NUS-ISS
Vision Systems

Module 5 - Building vision system using machine learning (1) - Detection and recognition, part 2

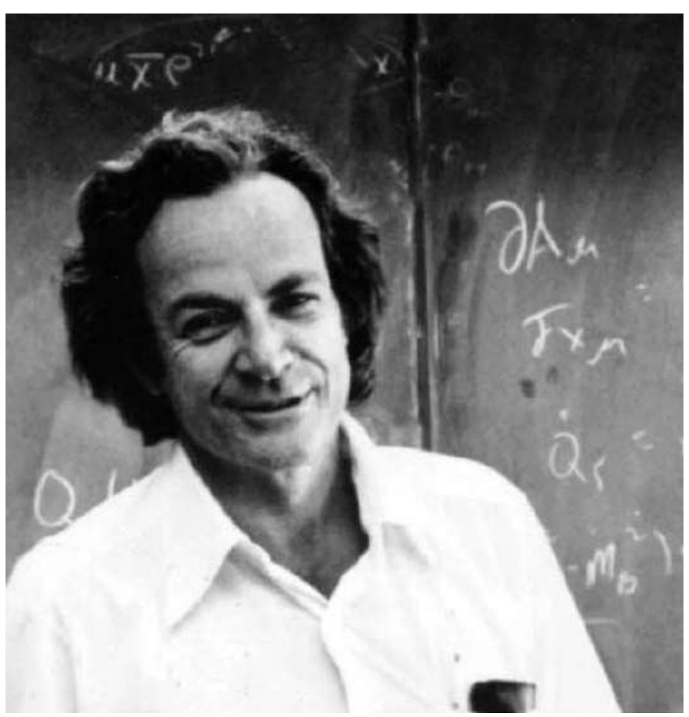
Dr. Tan Jen Hong
Lecturer & Consultant
Institute of System Science
National University of Singapore
issjht@nus.edu.sg

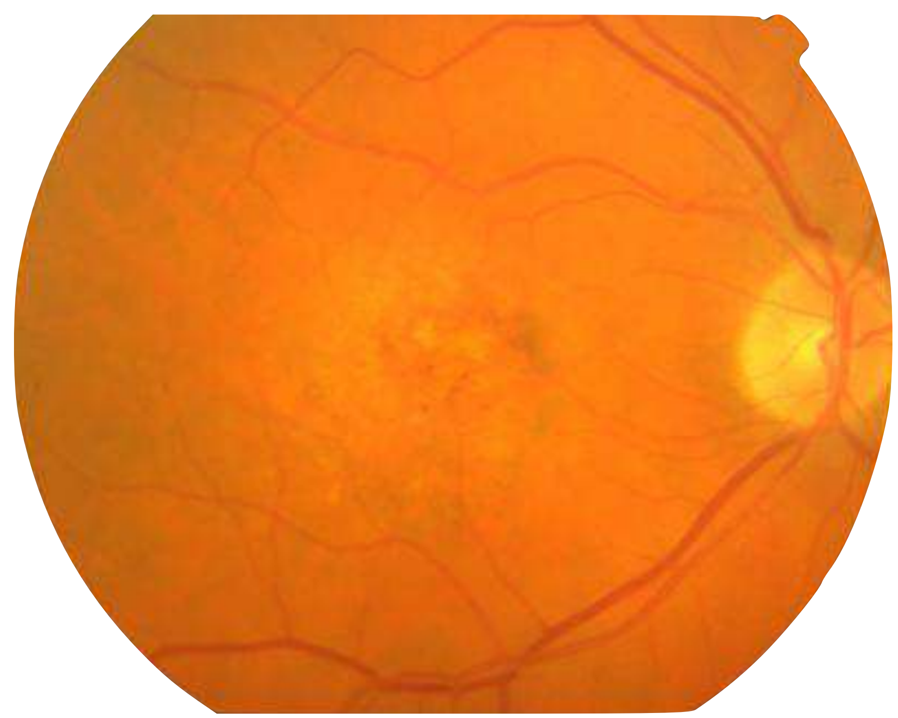
Learning objectives

- Understand the difference between image classification and object detection
- Understand the major challenges in object detection
- Perform object detection using YOLO v3

Image classification

- Algorithm looks at image and classifies the object within
- Meant for single item identification or cateogry classification
- Wide application, from face detection on social network to medical diagnosis in healthcare





Object classification and localization

- Tell not just if something exists in an image, but where it situates

- Draw bounding box and label

Source: https://mangsh.co/2014/09/16/walking-waiting/

Object classification and localization

Before the deep learning era

- Before we introduce the deep learning approach to object classification and localization, let's see how this was done in the past
- Goal: Locate carplate in an image
- Using only the concepts we have covered in these 3 days

Before

After

Locate carplate
preprocessing

- Colour image (which is 3D) has rich information, but 2D array is easier for many image processing techniques

```
> car      = cv2.imread('carplate01.png')
> cargray  = cv2.cvtColor(car, cv2.COLOR_BGR2GRAY)
```

Before

After

Locate carplate
Apply thresholding

- Use Otsu thresholding to separate foreground and background in the scene

```
> lightest  = cv2.threshold(cargray,
                            0,
                            255,
                            cv2.THRESH_OTSU)[1]
```

in Otsu thresholding, we set 0 for threshold value

assgined value

There are 2 outputs from Otsu thresholding, we are only interested in the second output

Before

After

Locate carplate
Edge detection

- After thresholding, we need to extract the salient features in the image, this is done through edge detection

```
> edges = cv2.Canny(lightest,
            minVal  31,
            maxVal  127,
  Sobel kernel size  apertureSize=3)
```

Before

After

Locate carplate

Fuse characters on carplate

- To identify the carplate, we need those characters on the carplate to be fused as a cohesive block, to do that, we need to dilate and erode the borders of the characters

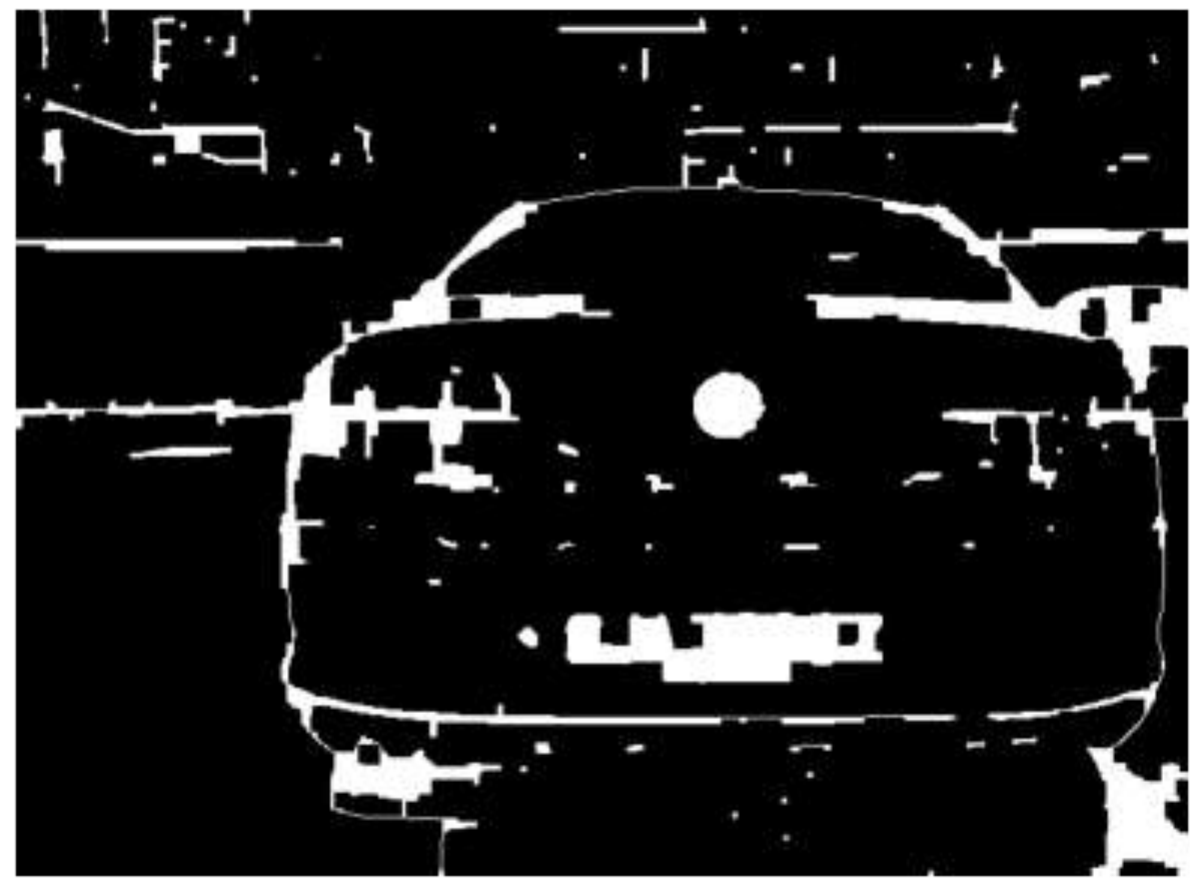
```
> bulk  = cv2.dilate(edges,None, iterations=4)
> bulk  = cv2.erode(bulk,  None, iterations=4)
```

Since we just use the default kernel (size 3 x 3), and thus we set 'None' for kernel

Each operation is run 4 times, hence the value 4 for 'iterations'

Before

After



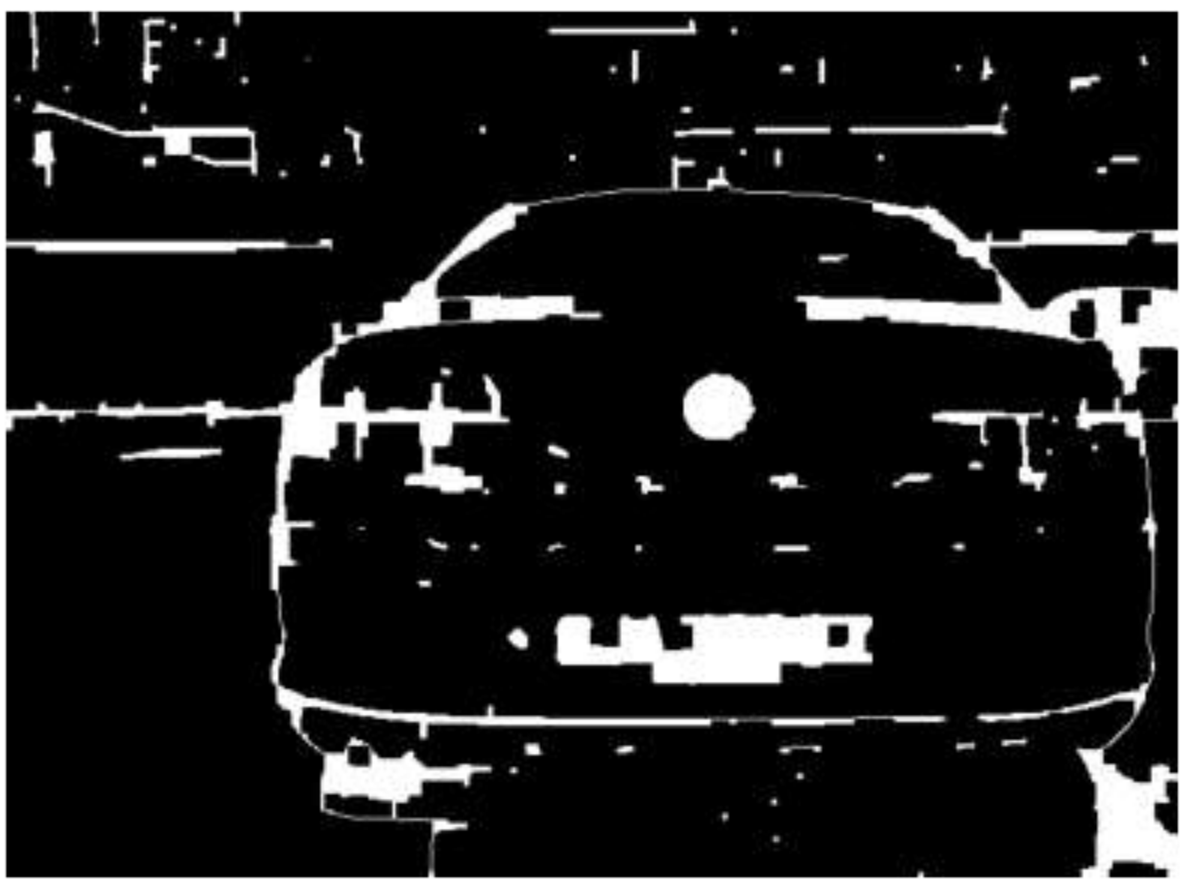
Locate carplate
Remove background noise

- To remove background noise, we apply erode followed by dilate 2 times

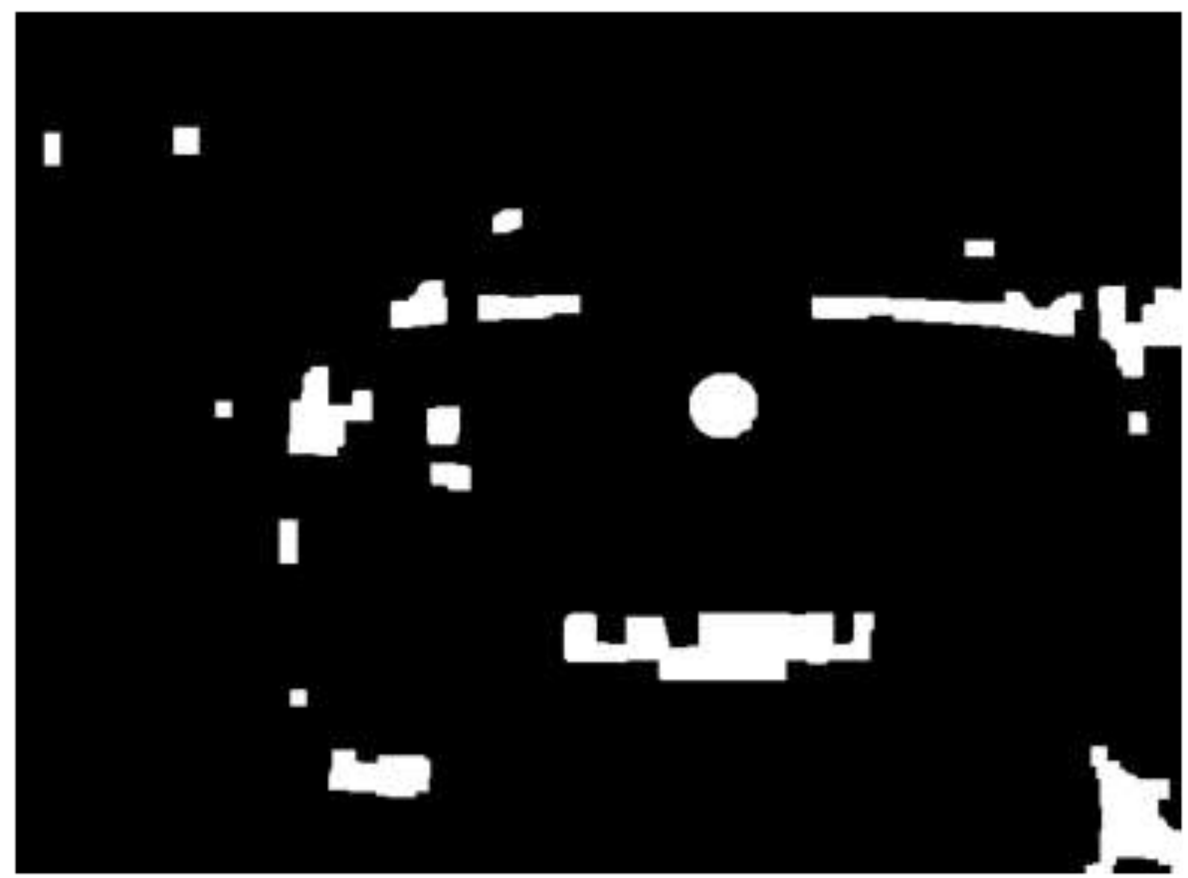
```
> bulk  = cv2.erode(bulk,  None, iterations=2)
> bulk  = cv2.dilate(bulk, None, iterations=2)

> bulk  = cv2.erode(bulk,  None, iterations=4)
> bulk  = cv2.dilate(bulk, None, iterations=4)
```

Before



After



Locate carplate

Get the contours

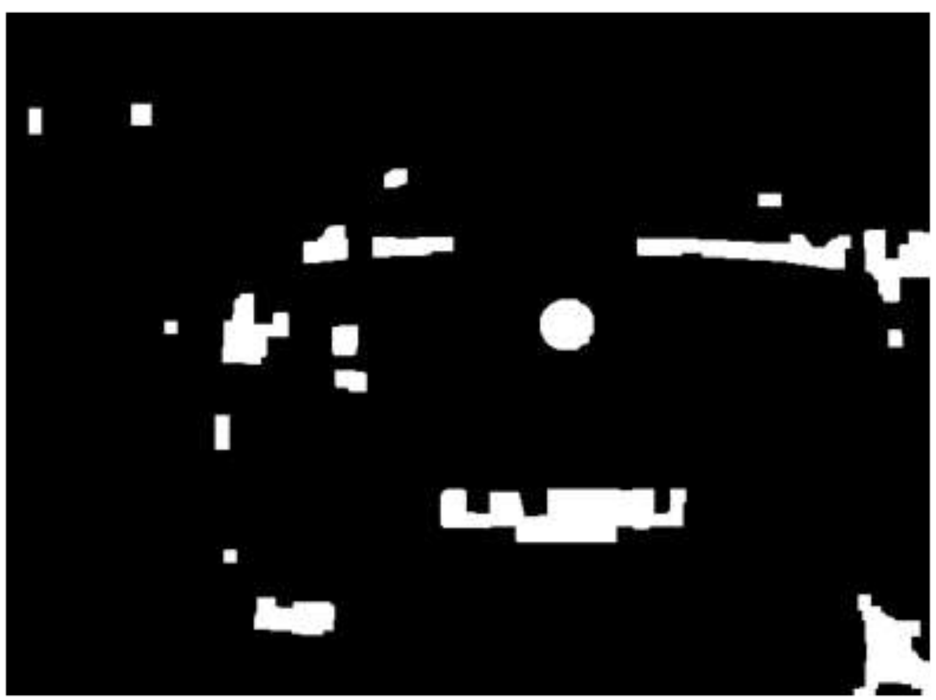
- Get some of of the largest contours in the background-remove image, and plot them out for inspection

Get all the contours in the image

```
> ctrs = cv2.findContours(bulk.copy(),
                          cv2.RETR_EXTERNAL,
                          cv2.CHAIN_APPROX_SIMPLE)
> ctrs = ctrs[0]
```

Sort the contours based on contour area, keep only the first 5 largest contour

```
> ctrs = sorted(ctrs,
                key=cv2.contourArea,
                reverse=True)[:5]
```



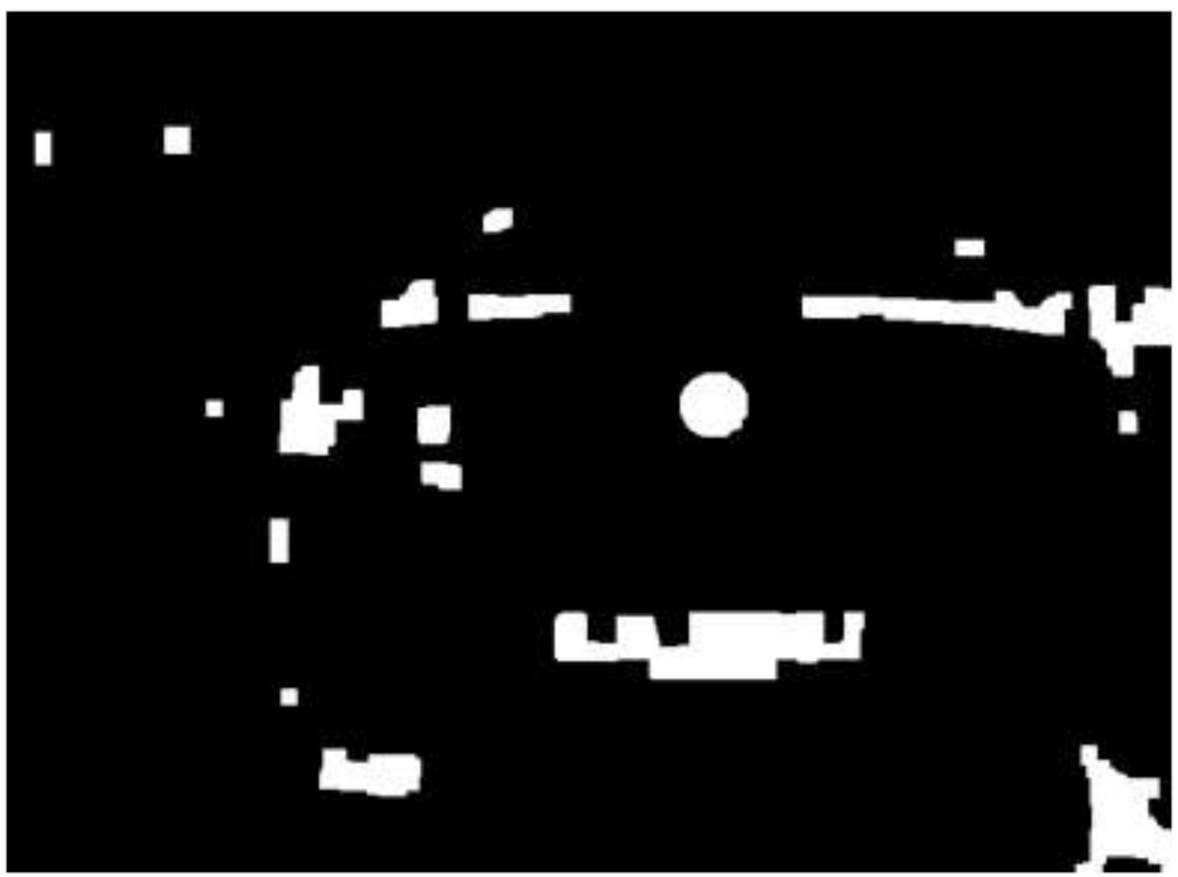
```
> cargrayrgb = cv2.merge((cargray,cargray,cargray))
> cv2.drawContours(cargrayrgb,
                   ctrs,
                   -1,
                   (0,0,255),
                   2)
```

Merge grayscale image into a 3D array, draw contours on the 3D array

Locate carplate
Get the contours

- Only the first 5 largest-area contours are highlighted and kept in the analysis

Before



After

Locate carplate

Get the carplate

- Apply bounding rectangle on each of the contour and determine the carplate by aspect ratio

Apply bounding rectangle and calculate the aspect ratio for each contours

If the bounding box has an aspect ration between 4 and 5, it is considered the carplate

The carplate region is extracted and applied Otsu thresholding to get the car numbers

```
> for c in ctrs:
    (x, y, w, h)  = cv2.boundingRect(c)
    aspr          = w/float(h)

    if aspr >= 4 and aspr <= 5:
      carPlate = cargray[y:y+h, x:x+w]
      carPlate = cv2.threshold(carPlate,
                               0,
                               255,
                               cv2.THRESH_OTSU)[1]
      break
```


```
> cv2.rectangle(car,
                (x,y),
                (x+w,y+h),
                (0,0,255),
                2)
```

Draw the carplate bounding box on the colour image

Locate carplate
Get the contours

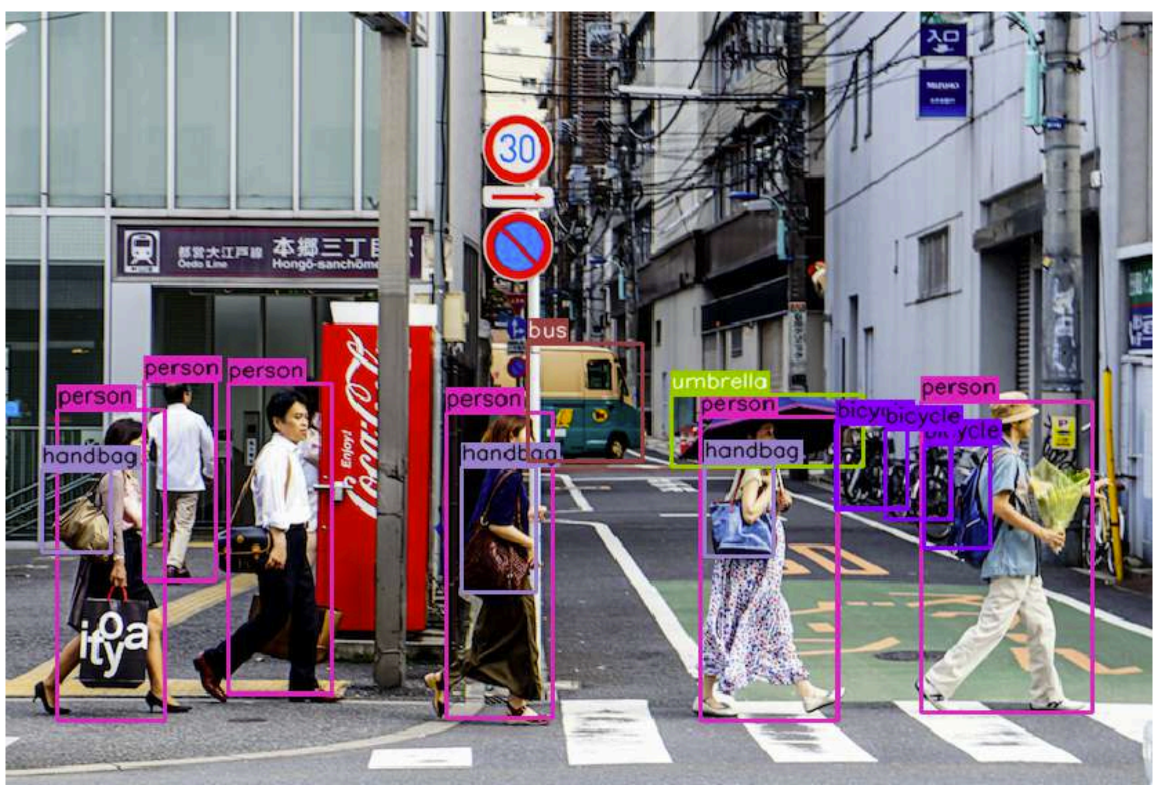
- The located carplate and the extracted region

- We can further send the extracted region to OCR library to read the characters

Before

After

Multiple objects detection



Source: https://mangsh.co/2014/09/16/walking-waiting/

- We have seen how in the past objects are located. But in the previous example, we are only locating one type of object

- Locate multiple objects (various categories) in an image is much more challenging

- Thus nowadays deep learning is the main solution to locate multiple objects in a scene

- Well known application: self-driving car; need to detect cars but also pedestrians

Why

- Why in deep learning can't we locate objects just sliding a box around image and performing classification on each extracted portion?

Primary object detectors

- Faster R-CNN, by Girshick et al. 2015; https://arxiv.org/abs/1506.01497
- Single shot detectors (SSD), by Liu et al., 2015; https://arxiv.org/abs/1512.02325
- YOLO v3, by Redmon and Farhadi, 2018; https://arxiv.org/abs/1804.02767
- RetinaNet, by Lin et al., 2017; https://arxiv.org/abs/1708.02002

YOLO v3

- YOLO stands for "you only look once", a one-stage detector
- One-stage detectors are faster, though less accurate compared to two-stage detector
- It is so fast to be used for tasks that require real-time response

Source: https://pythonawesome.com/yolov3-training-and-inference-in-pytorch/

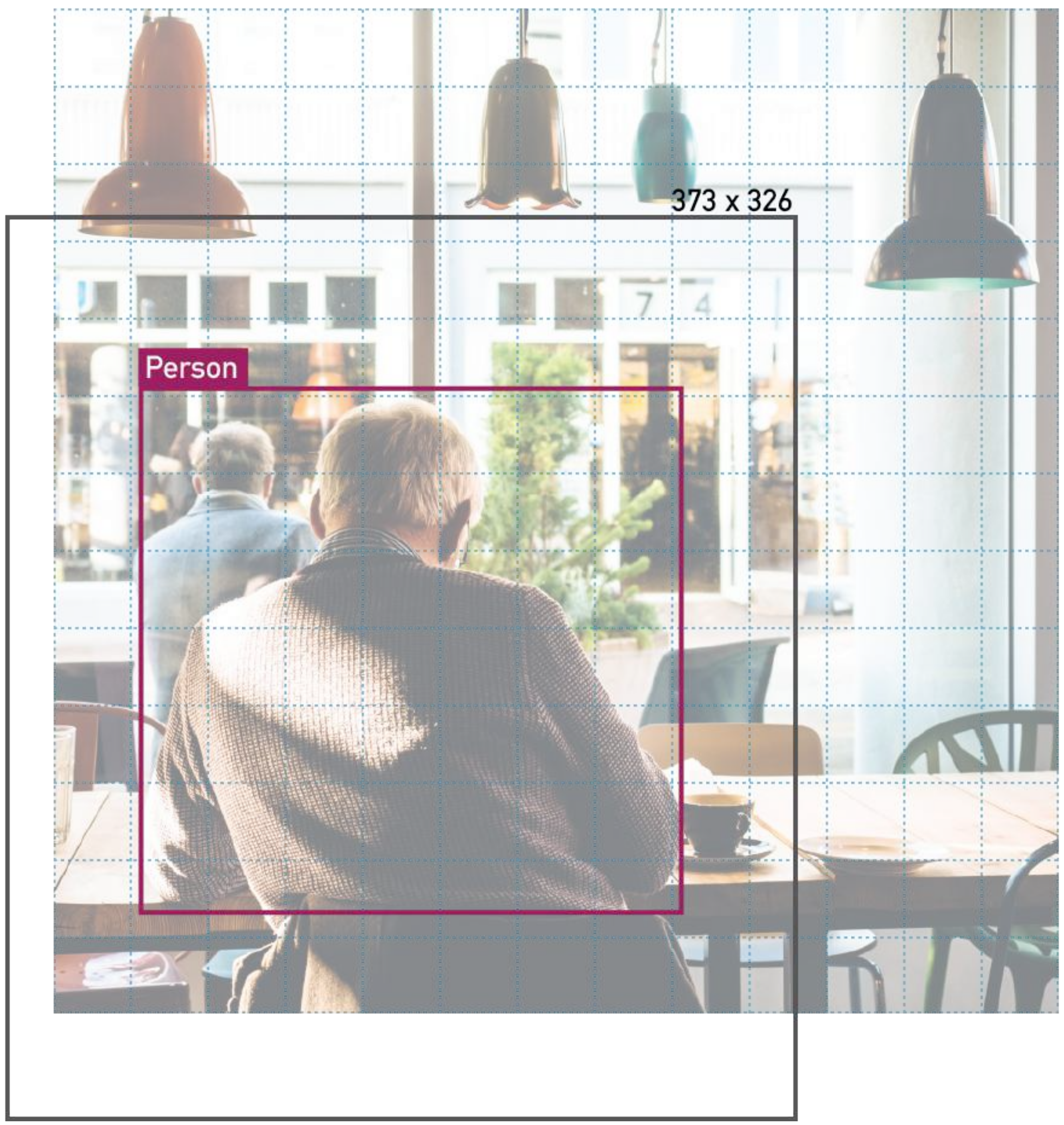
YOLO v3

- YOLO does detection at 3 stages / scales

YOLO v3

- At first stage, YOLO divides image into 13 x 13 cells
- In this stage YOLO tries to detect big objects in the image
- For each cell, the algorithm does object detection with three anchor boxes / prior boxes
- In total there are 13 x 13 x 3 = 507 anchor boxes in this stage

YOLO v3



- For each anchor box, YOLO locates the object and return the bounding box that encloses the object
- It also makes prediction on the class of the object within the bounding box

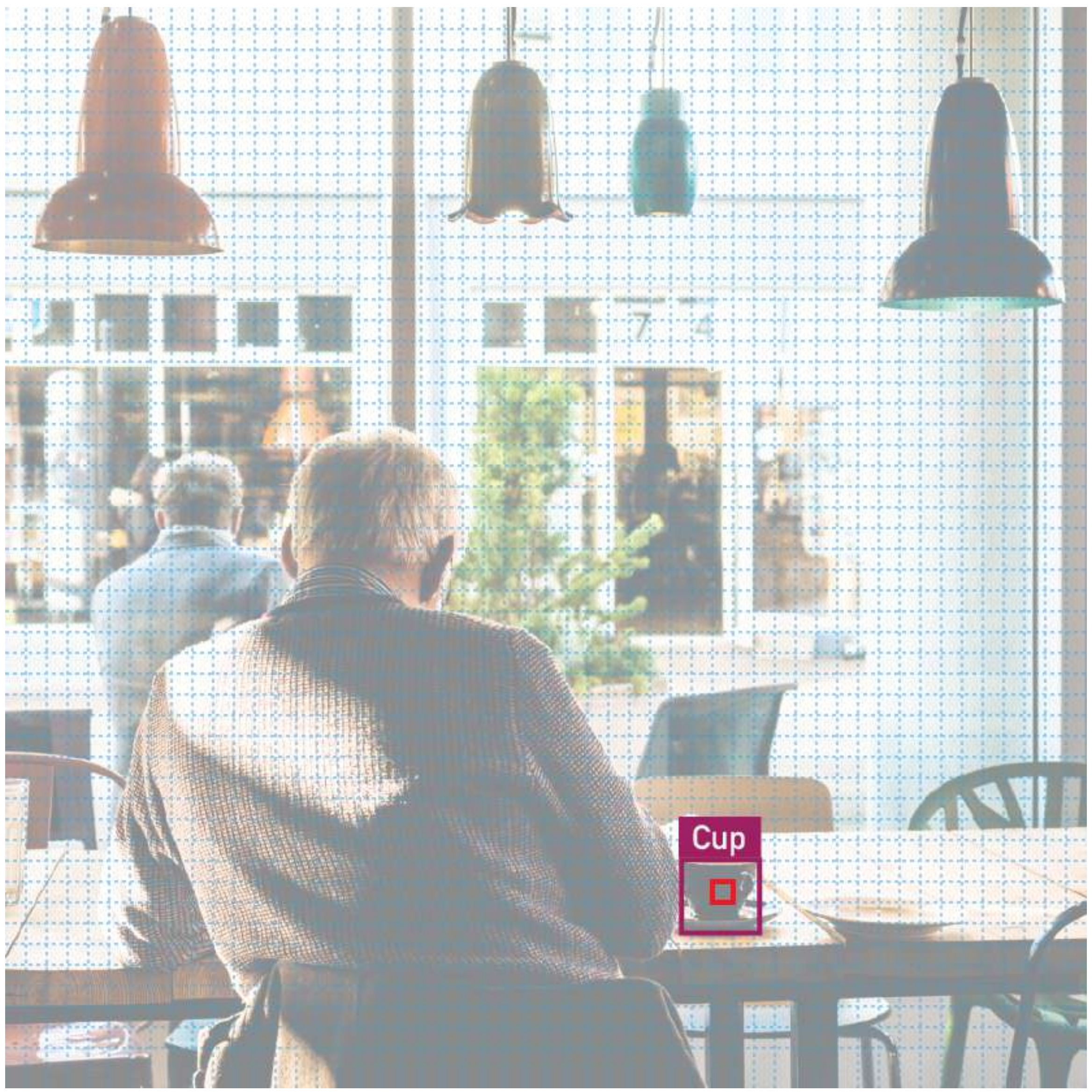
YOLO v3

- At second stage, YOLO divides image into 26 x 26 cells
- In this stage YOLO tries to detect medium objects in the image
- For each cell, the algorithm does object detection with another three anchor boxes / prior boxes
- In total there are 26 x 26 x 3 = 2028 anchor boxes in this stage

YOLO v3

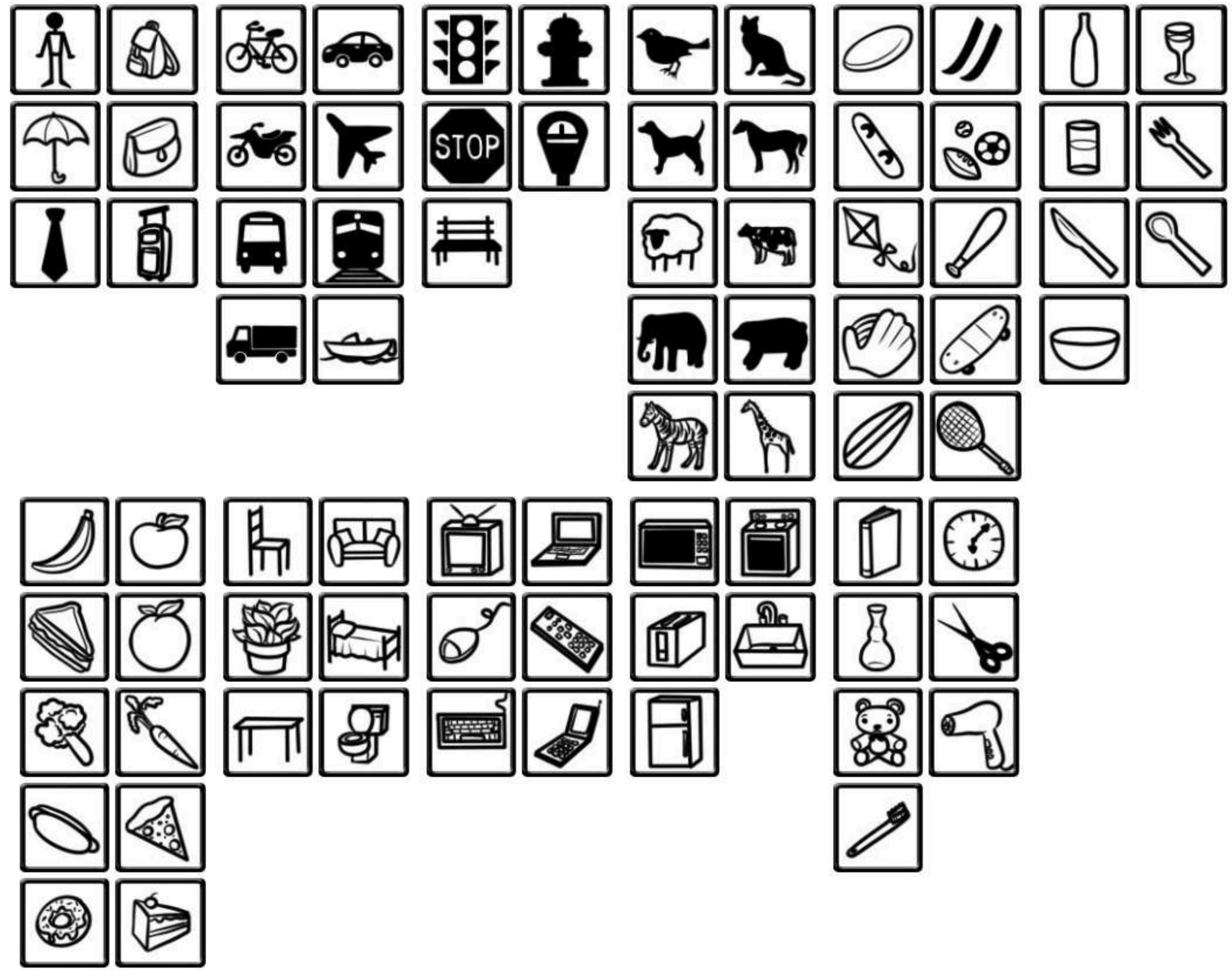
- For each anchor box, YOLO locates the object and display the result with bounding box
- It then makes prediction on the class of the object within the bounding box

YOLO v3



- At third stage, YOLO divides image into 52 x 52 cells
- In this stage YOLO tries to detect small objects in the image
- For each cell, the algorithm does object detection with three small anchor boxes / prior boxes
- In total there are 52 x 52 x 3 = 8112 anchor boxes in this stage
- (Note: The anchor boxes are too small to be illustrated properly in the left image

COCO dataset



Source: http://cocodataset.org

In total:
123,287 images; 886,284 instances
80 categories

YOLO Implementation

- Load the label file

```
> import cv2
> import numpy as np
> import matplotlib.pyplot as plt

> lbl_file = 'yolov3.txt'
> classes  = open(lbl_file).read().strip().split("\n")
```

- Load the net model

```
> yoloconfig = 'yolov3.cfg'
> yoloweights= 'yolov3.weights'
> net        = cv2.dnn.readNet(yoloweights,
                               yoloconfig)
```

yolov3.txt

```
person
bicycle
car
motorcycle
airplane
bus
train
truck
boat
traffic light
fire hydrant
stop sign
parking meter
bench
bird
cat
dog
horse
sheep
cow
elephant
bear
zebra
......
```

yolov3.cfg

```
[net]
# Testing
batch=1
subdivisions=1
# Training
# batch=64
# subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1
.....
```

YOLO Implementation

- Read image and create blob

```
> img   = cv2.imread('ms3.jpg')
> blob  = cv2.dnn.blobFromImage(image=img,
        scaling for image values (not image size)   scalefactor=1/255,
                                      output size   size=(416, 416),
       mean value to be subtracted from each channel  mean=(0,0,0),
                               swap R and B channel   swapRB=True,
                    Do cropping after resizing image   crop=False)
```

- Store the image height and width

```
> imgHeight = img.shape[0]
> imgWidth  = img.shape[1]
```

Source: https://mangsh.co/2014/11/09/of-street-01/

YOLO Implementation

- Create a function to get output layers since there are 3 output layers, each layer generates output at a particular scale

```
> def getOutputLayers(net):
      layers    = net.getLayerNames()
      outLayers = [layers[i[0] - 1] for i in net.getUnconnectedOutLayers()]

      return outLayers
```

- Set input, get output layers, run YOLO

```
> net.setInput(blob)
> outLyrs  = getOutputLayers(net)
> preds    = net.forward(outLyrs)
```

get output layers

Run the actual object detection.

The output is a list, consisting of three 2D arrays; each array represents the output of one particular scale.

The size of each array is n x 85.

YOLO Implementation

- `preds`, the output from YOLO model

Index	Type	Size	Value
0	float32	(507, 85)	[[0.03275699 0.052097 0.3618731 ... 0. 0. 0. ...
1	float32	(2028, 85)	[[0.02476406 0.02638628 0.05257511 ... 0. 0. 0. ...
2	float32	(8112, 85)	[[0.00781204 0.00618281 0.01307961 ... 0. 0. 0. ...

Each row in the 2D array represents a box:

t_x	t_y	t_w	t_h	p_o	p_1	p_2	...	p_c

box co-ordinates

objectness score (a measure to tell if the box overlaps the ground truth the most compared to other boxes)

class scores (in the case of coco dataset, it will have 80 classes)

YOLO Implementation

- Create 3 empty lists. One for storing the class of the object detected; one for the confidence level; last one for the position and the size of the box

```
> classId          = []
> confidences      = []
> boxes            = []
```

- Extract information from the output

```
> for scale in preds:              loop through each 2D array in the list
      for pred in scale:           loop through each row in the 2D array
          scores      = pred[5:]              Extract all the class scores for this box
          clss        = np.argmax(scores)     Find out which class has the highest score
          confidence  = scores[clss]          Extract the score of the class (which has the highest score)
```

YOLO Implementation

- Continue previous for loop

```
> for scale in preds:
      for pred in scale:
          scores      = pred[5:]
          clss        = np.argmax(scores)
          confidence  = scores[clss]

          if confidence > 0.5:
              xc      = int(pred[0]*imgWidth)
              yc      = int(pred[1]*imgHeight)
              w       = int(pred[2]*imgWidth)
              h       = int(pred[3]*imgHeight)
              x       = xc - w/2
              y       = yc - h/2

              classId.append(clss)
              confidences.append(float(confidence))
              boxes.append([x, y, w, h])
```

Will consider the box only if the confidence is > 0.5

- xc: Get the actual box's center position, in x axis
- yc: Get the actual box's center position, in y axis
- w: Get the actual box's width
- h: Get the actual box's height
- x: Get the actual box's top-left position, in x axis
- y: Get the actual box's top-left position, in y axis

Add the item and its corresponding parameters to the lists

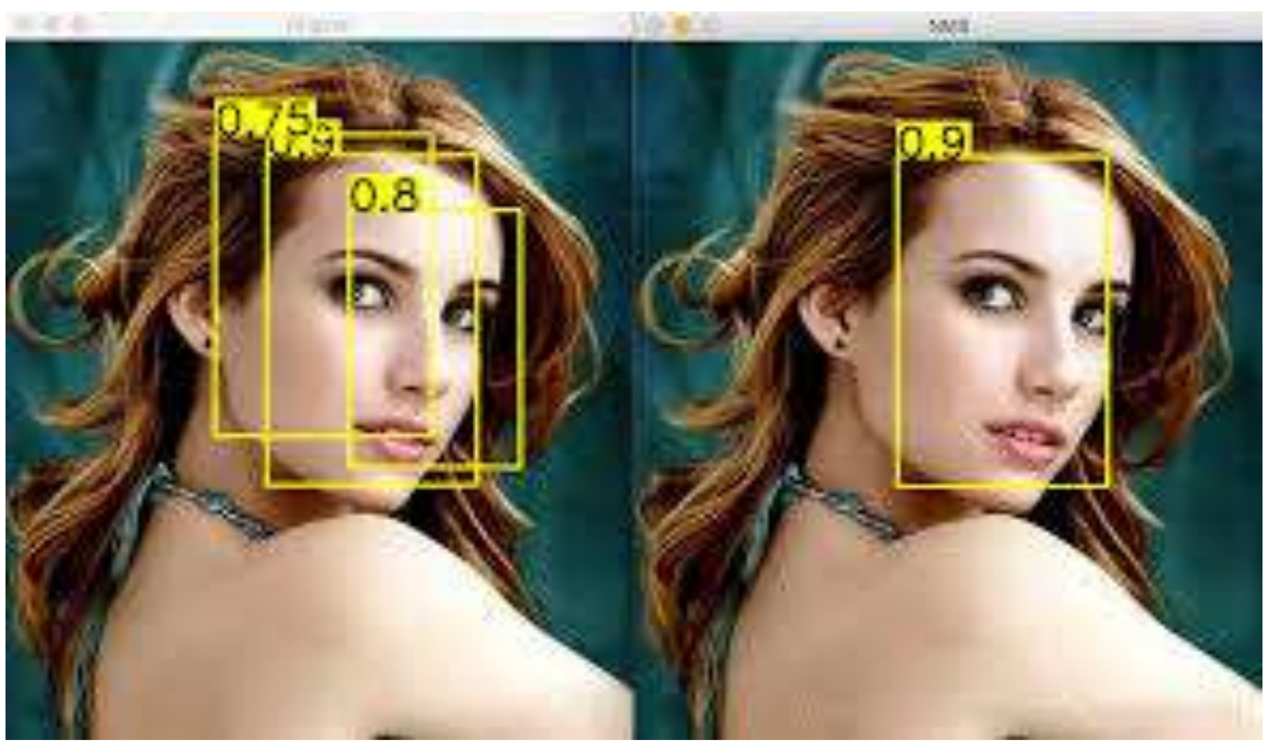
Must convert 'confidence' into float, else error in later function (NMSBoxes)

YOLO Implementation

- Perform non-maximal suppresion to remove redundant overlapping bounding boxes and get a single box with the highest confidence score

```
> scoreThres      = 0.5
> nmsThres        = 0.4
> selected        = cv2.dnn.NMSBoxes(bboxes=boxes,
                                     scores=confidences,
                                     score_threshold=scoreThres,
                                     nms_threshold=nmsThres)
```

'selected' is m x 1 array, m is the number of objects selected



Source: https://cloud.tencent.com/developer/article/1008757

YOLO Implementation

- Create colour set

```
> colorset  = np.random.uniform(0,
                                255,
                                size=(len(classes),3))
```

- Extract colour, label for the detected class, and also retrieve the box position for plotting

```
> for j in selected[:,0]:
      box     = boxes[j]
      color   = colorset[classId[j]]
      txtlbl  = str(classes[classId[j]])
      x       = int(box[0])
      y       = int(box[1])
      w       = int(box[2])
      h       = int(box[3])
```

because 'selected' is a m x 1 array, for convenience, we only take the first column

Get the colour for the detected class

Get the text label for the detected class

Convert values into int type to prevent potential TypeError when we draw rectangle or put text on image

YOLO Implementation

- Continue previous for loop

```
> for j in selected:
      ...
      w       = round(box[2])
      h       = round(box[3])
      cv2.rectangle(img,
                    (x,y),
                    (x+w,y+h),
                    color,
                    2)
      cv2.putText(img,
                  txtlbl,
                  (x,y-5),
                  cv2.FONT_HERSHEY_SIMPLEX,
                  0.5,
                  color,
                  1,
                  cv2.LINE_AA)
```

Draw the bounding box using rectangle

Put up the txt label that specify the class

YOLO implementation

- Note: the colour of the boxes will not be the same each time we run the code