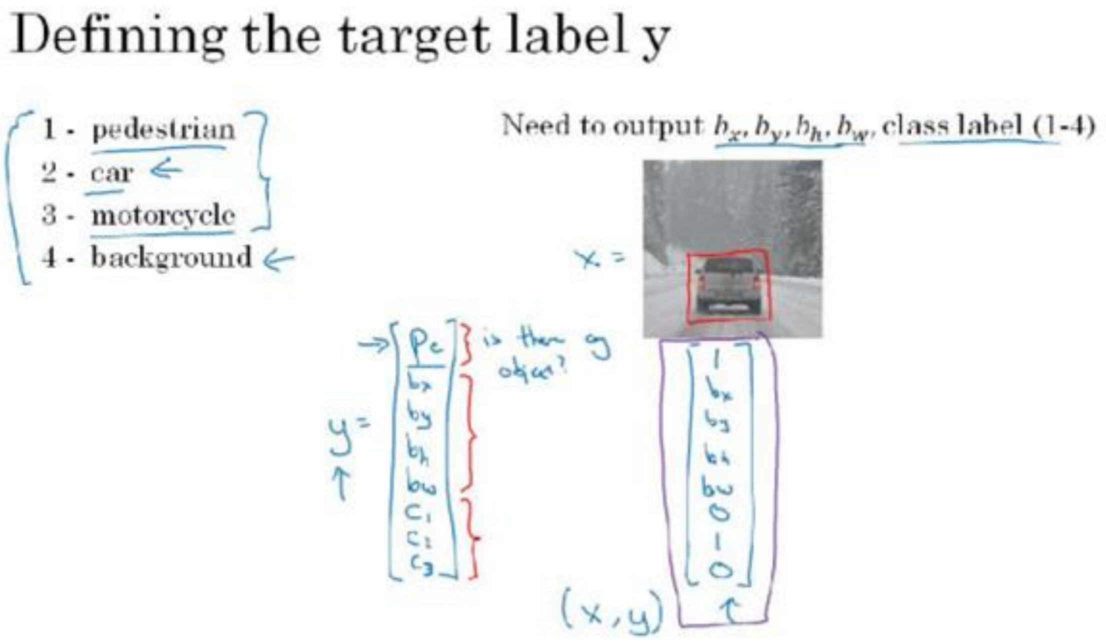
**Dataset preparation of classification with localization.** **(define label for image contains object vs not object)**



Defining the target label y

1 - pedestrian
2 - car
3 - motorcycle
4 - background

Need to output $b_x, b_y, b_h, b_w$, class label (1-4)



Defining the target label y

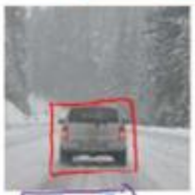
1 - pedestrian
2 - car
3 - motorcycle
4 - background

Need to output $b_x, b_y, b_h, b_w$, class label (1-4)

# Defining the target label y

$$\left[\begin{array}{l} 1 \text{ - } \underline{\text{pedestrian}} \\ 2 \text{ - } \underline{\text{car}} \leftarrow \\ 3 \text{ - } \underline{\text{motorcycle}} \\ 4 \text{ - background} \leftarrow \end{array}\right]$$

Need to output $\underline{b_x, b_y, b_h, b_w}$, $\underline{\text{class label (1-4)}}$

$x =$ 

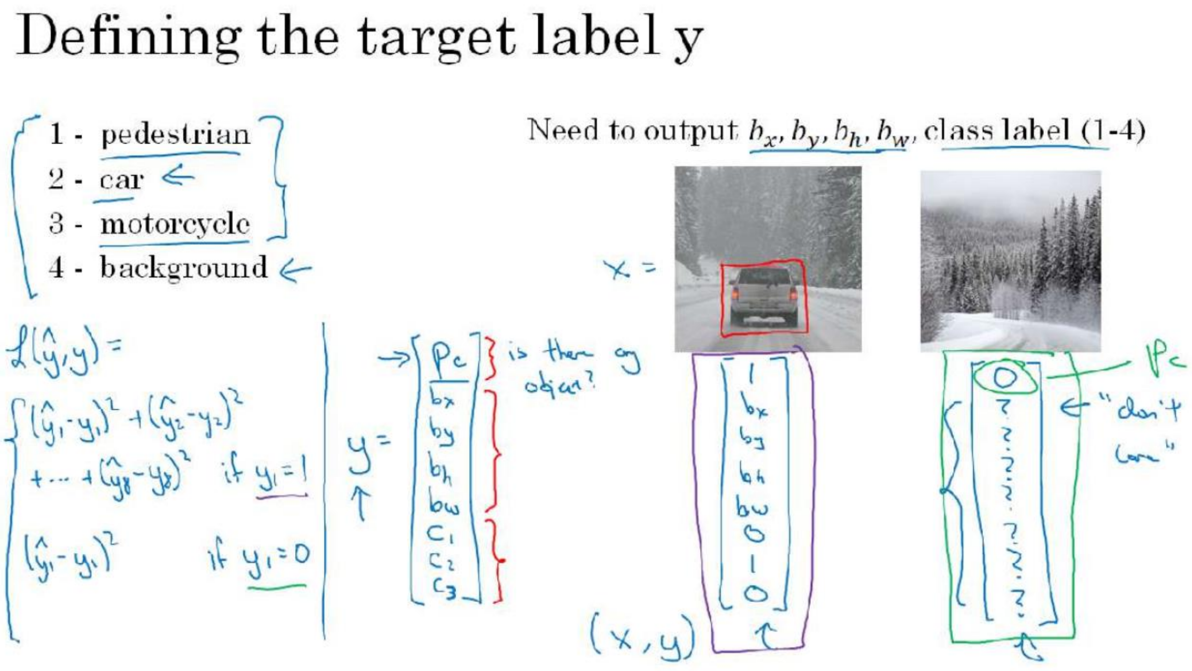
$$\uparrow y = \begin{bmatrix} \rightarrow p_c \\ \hline b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \begin{array}{l} \} \text{ is there an object?} \\ \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \end{array}$$

$$(x, y) \quad \boxed{\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}} \uparrow \qquad \boxed{\begin{bmatrix} \boxed{0} \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}} \uparrow$$

$p_c$

$\leftarrow$ "don't care"

## Loss function:



### Defining the target label y

1 - pedestrian
2 - car
3 - motorcycle
4 - background

Need to output $b_x, b_y, b_h, b_w$, class label (1-4)

$$\mathcal{L}(\hat{y},y) = \begin{cases} (\hat{y}_1-y_1)^2 + (\hat{y}_2-y_2)^2 + \dots + (\hat{y}_8-y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1-y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

is there any object?

$x =$

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

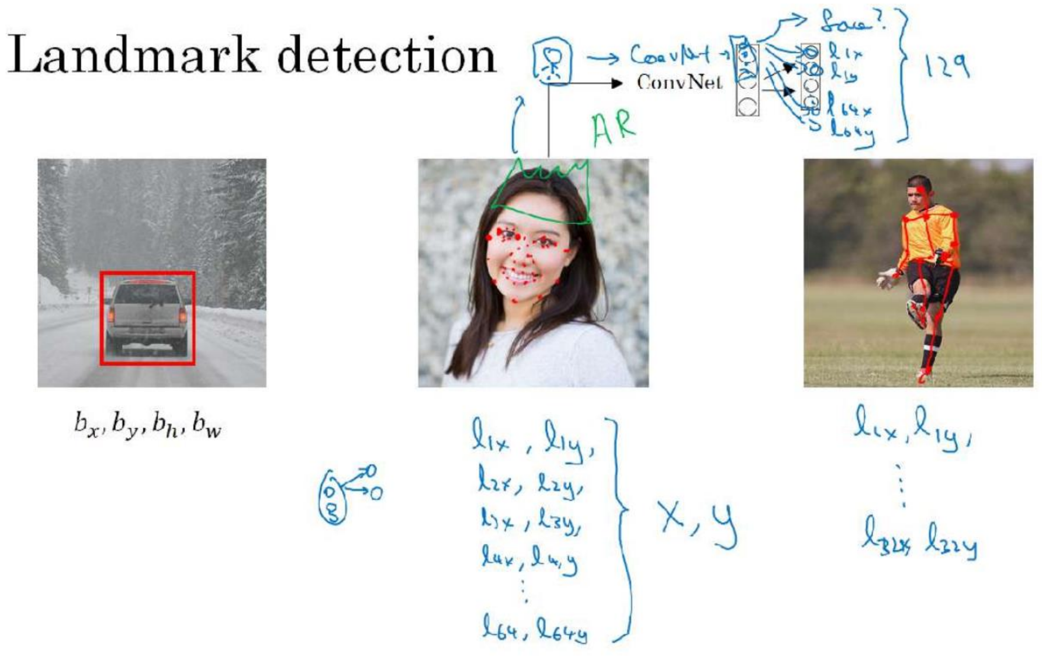
$(x, y)$

$P_c$

"don't care"

- The loss function for the Y we have created (Example of the square error) is shown in above figure.
- In practice we use **Sigmoid** for **`probability of class (Pc)`** (square error is still ok)
- For c1, c2, and c3 use **SoftMax** using CE loss
- **Squared error** for the bounding box (**bx, by, bw, bh**)

## Land marks detection for human: emotion recognition and pose identification:



- In the previous section, we saw how you can get a neural network to output four numbers of **bx, by, bh, and bw** to specify the bounding box of an object you want a neural network to localize.
- In more general cases, you can have a **neural network just output X and Y coordinates of important points on the image**, sometimes called **landmarks**, that you want the neural networks to recognize.
  Let's go through a few examples
- For example: human eye detector for wearing glasses
  - if you are working on a face recognition problem/application. You might want to know where eye is. Each person's eye has two landmarks on corner of eye. Now we can design the neural network architecture that has that predict face + eight points that show it is face and eyes belongs to this region. And you may design the Augmented reality application that can wear the sun glass by looking at these four landmarks points of eyes.
  - **Dataset X= input image, Y = [pc; l1x; l1y; l2x; l2y; l3x; l3y; l4x; l4y]**
  - **Last layer of CNN output these 9 outputs.**

- **Identification of person is alive or not may we paritally use the eye blinking, we may consider the multiple landmarks over the whole eye.**
- Changes in blinking frequency might indicate fatigue, stress, or certain medical conditions

- You might want some points (**68 landmarks**) on the face like corners of the eyes, corners of the mouth, corners of the nose, jaw of face and so on.

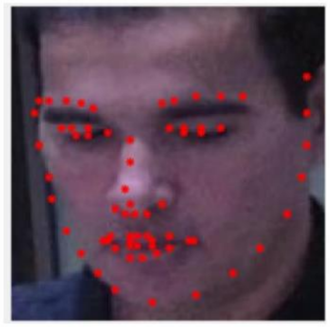
  - **Dataset X= input image, Y = [pc; l1x; l1y; l2x; l2y; l3x; l3y;…. L64x; l64y]**
  - **Last layer of CNN output these 129 outputs.**
  - **Label of Landmarks position should be consistence along all images**

- **This can help in a lot of applications for**
  - Building block for recognition of **emotions from faces**, **Facial expression of face, smile or not**
  - **Detecting the pose of the face**
  - **snapchat application and augmented reality filters and wear the crown on the face or weak glaass**

- People pose detection: define the mid point of cheat, left shoulder, left elbow, and the left wrist and similary for right shoulder, right elbow, and the right wrist.
- Person can hit the football or not.  / Person in a siting or standing position/
- **Activity prediction for a person**, we may predict **32 landmarks** from NN

## Used landmarks to generate better SR images.

- **Generate 68 heatmaps using** (Bulat et.al., 2017. **How far are we from solving the 2D & 3D Face Alignment problem?) work.**



**Use 4 heatmaps to represent four componets of a face marks are: (spread = 0.5)**

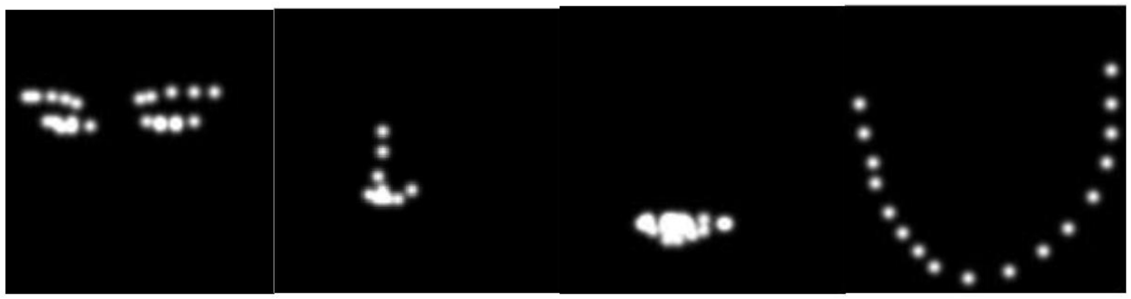
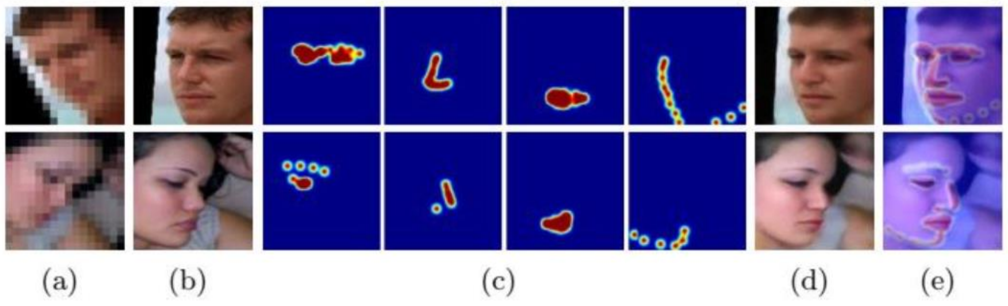


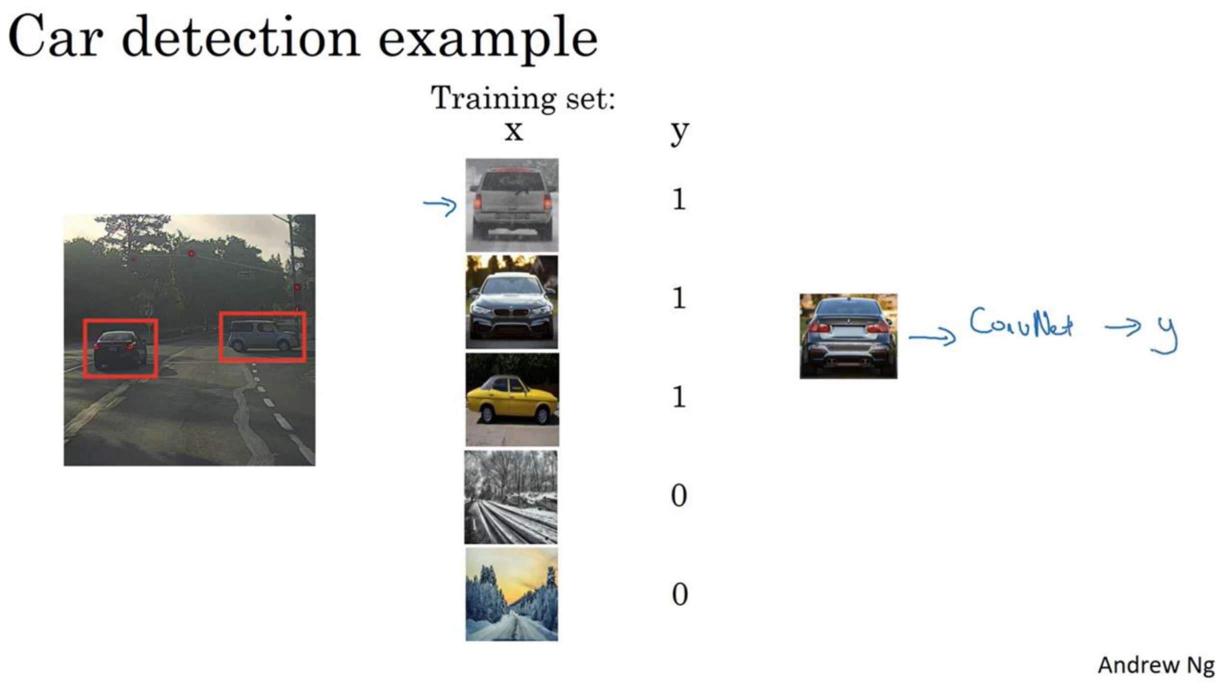
**Figure:11 For integrating training of CinCGAN with FAN, consider 4 heatmaps**

**Face Super-resolution Guided by Facial Componet HeatMaps Xin Yu – ECCV 2018**



## Object detection approach of sliding window with CNN for car detection

- For example lets say we are working on Car object detection.
- The first thing, we will **train a ConvNet (CONV-POOL-FC)** on closely cropped car images and non car images. [just like a simple classification algorithm]
- **Training process:**

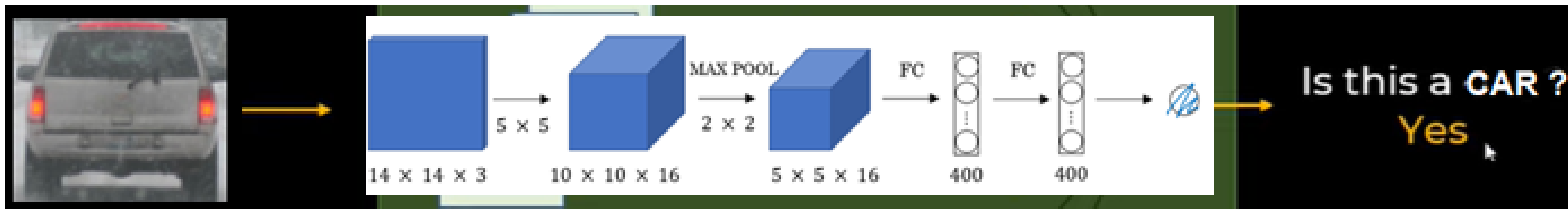


### Car detection example

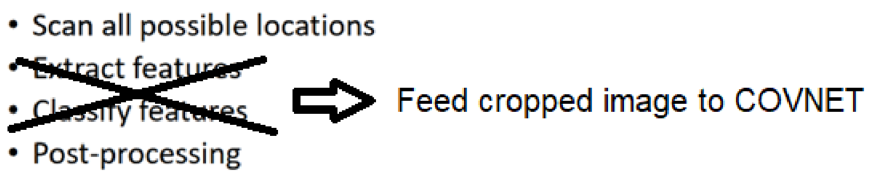
Training set:

| x | y |
|---|---|
| car image | 1 |
| car image | 1 |
| car image | 1 |
| non-car image | 0 |
| non-car image | 0 |

-

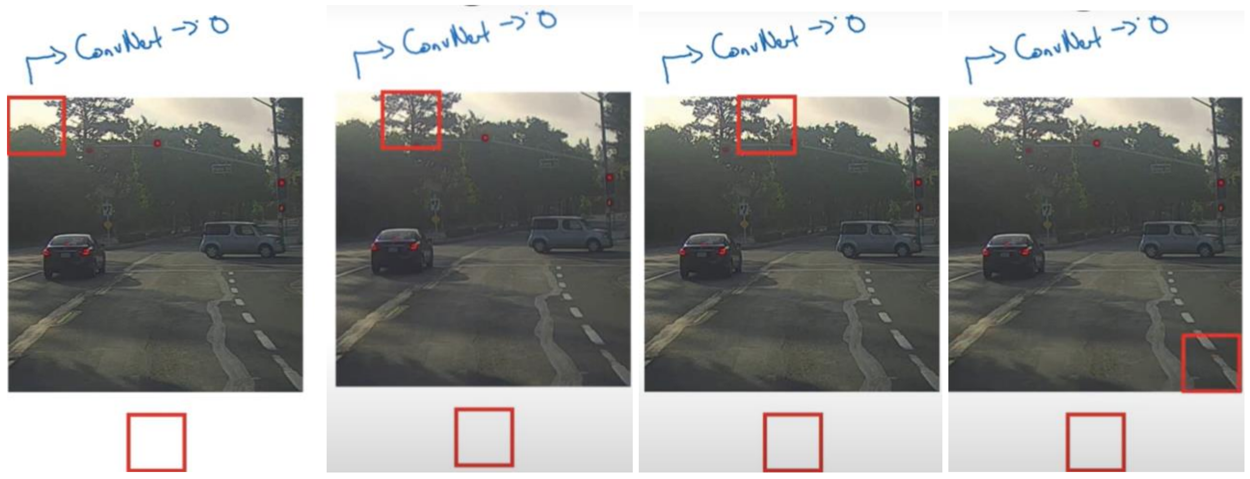


- **At test time** use a technique called the **sliding windows detection algorithm to detect the object**.



- Scan all possible locations
- ~~Extract features~~
- ~~Classify features~~
- Post-processing

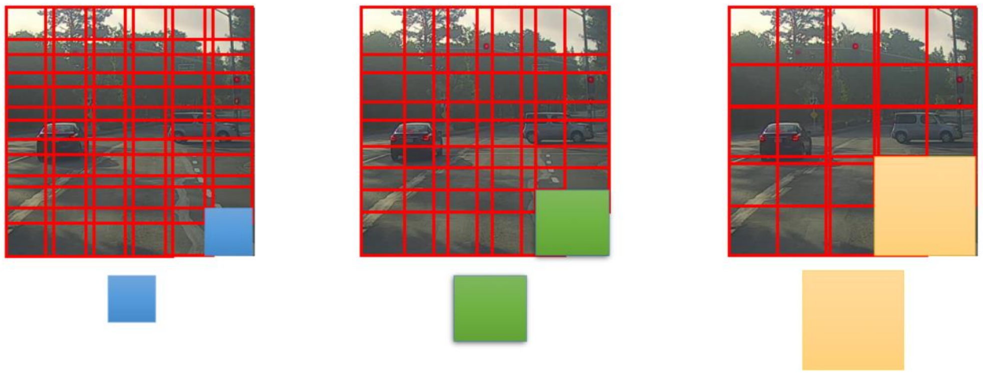
$\Rightarrow$ Feed cropped image to COVNET

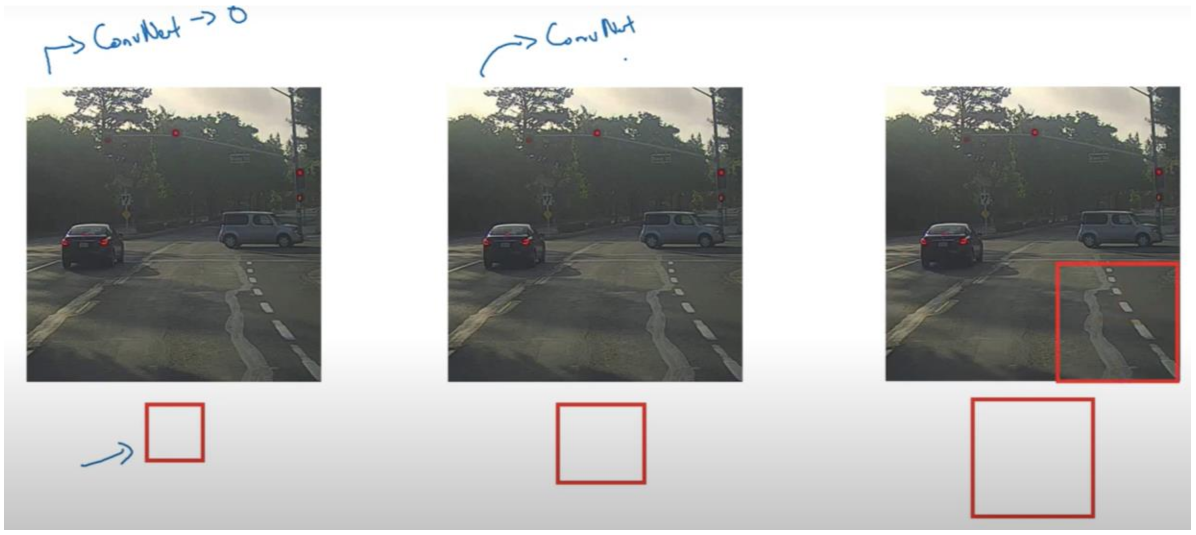


Select a window. we pass though the every part of image let say we have very small stride, and classify a 0 or 1 for every position.

Select a larger window and we pass though the every part of image let say we have very small stride, and classify a 0 or 1 for every position.

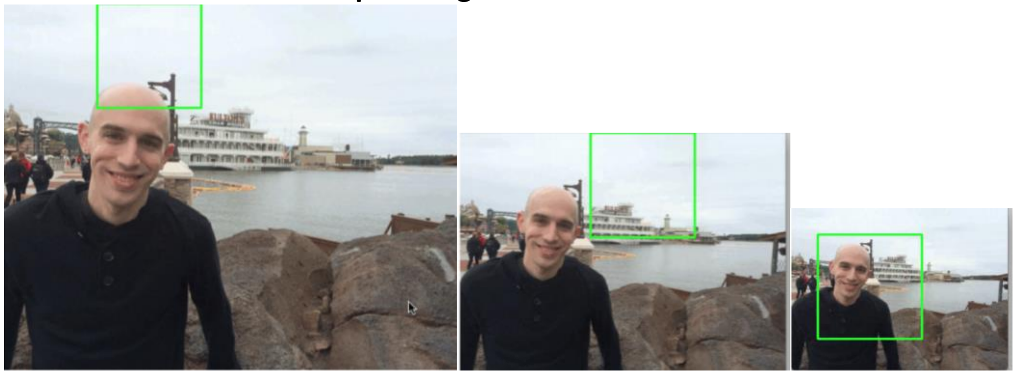
`





## Sliding windows detection algorithm:

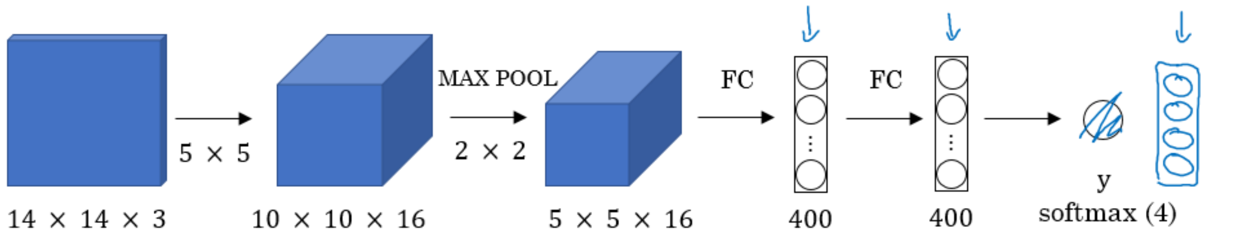
**Or use different scale of input image for a fixed window size:**

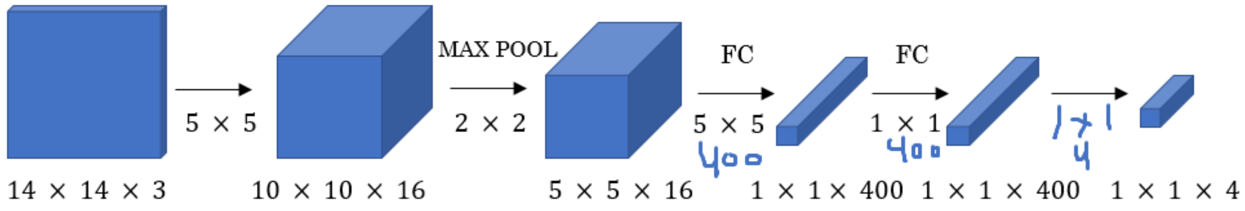


i. Decide a rectangle size. (**Try different sizes of image and work iteratively**)
ii. Crop your image into rectangles of the size you picked.
iii. Feed the cropped image into the ConvNet and decide if it's a car or not
iv. Use some strides to pick the next crop image to cover whole the image.
v. Pick larger/smaller image sizes and repeat the process from ii to IV.
**vi. Store the rectangles that contain the cars.**
vii. If two or more rectangles clam for object detection than do some posts process to decide for more accurate rectangle (Non max suppression).
viii. [We can train the COVNET for multiple object detection, four classes, Pedestrian, Car, Motorcycle, backgroud]

1. **Disadvantage of the sliding window is:**
    1) Computation time is high as we deal with different scale.
    2) Rectangle size may not perfect

- To solve this problem, we can implement the sliding windows with a ***Convolutional approach***.





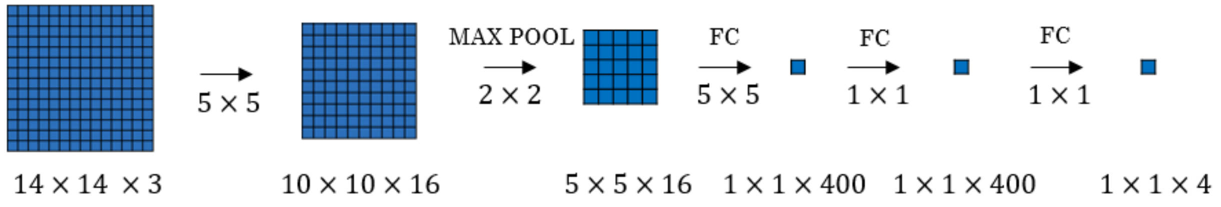
## Convolutional Implementation of Sliding Windows (It makes NW independent of size of input)

1. Turning FC layer into convolutional layers (predict image class from four classes):

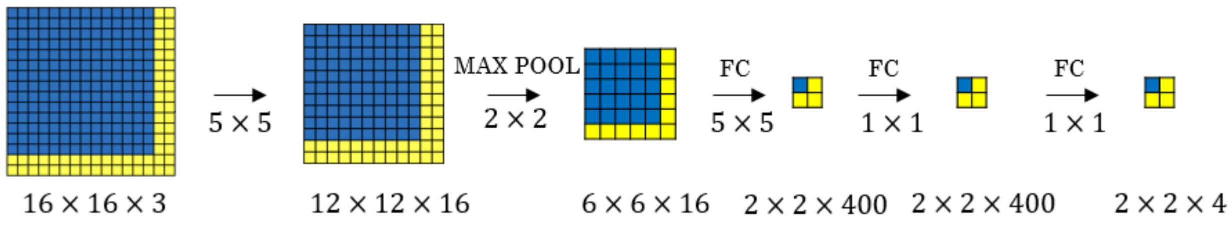
- As you can see in the above image, we turned the FC layer into a Conv layer using convolution layers

2. **Convolution implementation of sliding windows**:
2. [Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection usingconvolutional networks]

1) First let's consider that the ConvNet is trained (No FC all is conv layers): [**Training set image is 14x14x3**]. 1x1x4 activation pass through softmax predict 4 outputs

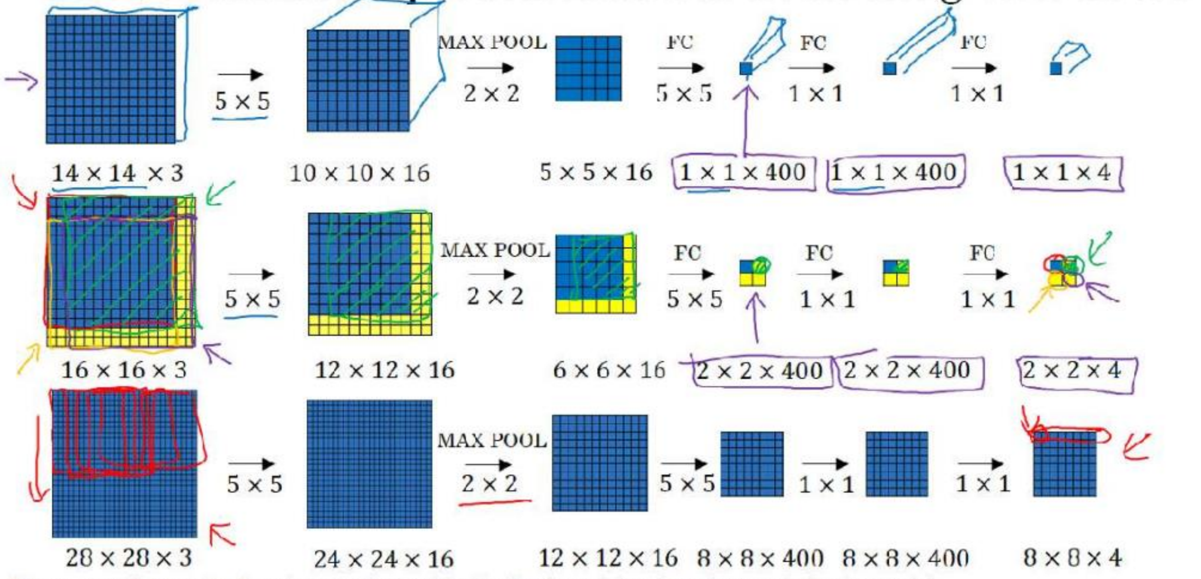


2) The convolution implementation will be as follows: **let consider larger size images at test time we output the 2x2x4. With sliding window alogiritthm we have to send 14 x14x3 cropped image 4 time to find either of 4 cropped images have a CAR or not.**



3)

## Convolution implementation of sliding windows



[Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

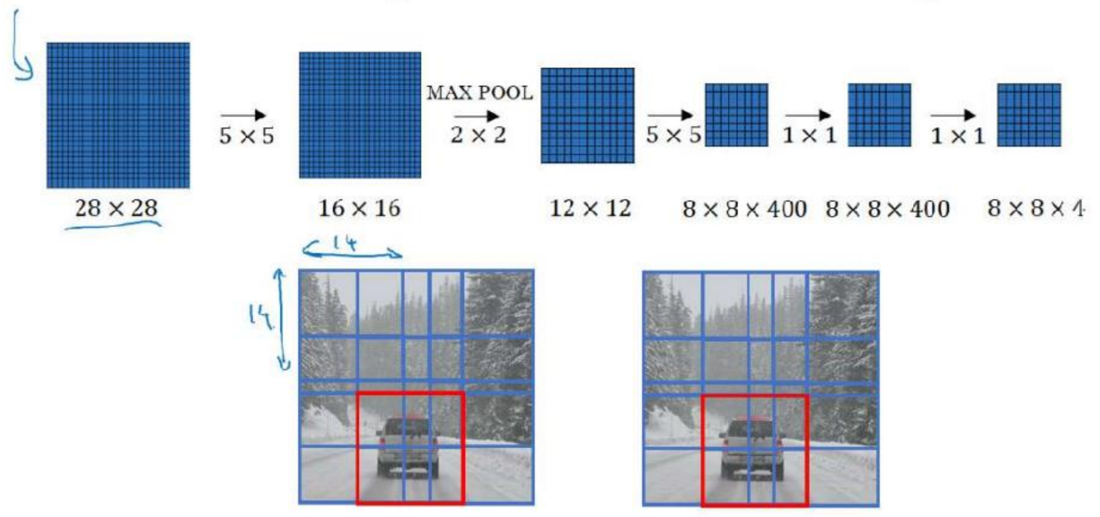
Consider the green output and see how it come, during the parallel computation of 4 outputs. We can see the green region activation region at each layer ultimately map to the top right output.

For 16x16x3 input image convolutional network produce 2x2 ouputs that  corresponding to 4 different crop 14x14x3 images.

Similarly 28x28x3 image produce 8x8x4 outputs.

Stride 2 is come due to max pool of 2x2.

## Convolution implementation of sliding windows

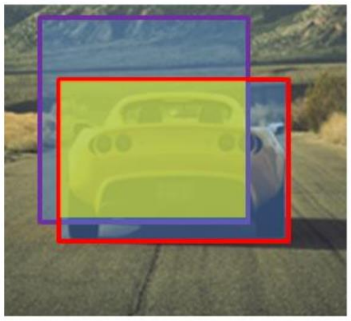


In above testing image instead of cropping input image in 14x14 images and send to convNet. Instead of that we use convolutional implementation of sliding window algorithm and process the whole image at once. It reduces a lot of computation. But still, we have one problem which is the rectangle position.

1. **The weakness of the algorithm** is that the position of the rectangle wont be so accurate. Maybe none of the rectangles is exactly on the object you want to recognize.
2. **Need to add bounding box prediction along the class classification.**

## Evaluation

- Intersection over union

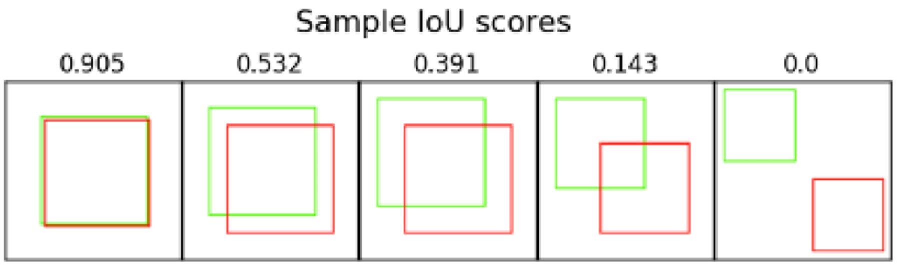


Intersection over union (IoU)

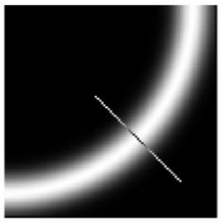
$$= \frac{\text{size of } ▨}{\text{size of } ▨}$$

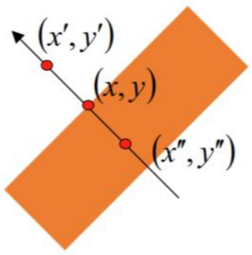
"Correct" if $\mathrm{IoU} \geq 0{,}5$

## Sample IoU



## Non-maximum suppression

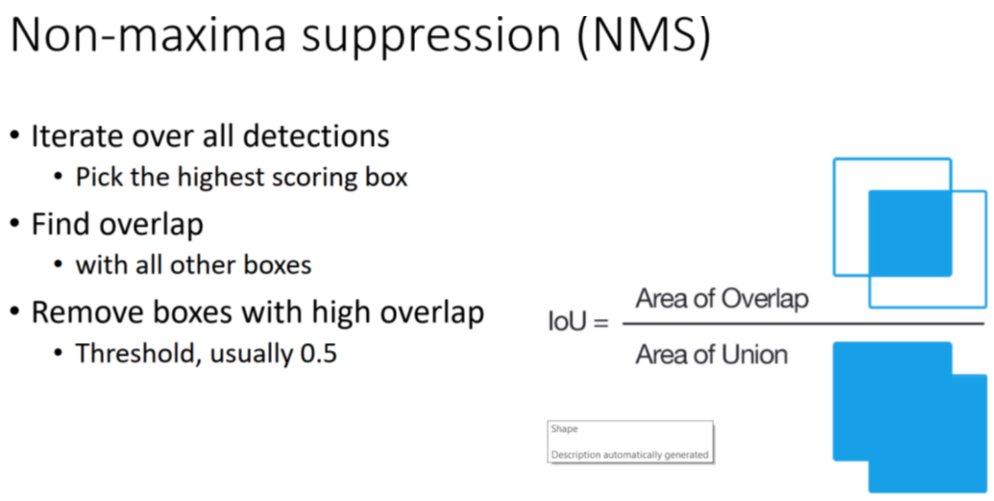




$$M(x,y) = \begin{cases} |\nabla S|(x,y) & \text{if } |\nabla S|(x,y) > |\Delta S|(x',y') \\ & \& \, |\Delta S|(x,y) > |\Delta S|(x'',y'') \\ 0 & \text{otherwise} \end{cases}$$

**x**' and **x**" are the neighbors of **x** along normal direction to an edge

Post processing: - duplicate detection avoid using non max suppression.



## Non-maxima suppression (NMS)

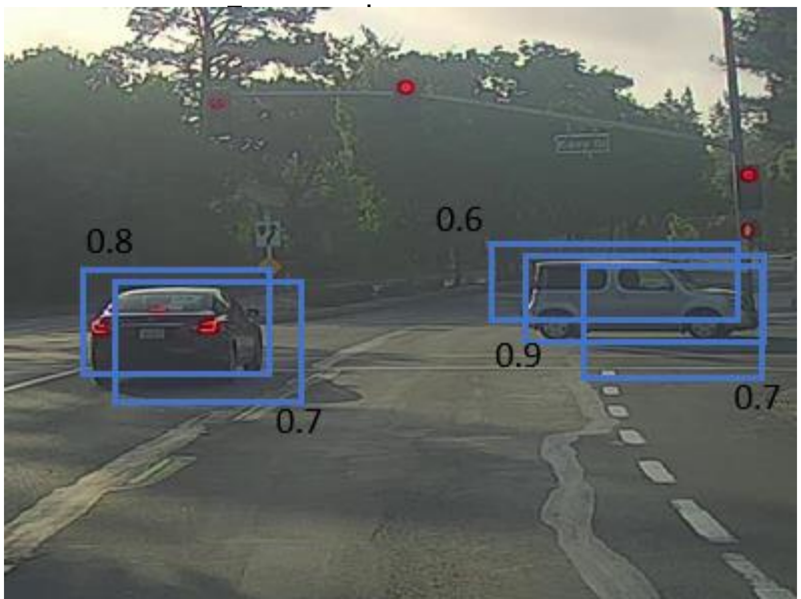
- Iterate over all detections
  - Pick the highest scoring box
- Find overlap
  - with all other boxes
- Remove boxes with high overlap
  - Threshold, usually 0.5

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Shape

Description automatically generated

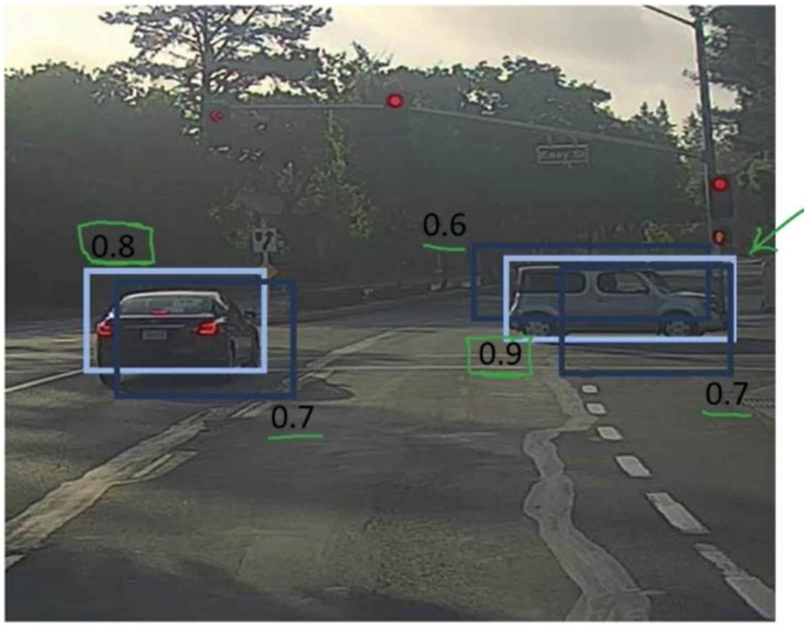
- One of the problems we have addressed in YOLO is that **it can detect an object multiple times**
- Non-max Suppression is a way to make sure that YOLO detects the object just once.



i) Discard Box with low Pc

ii) For a specific object class
- Output 0.9 box,
- discard overlapping boxes 0.6 and 0.7, as have IOU >=0.5
- Output 0.8 box

- discard overlapping box 0.7, as have IOU>=0.
- We left with 2 boxes



- Non-max suppression algorithm:
  - Lets assume that we are targeting **one class as an output class**.
  - Y shape should be [Pc, bx, by, bh, hw] Where Pc is the probability if that object occurs.
  - **Discard** all boxes with **Pc <= 0.6 (from prediction volume)**
  - While there are any remaining boxes:
    a. Pick the box with the **largest Pc** Output that as a prediction.
    b. **Discard** any remaining box with **IoU > 0.5** with that box output in the previous step i.e any box with high overlap (greater than overlap threshold of 0.5).
- If there are **multiple classes/object types** c you want to detect, you should run the Non-max suppression **c** times, once for every output class.