BST 261: Data Science II
Lecture 8

Object Localization and Detection, Face Recognition

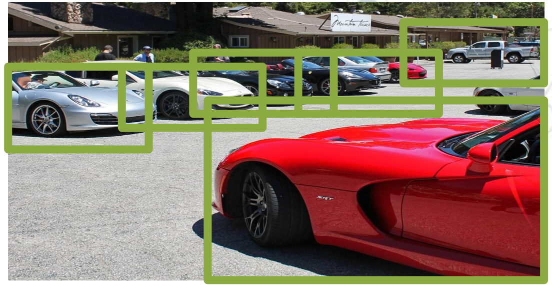
Heather Mattie
Harvard T.H. Chan School of Public Health
Spring 2 2021

Recipe of the Day!

[Turkey Pot Pie]

Object Detection and Location

Localization and Detection



Car

(Classification)

Car, but where?

(Classification with localization)

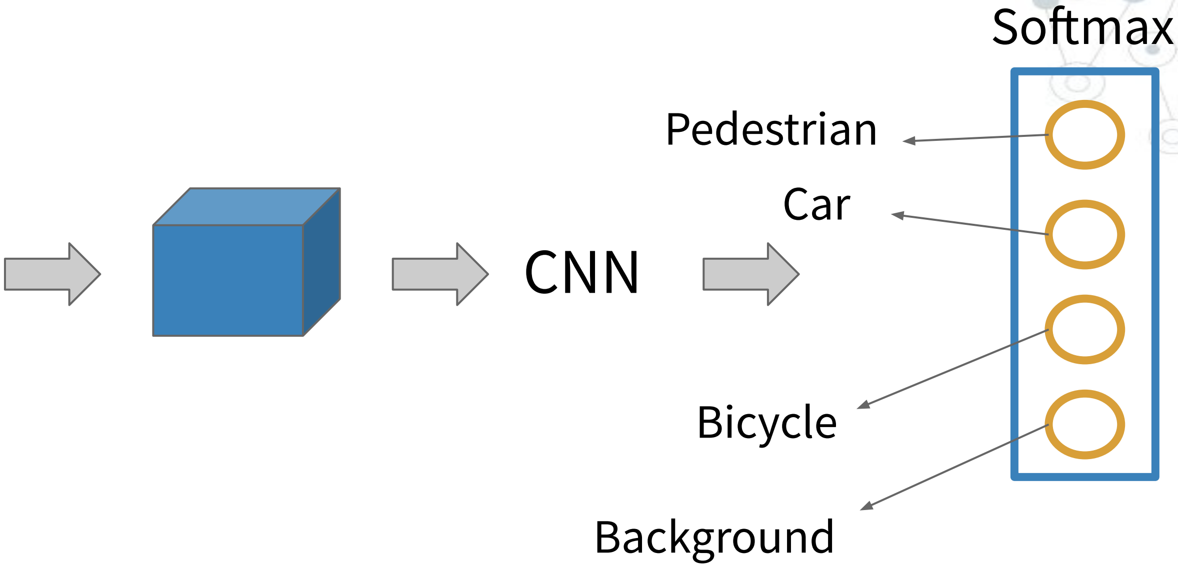
Multiple cars

(Detection)

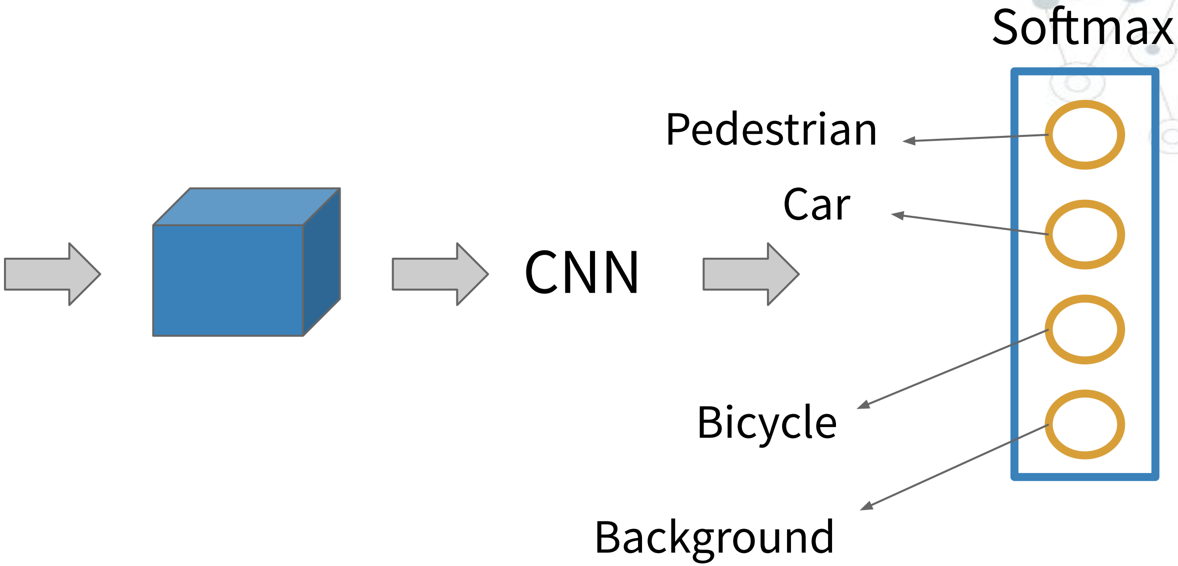
1 object

Multiple objects; could be from different classes

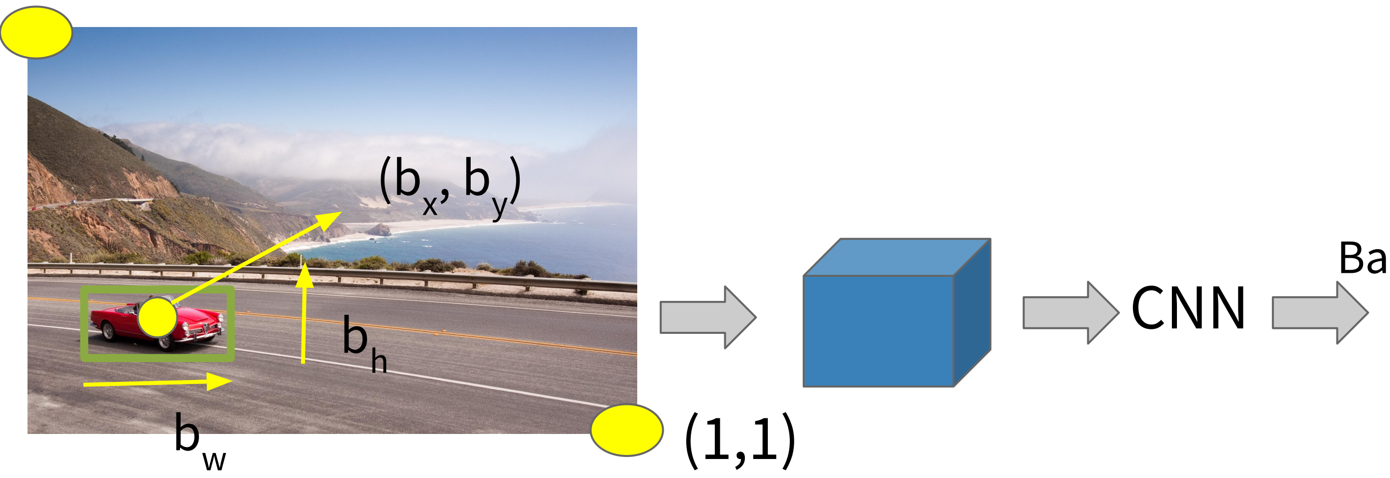
Classification

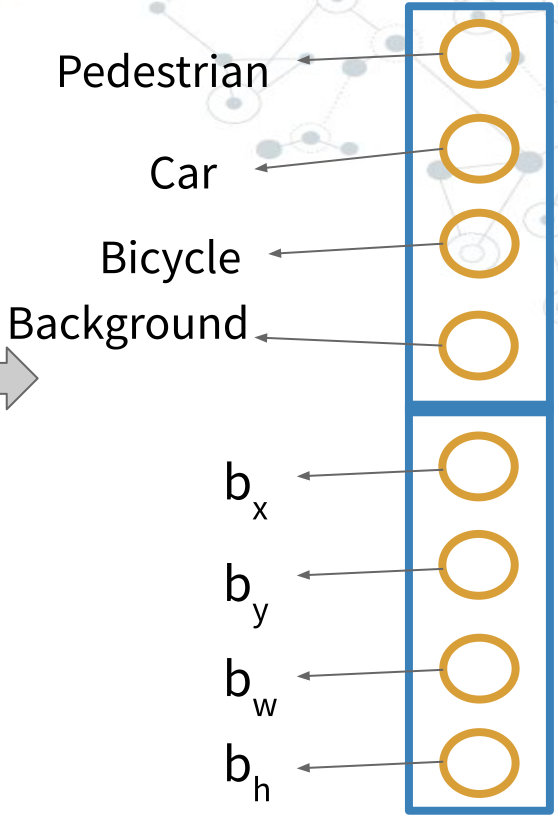


Classification



Classification with Localization





- To train a network to detect and locate objects,

  we need a lot of training data with bounding box labels:

  - (b_x, b_y): the x and y coordinates of the center of the object
  - b_w: the width of the bounding box
  - b_h: the height of the bounding box
  - New image label: [class, b_x, b_y, b_w, b_h]

Classification with Localization

<u>Classes</u>

Pedestrian (c_1)

Car (c_2)

Bicycle (c_3)

Background (no object)

New y label: [p_d, b_x, b_y, b_w, b_h, c_1, c_2, c_3]

Loss:

$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2 + (\hat{y}_2, y_2)^2 \ldots (\hat{y}_8, y_8)^2 \quad \text{if } p_d = 1$$

$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2 \quad \text{if } p_d = 0$$

Note: assuming there is only 1 object in image

Probability there is an object in the image (and not just background)

Localization and Detection

<u>Classes</u>

Pedestrian (c_1)

Car (c_2)

Bicycle (c_3)

Background (no object)

$y = [1, 0.25, 0.75, 0.2, 0.15, 0, 1, 0]$

New y label: $[p_d, b_x, b_y, b_w, b_h, c_1, c_2, c_3]$

Loss:

$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2 + (\hat{y}_2, y_2)^2 \ldots (\hat{y}_8, y_8)^2 \qquad \text{if } p_d = 1$$

$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2 \qquad \text{if } p_d = 0$$

Localization and Detection

<u>Classes</u>

Pedestrian (c_1)

Car (c_2)

Bicycle (c_3)

Background (no object)

y = [0, ?, ?, ?, ?, ?, ?, ?]

New y label: [p_d, b_x, b_y, b_w, b_h, c_1, c_2, c_3]

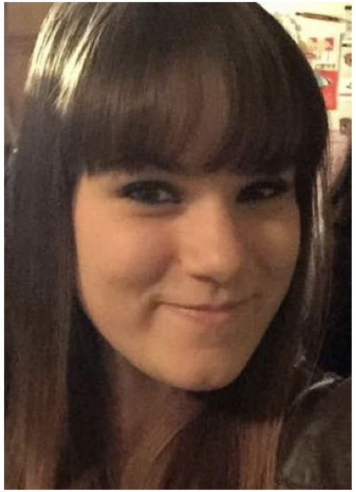
Loss:

$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2 + (\hat{y}_2, y_2)^2 \ldots (\hat{y}_8, y_8)^2 \qquad \text{if } p_d = 1$$

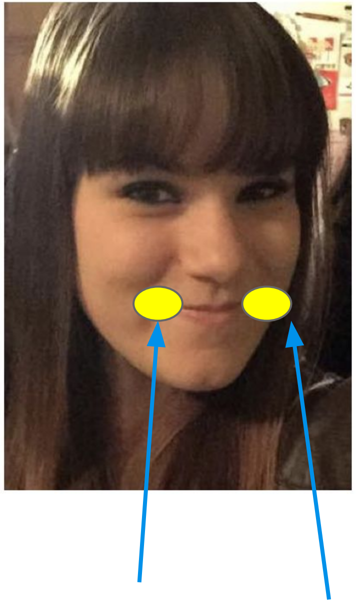
$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2 \qquad \text{if } p_d = 0$$

Landmark Detection

Landmark Detection

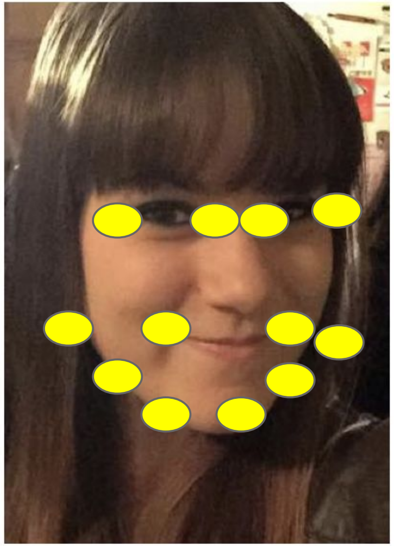


Landmark Detection



(l_{x1}, l_{y1}) (l_{x2}, l_{y2})

Landmark Detection



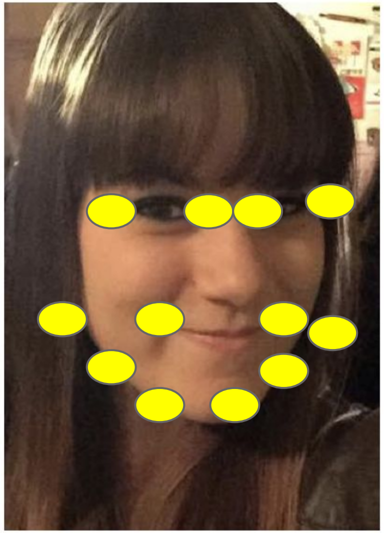
Label every point

Input: image with n landmarks
Output:
$[p_{face}, l_{x1}, l_{y1}, l_{x2}, l_{y2}, \dots, l_{xn}, l_{yn}]$

Fit CNN to output if the image is of a face and the locations of the landmarks if it is a face

Note: landmarks have to be consistent across all training images, i.e. landmark # 1 is the left corner of the right eye, for example

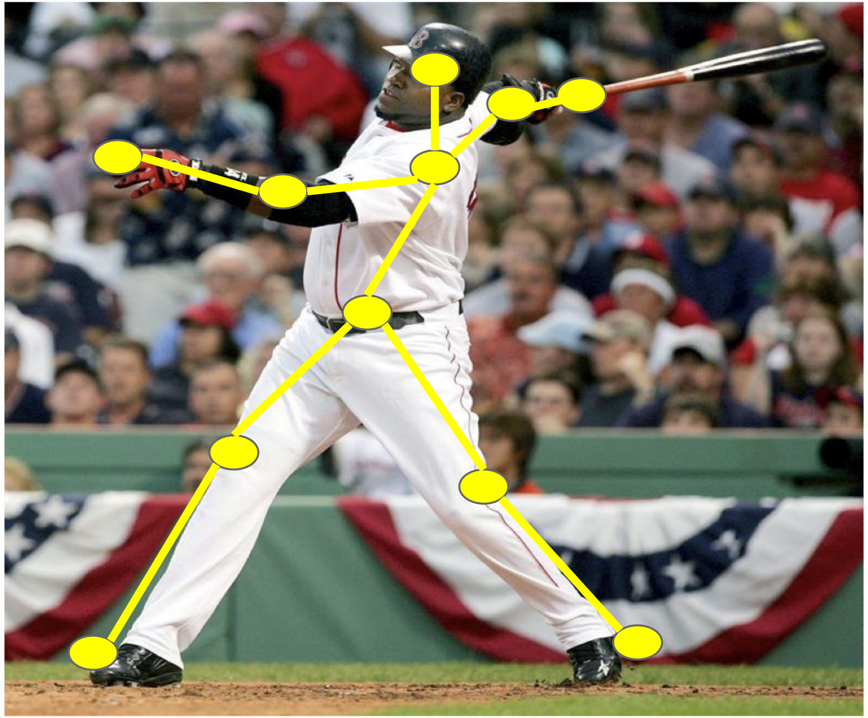
Landmark Detection



If I can detect where the landmarks are, I can add filters in appropriate places

Landmark Detection

Landmark Detection



“Pose” detection

Object Detection

Object Detection

- Goal: locate and classify objects in an image
- Train CNN on cropped images of objects, where the object takes up most of the space in the image

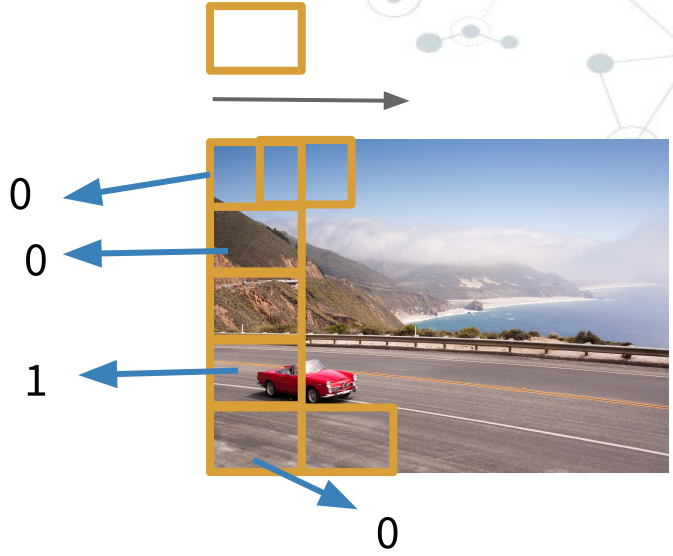
X: image of a car
y: 1

X: image of a car
y: 1

X: image of not a car
y: 0

Object Detection

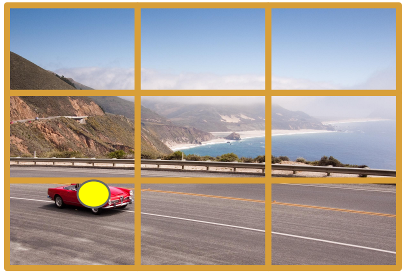
- Sliding windows detection algorithm
  - Slide a window across your image
  - In each region covered by the window, try to detect object (classify every region as containing an object or not)
  - **Very computationally expensive**, especially for small window and small stride
  - Bigger windows or strides result in fewer regions and less computational expense, but could hurt performance
  - Won't output the most accurate bounding boxes



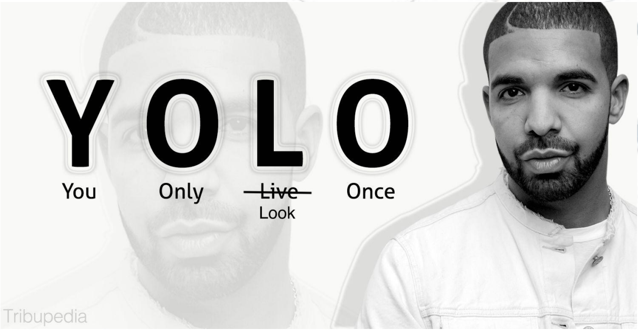
Repeat with larger and larger windows

Object Detection

- One way to predict more accurate bounding boxes is by implementing the YOLO (You Only Look Once) algorithm
  - Redmon et al. 2015
  - Overly dramatic YOLO video
- Split image into grid cells
- Assign the object to the grid cell containing the midpoint of the object
- Works well when there is only 1 object in a particular cell
- Cuts down on computational cost because it can be run as a single convolutional implementation
  - So fast it performs well for real time object detection
- GitHub repo with easy implementation in Keras

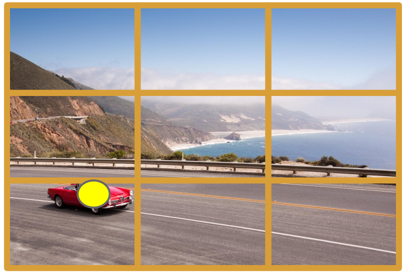


Object Detection

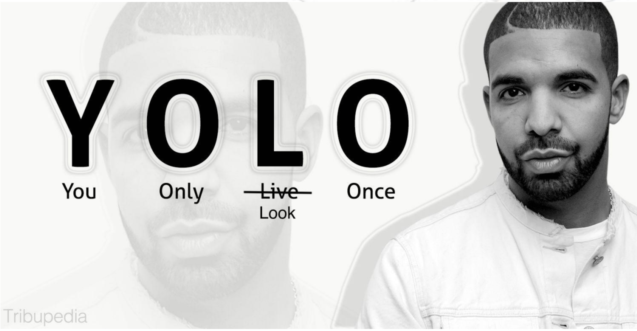


- One way to cut down on computational expense and predict more accurate bounding boxes is by implementing the YOLO (You Only Look Once) algorithm
  - Redmon et al. 2015
  - Overly dramatic YOLO video
- Split image into grid cells
- Assign the object to the grid cell containing the midpoint of the object
- Works well when there is only 1 object in a particular cell
- Cuts down on computational cost because it can be run as a single convolutional implementation
  - So fast it performs well for real time object detection
- GitHub repo with easy implementation in Keras

$$y = [0, ?, ?, ?, ?, ?, ?, ?]$$

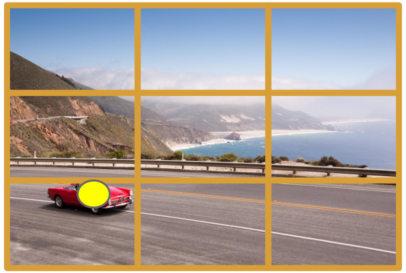


Object Detection



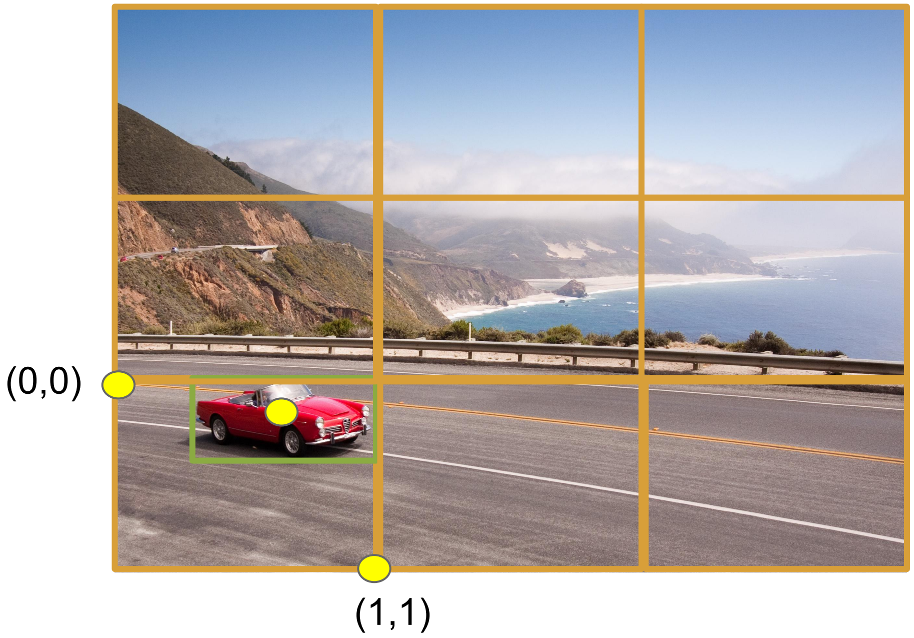
- One way to cut down on computational expense and predict more accurate bounding boxes is by implementing the YOLO (You Only Look Once) algorithm
  - Redmon et al. 2015
  - Overly dramatic YOLO video
- Split image into grid cells
- Assign the object to the grid cell containing the midpoint of the object
- Works well when there is only 1 object in a particular cell
- Cuts down on computational cost because it can be run as a single convolutional implementation
  - So fast it performs well for real time object detection
- GitHub repo with easy implementation in Keras

$y = [1, b_x, b_y, b_w, b_h, 0, 1, 0]$ →



Better Bounding Boxes

- b_x, b_y, b_w, b_h are defined relative to the grid cell
- b_x, b_y will be between 0 and 1 by definition
- b_w, b_h could be greater than 1, depending on how large the object is and if it spans more than the grid cell with the midpoint



Evaluating Object Localization

- How well is your algorithm working in terms of finding the bounding boxes?

Evaluating Object Localization

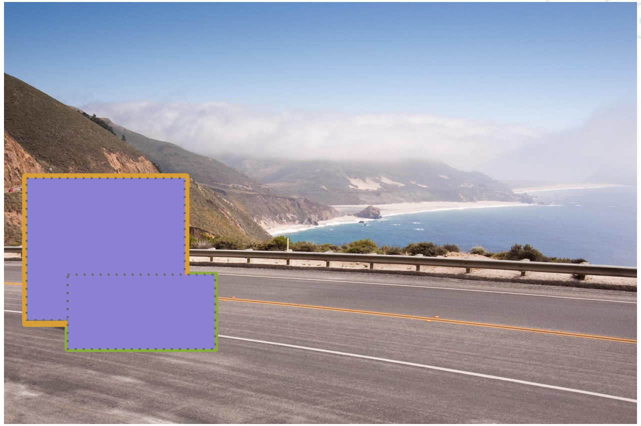
- How well is your algorithm working in terms of finding the bounding boxes?
- One metric to measure the performance of the algorithm is Intersection over Union

$$IoU = \frac{\text{size of intersection}}{\text{size of union}}$$

Evaluating Object Localization

- How well is your algorithm working in terms of finding the bounding boxes?
- One metric to measure the performance of the algorithm is Intersection over Union

$$IoU = \frac{\text{size of intersection}}{\text{size of union}}$$



Evaluating Object Localization

- How well is your algorithm working in terms of finding the bounding boxes?
- One metric to measure the performance of the algorithm is Intersection over Union

$$IoU = \frac{\text{size of intersection}}{\text{size of union}}$$

Evaluating Object Localization

- How well is your algorithm working in terms of finding the bounding boxes?
- One metric to measure the performance of the algorithm is Intersection over Union

$$IoU = \frac{\text{size of intersection}}{\text{size of union}}$$

- "Correct" if , $IoU \geq 0.5$ or some other threshold
- Basically measures the overlap of the predicted bounding box with the ground truth bounding box - more overlap is better

Non-max suppression

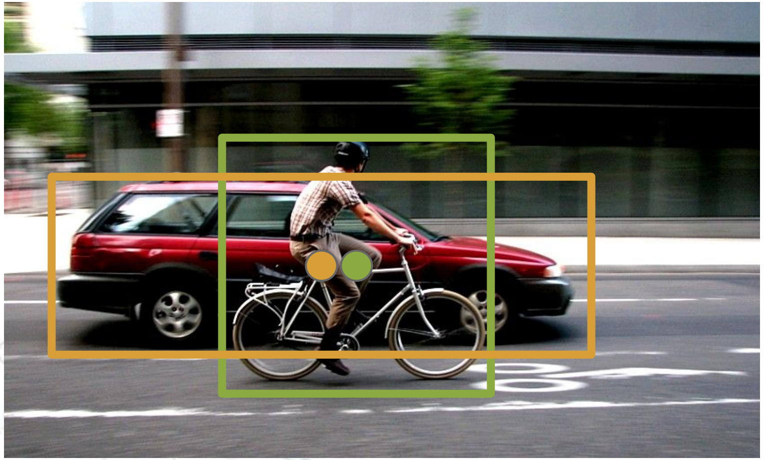
- Your algorithm may detect the same object multiple times
- Non-max suppression is a way to make sure you detect each object only once
- Steps
  - Discard all boxes with $p_d \leq 0.6$ (probability that object is detected)
  - While boxes remain: find box with highest p_d
  - Suppress (discard) all other boxes that have IoU ≥ 0.5 with the box in the previous step
  - Repeat until no more boxes remain
- Repeat this process independently for each type of object you are trying to detect

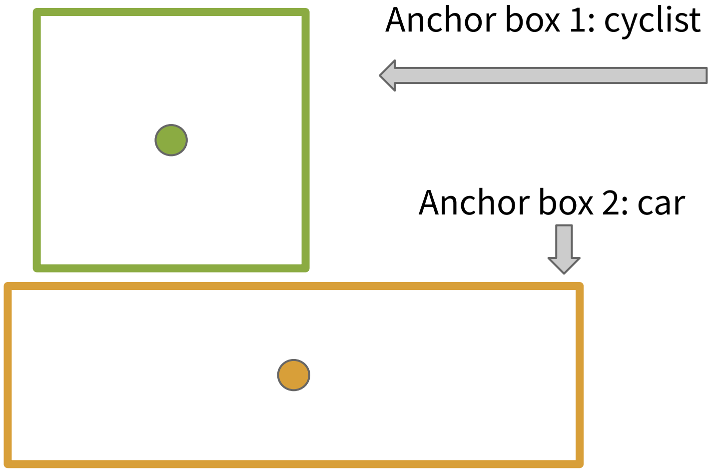
Anchor Boxes

- So far we have assumed a grid cell can only detect one object
- What if you want to **detect multiple objects in the same cell**?
- Originally, we assigned each object in an image to the grid cell that contained its midpoint
- Now, we will assign an object to a grid cell that contains its midpoint and an anchor box for that cell with the highest IoU

Anchor Boxes

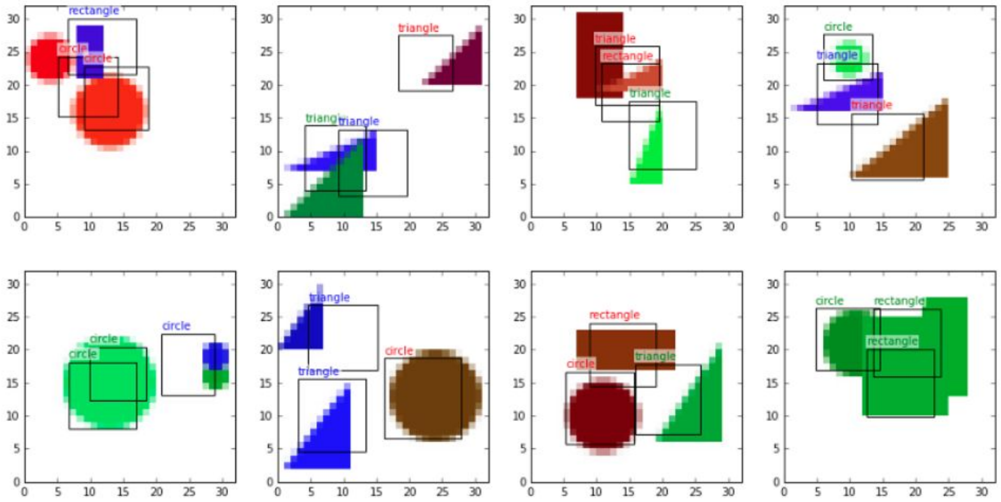
$$y = [\underbrace{p_d, b_x, b_y, b_w, b_h, c_1, c_2, c_3}_{\text{Anchor box 1}}, \underbrace{p_d, b_x, b_y, b_w, b_h, c_1, c_2, c_3}_{\text{Anchor box 2}}]$$





Object Detection Tutorial

- [Object detection with neural networks - a simple tutorial using Keras]



Face Recognition

Terminology

- Recognition
  - Have a database of K persons
  - Get an input image
  - Output ID if the image is any of the K persons, or “not recognized” if not like any of the K persons

- Verification
  - Input image and name/ID
  - Output whether the input image is that of the claimed person

- Andrew Ng demo video

Face Recognition

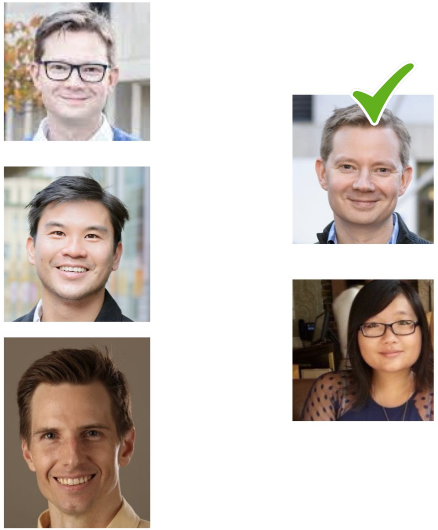
- One-shot Learning: learning from 1 example to recognize that person again
- Major downside: needs to be re-trained every time another person is added to group, only 1 example to learn from





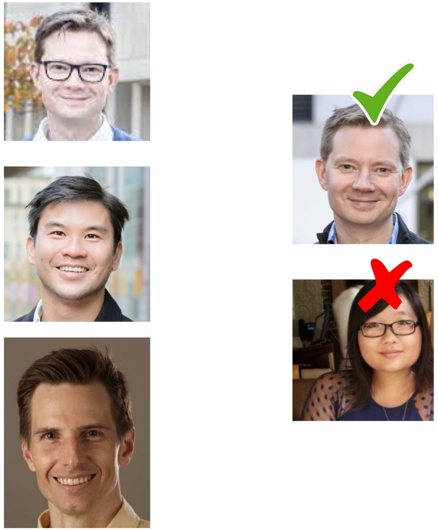
Face Recognition

- One-shot Learning: learning from 1 example to recognize that person again
- Major downside: needs to be re-trained every time another person is added to group, only 1 example to learn from



Face Recognition

- One-shot Learning: learning from 1 example to recognize that person again
- Major downside: needs to be re-trained every time another person is added to group, only 1 example to learn from



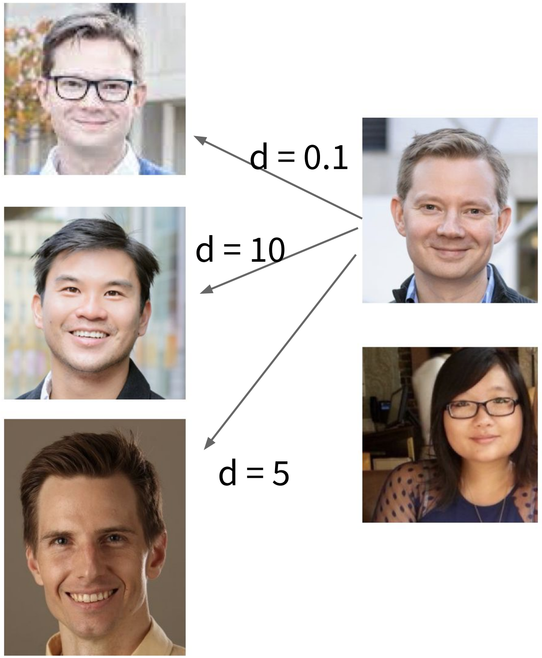
Similarity Function

- Similarity function: quantify how similar or different two images are
- If difference is large, the images are of two different people
- If difference is small, the images are of the same person
- DeepFace by Taigman et al 2014
- Define the similarity network as

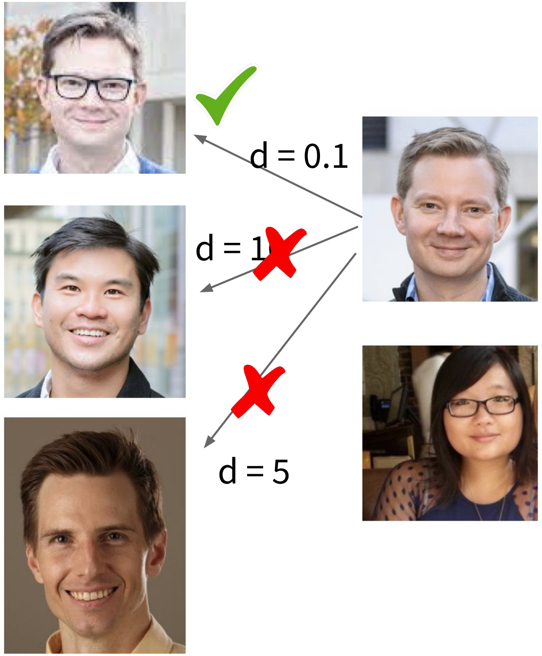
$$d(x^{(1)}, x^{(2)}) = \|f(x^{(1)}) - f(x^{(2)})\|_2^2$$

- Learn parameters such that if $x^{(i)}$ and $x^{(j)}$ are the same person, d is small, and if $x^{(i)}$ and $x^{(j)}$ are different people, d is large
- Come up with threshold of what is “small”

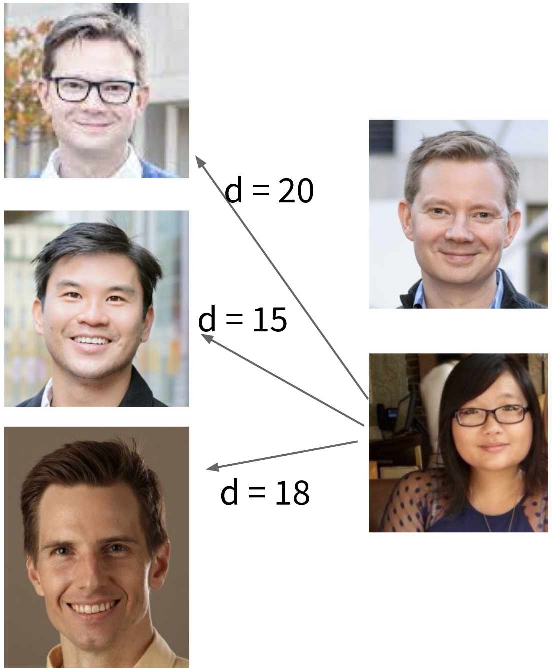
Similarity Function



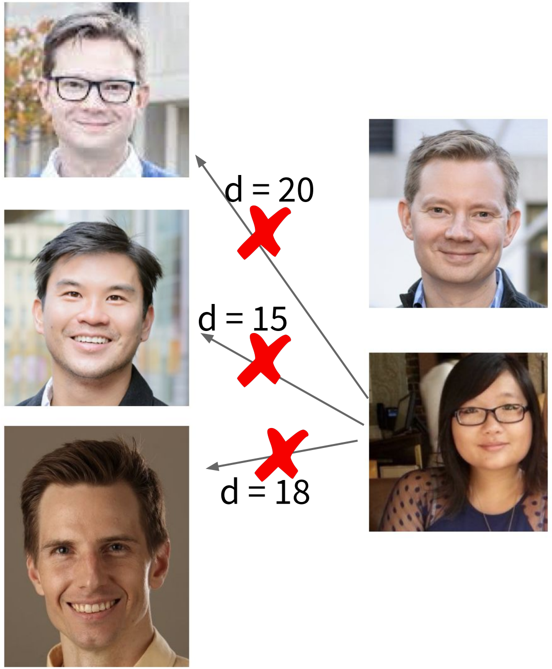
Similarity Function



Similarity Function



Similarity Function



Not in database

Triplet Loss

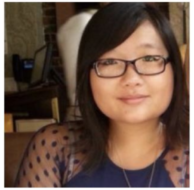
- To learn the parameters of your network (get good encodings for images), can use gradient descent to minimize the triplet loss
- Given 3 images A (anchor), P (positive) and N (negative), can we minimize the "triplet loss":

$$L(A, P, N) = \max(\|f(A) - f(P)\|_2^2 - \|f(A) - f(N)\|_2^2 + \alpha, 0)$$

- Note that multiple pictures of each person are needed for this to be effective

Anchor
(A)

Positive
(P)



Anchor
(A)

Negative
(N)

FaceNet

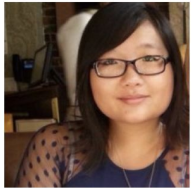
Margin parameter - ensures the network doesn't just label every difference as 0

$$d(A, P) + \alpha \leq d(A, N)$$

- During training, if A, P, and N are chosen randomly, is easily satisfied
  - It's really easy to randomly pick two very different looking people (if your sample is heterogeneous)
  - It's better to choose A, P, and N such that training is more difficult and will be better at recognizing differences on test sets

Anchor (A)

Positive (P)



Anchor (A)

Negative (N)