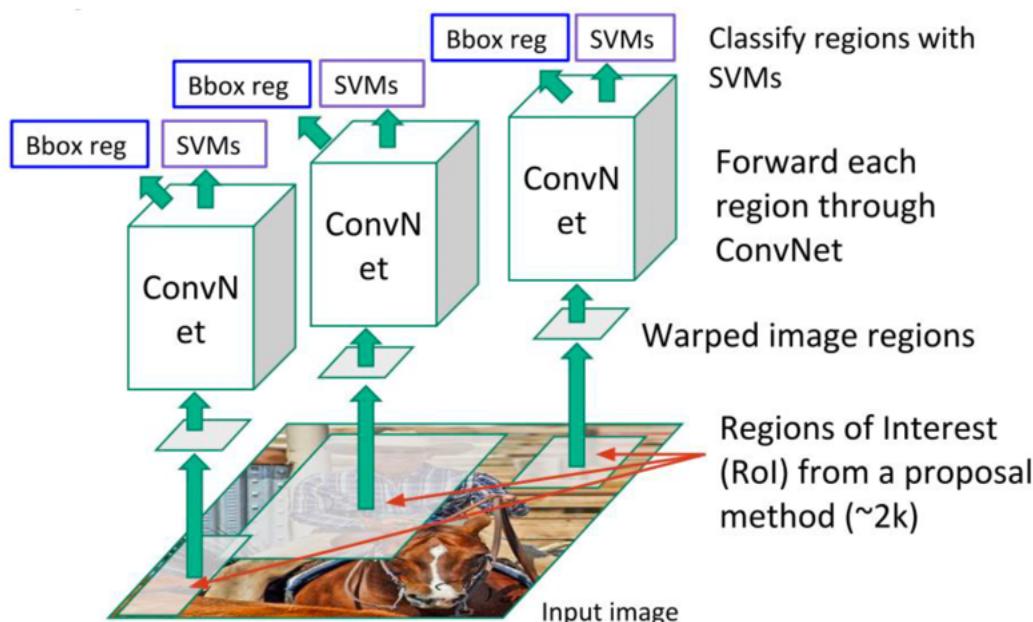


R-CNN

نحوه عملکرد آن به این صورت است که ابتدا با روش های سنتی پردازش تصویر و براساس ویژگی هایی مانند رنگ، بافت و ... تعدادی (حدود ۲۰۰۰ عدد) **proposal** (ناحیه های پیشنهادی) در تصویر انتخاب میکند، برای مثال برای این قسمت میتوان از **selective search** استفاده کرد؛ سپس هر کدام از این ناحیه ها برش میخورند و وارد یک شبکه CNN میشوند تا ویژگی های سطح بالا از آن ها استخراج شود، سپس خروجی این شبکه CNN وارد یک **classifier** و یک **bounding box registrator** میشود. قسمت **classifier** مشخص میکند این پروربوزال مربوط به کدام کلاس است (برای مثال کلاس ماشین، انسان، بکگراند) و قسمت **bounding box registrator** مرز های آن ناحیه را اصلاح میکند تا دقیق تر شیء را نشان دهد.

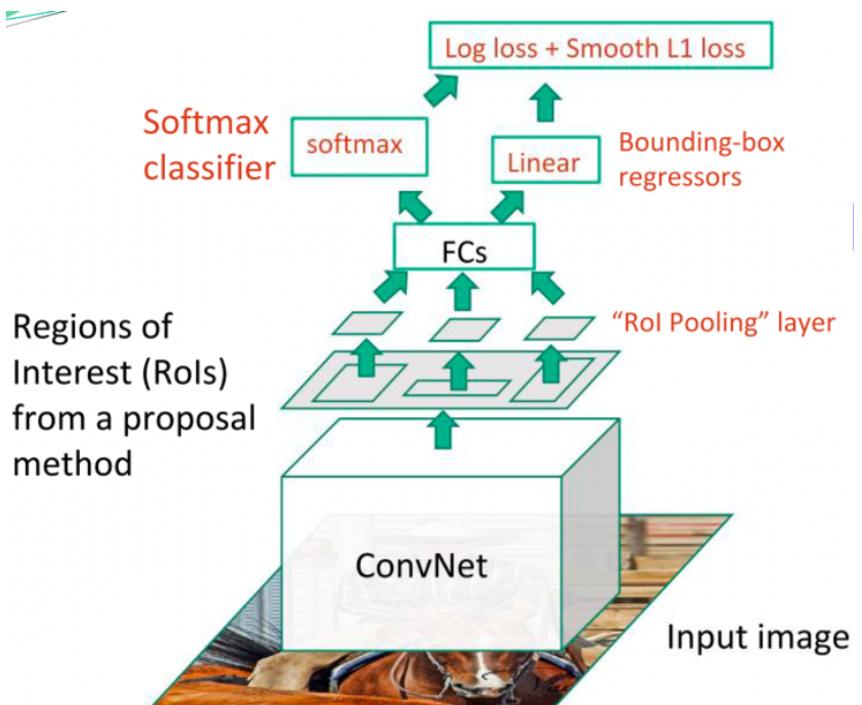


در روش R-CNN تصویر تبدیل به حدود ۲۰۰۰ پروربوزال میشود و این پروربوزال ها باید یکی یکی وارد شبکه CNN شوند که باعث میشود این روش با وجود داشتن دقت خوب ولی کند باشد.

Fast R-CNN

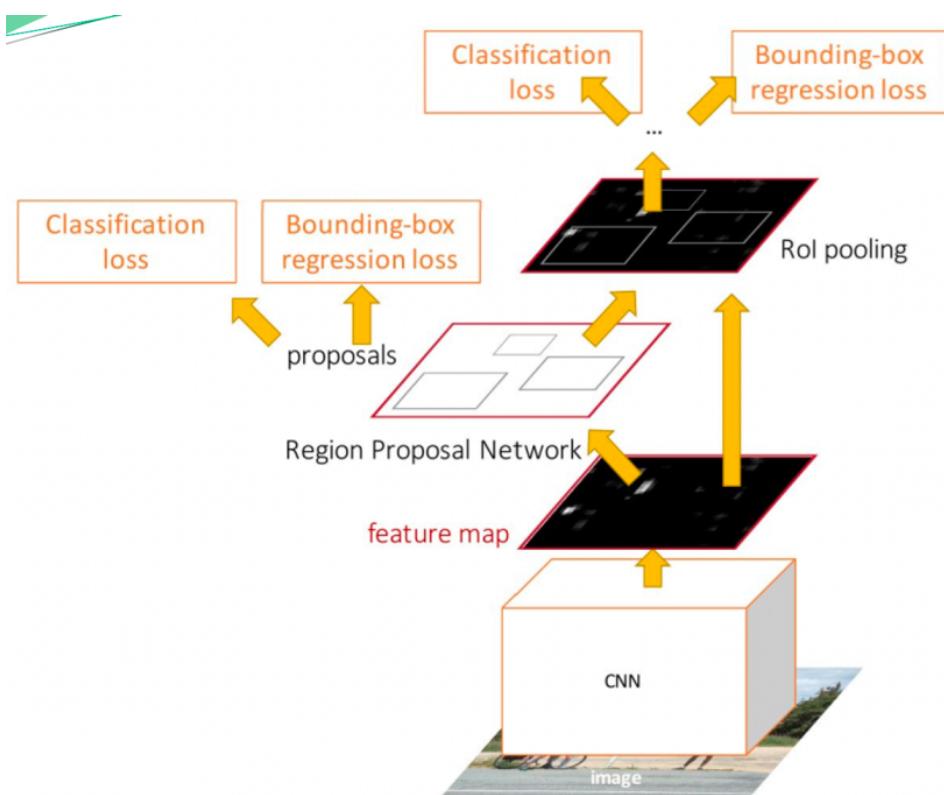
ایده این روش این است که تعداد اجرا شدن شبکه CNN را کاهش دهد به این منظور ابتدا شبکه CNN را روی کل تصویر اجرا میکند و در مرحله بعدی ناحیه متناظر با هر **proposal** را از خروجی شبکه CNN استخراج میکند، به این ترتیب لازم نیست برای هر پروربوزال شبکه CNN اجرا شود، شبکه CNN فقط یکبار اجرا میشود (البته با ابعاد ورودی بزرگتر) و سرعت پردازش به شدت افزایش میابد.

در این روش برخلاف R-CNN شبکه CNN فریز نیست و میتواند آموزش ببینند.



:Faster R-CNN

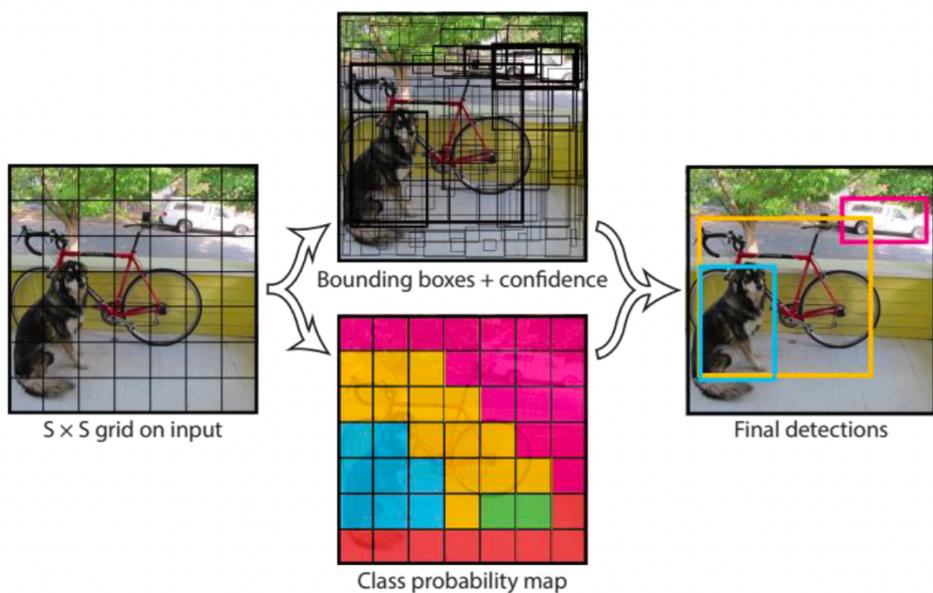
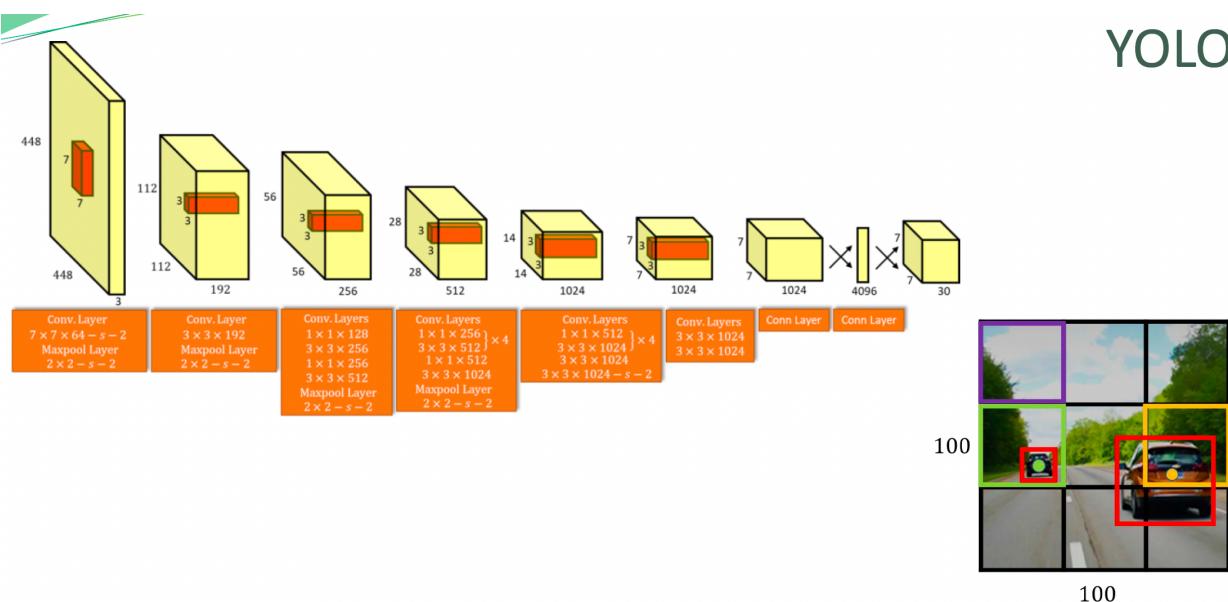
در این مدل به جای استفاده از روش های سنتی پردازش تصویر برای استخراج **proposal**، این وظیفه را هم به شبکه CNN میسپاریم به این ترتیب استخراج پروپوزال هم توسط شبکه CNN انجام میشود و قابلیت یادگیری دارد که باعث میشود سرعت قسمت **proposal** گیر بهتر شود زیرا از همان ویژگی هایی که شبکه CNN استخراج کرده کمک میگیریم. علاوه بر این قسمت استخراج پروپوزال دیگر یک پروپوزال گیر عمومی نیست و خاص همین مساله است که باعث میشود پروپوزال های بهتری استخراج شود و دقت آن ها افزایش یابد. همچنین به دلیل استخراج پروپوزال های بهتر به تعداد کمتری از آن ها نیاز داریم و به این ترتیب محاسبات بعدی نیز کاهش می یابد و باز هم افزایش سرعت داریم.



:YOLO

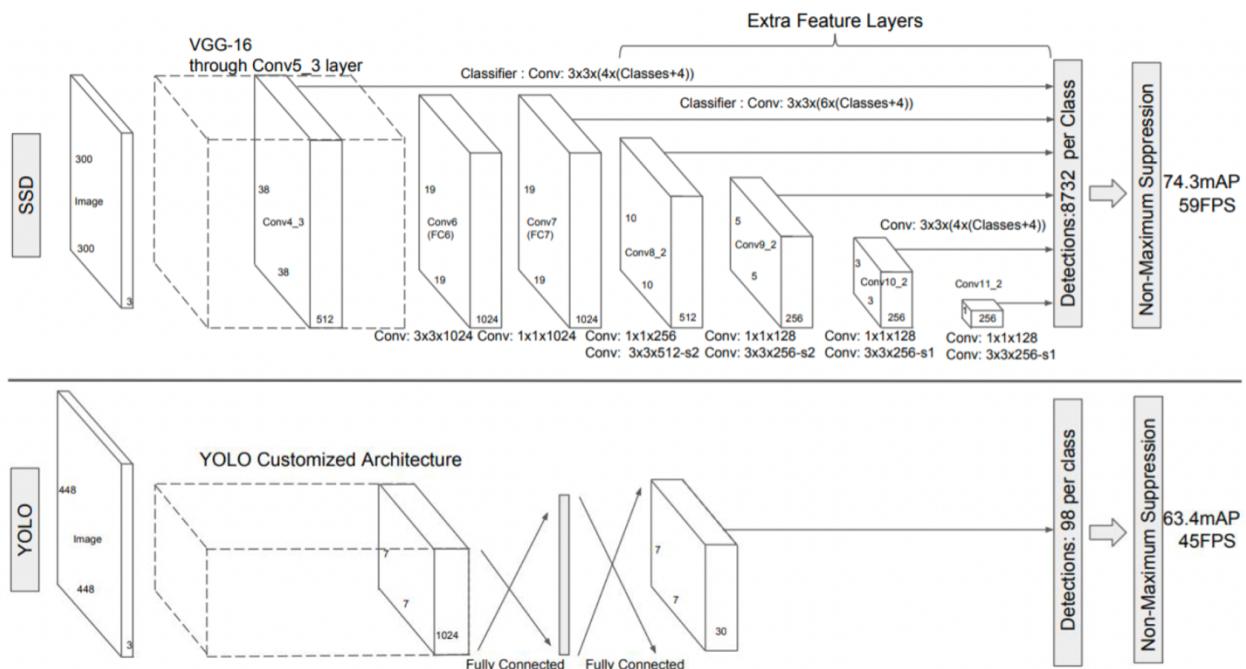
در این الگوریتم ابتدا تصویر به صورت یک شبکه (Grid) ناحیه بندی میشود، همانطور که در شبکه جلوتر میرویم به دلیل pooling های متوالی، نقشه های ویژگی کوچک تر و در عین حال عمیق تر میشوند و در آخر برای هر ناحیه یک bounding box regreter و یک classifier تعریف میشود.

YOLO



SSD

یکی از ایراد multi scale yolo v1 نبودن آن بود، ssd با ایده جالبی این مشکل را کاهش داد. در این الگوریتم از روش پنجره لغزان برای استخراج proposal استفاده میشود. ایده اصلی آن این است که برخلاف yolo که فقط از feature map نهایی برای تشخیص اشیا استفاده میکرد، از CNN های سطوح مختلف شبکه برای تشخیص اشیا و محل آن ها استفاده کنیم و به این ترتیب ویژگی چند مقیاسه بودن را به دست آوریم. همچنین برای تمایز قائل شدن بین اشیاء افقی و عمودی، از ایده anchor box استفاده میکند به این صورت که برای اشیاء افقی و عمودی classifier های متفاوتی آموزش میدهد.



RetinaNet

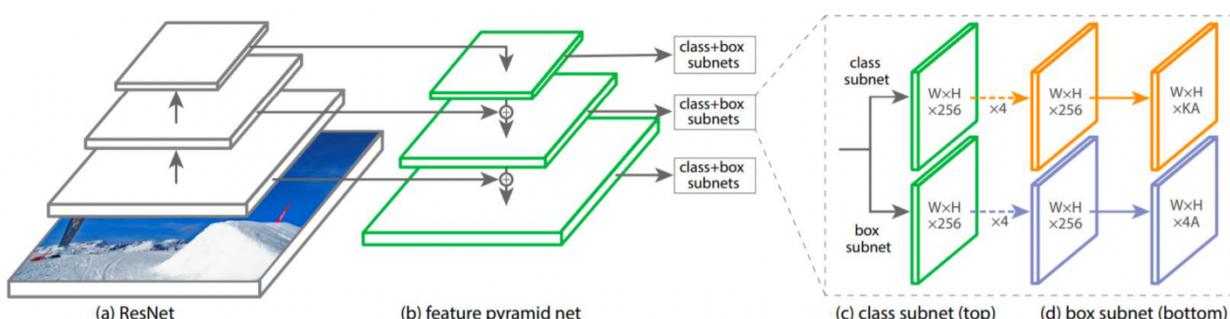
یکی از مشکلات **object detector** ها نا متوازن بودن تعداد پروربوزال ها در کلاس های مختلف است، به این ترتیب که معمولاً تعداد بسیار زیادی پروربوزال بکگراند داریم و تعداد کمی پروربوزال آبجکت، اگر از توابع خطایی مثل cross entropy استفاده کنیم، شبکه برای آن که خطای کلی را کاهش دهد مجبور میشود سعی کند خطای تشخیص بکگراند ها را کاهش دهد و آن را به صفر برساند، زیرا در مورد پروربوزال های بکگراند، حتی اگر هر کدام با خطای کمی تشخیص داده شده باشند ولی به دلیل زیاد بودن آن ها خطای نهایی باز هم زیاد است. این خاصیت باعث میشود شبکه روی یادگیری بکگراند ها تمرکز کند تا بتواند آن ها را با دقت بهتری تشخیص دهد و از آبجکت های اصلی غافل شود.

تابع خطای focal loss سعی میکند این مشکل را کاهش دهد به این ترتیب که سعی میکند loss را برای کلاس هایی که با دقت خوبی شناسایی شده اند، کاهش دهد.

یکی از ویژگی های retinanet این است که از این تابع loss استفاده میکند.

تغییرات retinanet نسبت به ssd:

در مرحله دسته بندی و مکان یابی فقط از یک لایه استفاده نمیکند بلکه برای بخش های مختلف (مثلاً بخشی که به دنبال اشیاء کوچک میگردد و یا بخشی که به دنبال اشیاء بزرگ میگردد) چند لایه تخصصی تعریف میکند. همچنین برای اشیاء کوچکتر فقط به featur map های همان سطح نگاه نمیکند بلکه از اطلاعات سطح بالایی که در featur map های سطوح بالاتر هست هم کمک میگیرد. البته به دلیل اضافه شدن این لایه ها سرعت آن تا حدی کاهش میابد.



```

def selective_search(image):
    """
    Runs the Selective Search algorithm on the image and returns the potential regions.

    Parameters
    -----
    image: ndarray
        Numpy array containing the BGR pixel values with the shape `(H, W, 3)`.

    Returns
    -----
    rectangles: list
        Potential regions in the image, formatted as `xmin`, `ymin`, `xmax`, `ymax`.
    """

#####
# Your code goes here.
https://pyimagesearch.com/2020/07/13/r-cnn-object-detection-with-keras-tensorflow-and-deep-learning/

# run selective search on the image and initialize our list of
# proposed boxes
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
ss.setBaseImage(image)
ss.switchToSelectiveSearchFast()
rects = ss.process()
rectangles= []
# loop over the rectangles generated by selective search
for (x, y, w, h) in rects:
    # convert our bounding boxes from (x, y, w, h) to (startX,
    # startY, startX, endY)
    rectangles.append((x, y, x + w, y + h))

#####
return rectangles

```

در این قسمت با استفاده از الگوریتم proposal تعدادی selective_search در تصویر انتخاب و برگردانده میشود. فرمت این پروپوزال ها به صورت آرایه ای از مستطیل هاست که مختصات دو گوشه روبرویی آن ها ذخیره شده است. برای پیاده سازی این تابع از کتابخانه open cv استفاده شده است.

```

def area(box):
    xmin, ymin, xmax, ymax = box

#####
# Your code goes here.
area = (xmax - xmin + 1) * (ymax - ymin + 1)
#####

return area

```

در این قسمت مساحت این پروپوزال ها محاسبه میشود که در قسمت های بعد از آن استفاده خواهد شد. برای جلوگیری از صفر شدن این مساحت ها و ایجاد مشکل در قسمت هایی که مساحت در مخرج قرار دارد، مقدار ۱ به ترم های ضربی آن اضافه شده است.

```

def compute_iou(box_a, box_b):
    xmin_a, ymin_a, xmax_a, ymax_a = box_a
    xmin_b, ymin_b, xmax_b, ymax_b = box_b

    #####
    # Your code goes here.
    #https://pyimagesearch.com/2020/07/13/r-cnn-object-detection-with-keras-tensorflow-and-deep-learning/
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(xmin_a, xmin_b)
    yA = max(ymin_a, ymin_b)
    xB = min(xmax_a, xmax_b)
    yB = min(ymax_a, ymax_b)
    # compute the area of intersection rectangle
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (xmax_a - xmin_a + 1) * (ymax_a - ymin_a + 1)
    boxBArea = (xmax_b - xmin_b + 1) * (ymax_b - ymin_b + 1)
    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the intersection area
    iou = interArea / float(boxAArea + boxBArea - interArea)
    # return the intersection over union value
#####

return iou

```

در این قسمت معیار **iOU** محاسبه شده است که نسبت اشتراک به اجتماع دو ناحیه را به دست می آورد. طریقه کار کرد آن به این صورت است که ابتدا دو گوش ناحیه اشتراکی را به دست می آورد سپس با جمع کردن مساحت هر ناحیه و کم کردن مساحت ناحیه اشتراک، مساحت ناحیه اجتماع نیز به دست می آید.

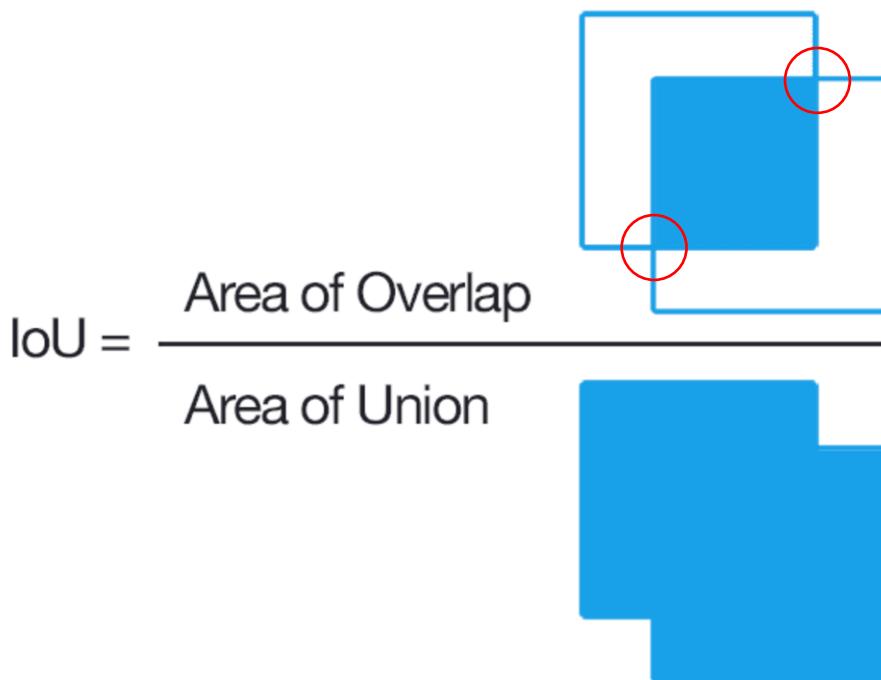


Figure 4: Computing the Intersection over Union is as simple as dividing the area of overlap

این معیار برای مقایسه پر پوزال ها با **groundTruth** ها استفاده می شود، اگر **iou** یک پر پوزال از حدی بیشتر شود به این معناست که با یک **groundTruth** اشتراک زیادی داشته و پر پوزال مناسبی است که پر پوزال ثابت نامیده می شود و در صورتی که **iou** آن از حدی کمتر باشد، و پر پوزال نامناسبی است و پر پوزال منفی نامیده می شود.

```

def fully_inside(region, box):
    xmin_r, ymin_r, xmax_r, ymax_r = region
    xmin_b, ymin_b, xmax_b, ymax_b = box

    #####
    # Your code goes here.
    if xmin_b < xmin_r and ymin_b < ymin_r and xmax_b > xmax_r and ymax_b > ymax_r:
        result = True
    else:
        result = False
    #####
    return result

```

در این قسمت بررسی میکنیم آیا پروپوزالی کاملا در یک ناحیه groundTruth قرار دارد یا نه، اگر قرار داشت حتی اگر معیار iou آن مناسب نبود در دسته پروپوزال های منفی قرار نمیگیرد.

```

for subset, iterator in subsets.items():

    index = 1

    for sample in tqdm(iterator, desc=subset):

        num_positives = 0
        num_negatives = 0

        image = sample['image'].numpy()
        height, width, channels = image.shape
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        objects = sample['objects']
        labels = objects['label'].numpy()

        if target_class not in labels:
            continue

        boxes = objects['bbox'].numpy()
        boxes = [(xmin * width, ymin * height, xmax * width, ymax * height) for (ymin, xmin, ymax, xmax) in boxes]
        regions = selective_search(image)

        regions = sorted(regions, key=area)
        regions = reversed(regions)

        matched = [(region, [(box, compute_iou(box, region)) for box in boxes]) for region in regions]
        matched = [(region, *max(li, key=lambda pair: pair[1])) for region, li in matched]

        for region, box, iou in matched:
            xmin, ymin, xmax, ymax = region

            if iou > positive_thresh:
                if num_positives < max_positives:
                    crop = image[ymin:ymax, xmin:xmax, :]
                    crop = cv2.resize(crop, target_size)
                    filepath = os.path.join(output_dir, subset, 'positive', f'{index:06d}.jpg')
                    cv2.imwrite(filepath, crop)
                    num_positives += 1
                    index += 1

            if iou < negative_thresh and not fully_inside(region, box):
                if num_negatives < max_negatives:
                    crop = image[ymin:ymax, xmin:xmax, :]
                    crop = cv2.resize(crop, target_size)
                    filepath = os.path.join(output_dir, subset, 'negative', f'{index:06d}.jpg')
                    cv2.imwrite(filepath, crop)
                    num_negatives += 1
                    index += 1

            if num_positives == max_positives and num_negatives == max_negatives:
                break

```

نتایج زیر به دست آمد و تعدادی از این پروپوزال‌ها با لیبل مربوطه نشان داده شده است.

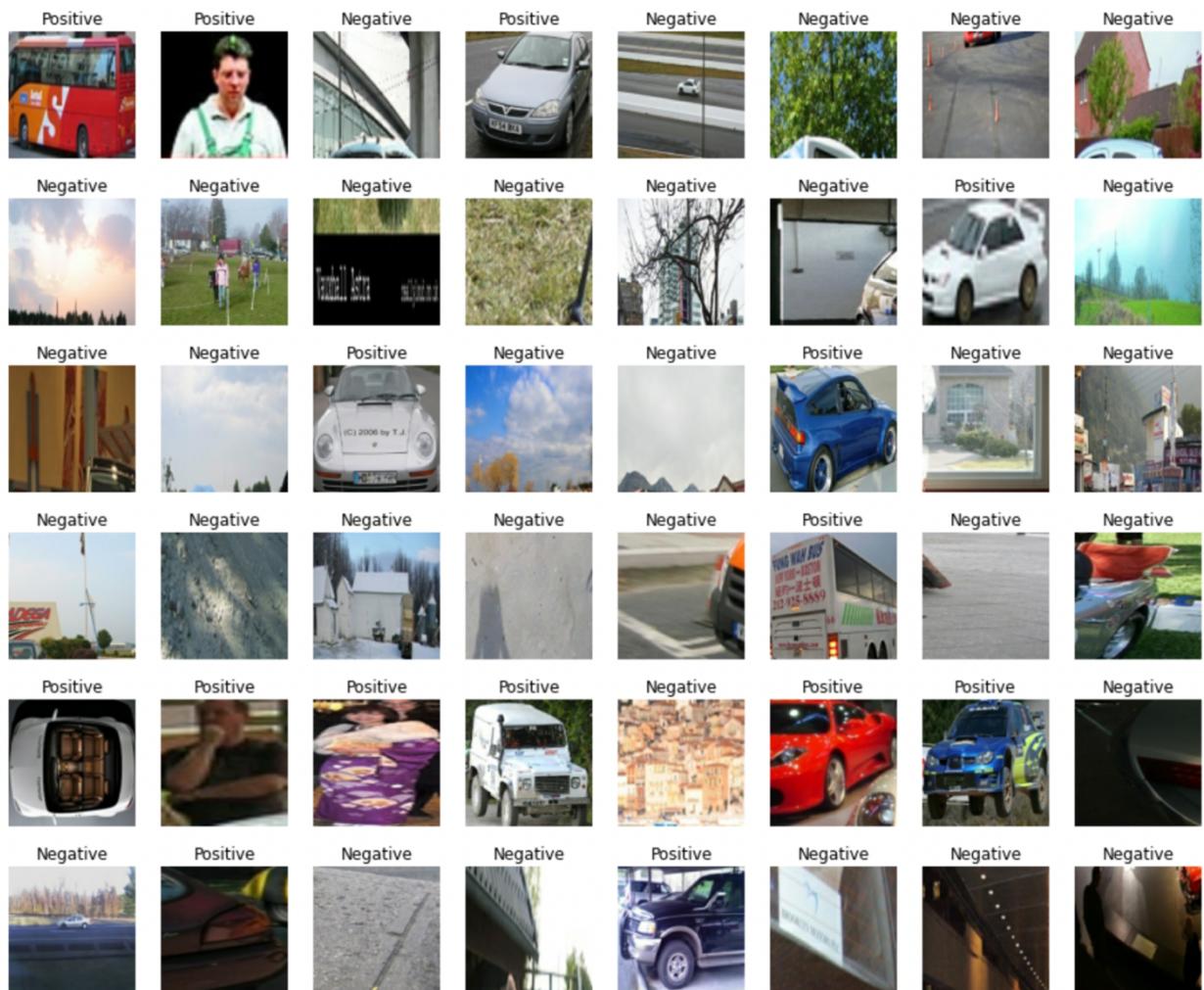
این قسمت معیار IOU را برای تک تک پروپوزال‌ها محاسبه می‌کند و بر اساس این معیار پروپوزال‌ها را به دو دسته منفی و مثبت دسته بندی می‌کند.

The total size of the extracted dataset came out to:

```
[ ] !du -sh data
```

and the number of positive and negative samples, respectively, are:

```
[ ] !ls data/train/positive -l | wc -l  
!ls data/train/negative -l | wc -l  
  
1563  
4020
```



```
input_tensor = tf.keras.Input(shape=(*target_size, 3), dtype = tf.uint8)
#####
#https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/preprocess_input
x = tf.cast(input_tensor, tf.float32)
x = tf.keras.applications.resnet50.preprocess_input(x)
feature_map = resnet(x)
# Your code goes here.
#####
```

حال باید نتایج را به یک شبکه از پیش آموزش دیده resnet50 بدهیم، به این منظور این شبکه را البته به غیر از لایه های آخر آن (include top=False) وارد مکنیم.

پس از اعمال preprocessing های مربوط به این شبکه بر روی داده های خودمان، این داده ها را به شبکه میدهیم.

```
#####
# construct the head of the model that will be placed on top of the base model

headModel = Dense(64, activation="relu")(feature_vector)
headModel = Dense(32, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
output_tensor = Dense(1, activation="sigmoid")(headModel)

# Your code goes here.
#####
```

برای لایه های آخر، یک لایه dense سپس یک لایه dropout و در آخر یک لایه dense با یک نورون و تابع فعال ساز sigmoid (چون مسئله ما تک کلاسه است) تعریف میکنیم.

```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
model_check_point = tf.keras.callbacks.ModelCheckpoint("model3.h5", monitor='val_loss', verbose=1, save_best_model=True)

model.fit(train_set, validation_data=validation_set, epochs=10, callbacks=[early_stop, model_check_point])

Epoch 1/10
88/88 [=====] - ETA: 0s - loss: 0.0069 - auc: 0.9998 - binary_accuracy: 0.9927
Epoch 1: saving model to model3.h5
88/88 [=====] - 126s 1s/step - loss: 0.0069 - auc: 0.9998 - binary_accuracy: 0.9927 - val_loss: 0.0479 - val_auc: 0.9902 - val_binary_accuracy: 0.9605
Epoch 2/10
88/88 [=====] - ETA: 0s - loss: 0.0065 - auc: 0.9997 - binary_accuracy: 0.9923
Epoch 2: saving model to model3.h5
88/88 [=====] - 73s 830ms/step - loss: 0.0065 - auc: 0.9997 - binary_accuracy: 0.9923 - val_loss: 0.0482 - val_auc: 0.9904 - val_binary_accuracy: 0.9595
Epoch 3/10
88/88 [=====] - ETA: 0s - loss: 0.0054 - auc: 0.9998 - binary_accuracy: 0.9944
Epoch 3: saving model to model3.h5
88/88 [=====] - 73s 829ms/step - loss: 0.0054 - auc: 0.9998 - binary_accuracy: 0.9944 - val_loss: 0.0499 - val_auc: 0.9907 - val_binary_accuracy: 0.9601
Epoch 4/10
88/88 [=====] - ETA: 0s - loss: 0.0049 - auc: 0.9999 - binary_accuracy: 0.9939
Epoch 4: saving model to model3.h5
88/88 [=====] - 73s 830ms/step - loss: 0.0049 - auc: 0.9999 - binary_accuracy: 0.9939 - val_loss: 0.0545 - val_auc: 0.9898 - val_binary_accuracy: 0.9595
<keras.callbacks.History at 0x7f4a8026ae80>
```

پس از آموزش شبکه نتایج بالا حاصل شد و به دقت 0.99 رسیدیم.

```
def extract_crops(image, regions, target_size):
    """
    Extracts the `regions` from the image and returns the resized versions.

    Parameters
    -----
    image: ndarray
        Numpy array containing the BGR pixel values with the shape `(H, W, 3)`.

    regions: list
        Potential regions in the image, formatted as `xmin`, `ymin`, `xmax`, `ymax`.

    target_size: tuple
        Target height and width of the images, as a tuple of integers.

    Returns
    -----
    crops: list
        Cropped and resized regions.
    """

#####
# Your code goes here.
regions = selective_search(image)
crops = np.empty((len(regions), target_size[0], target_size[1], image.shape[-1]))
for index, region in enumerate(regions):
    xmin, ymin, xmax, ymax = region
    crop = image[ymin:ymax, xmin:xmax, :]
    crop = cv2.resize(crop, target_size)
    crops[index, :, :, :] = crop[:, :, :]
#####

return crops
```

```
[48] crops = extract_crops(image, regions, target_size)
crops.shape
```

```
(2293, 224, 224, 3)
```

حال میتوانیم به وسیله `model.predict` مدل خودمان را تست کنیم:

```
[61] scores = model.predict(crops, batch_size)
scores = np.squeeze(scores)
scores.shape
```

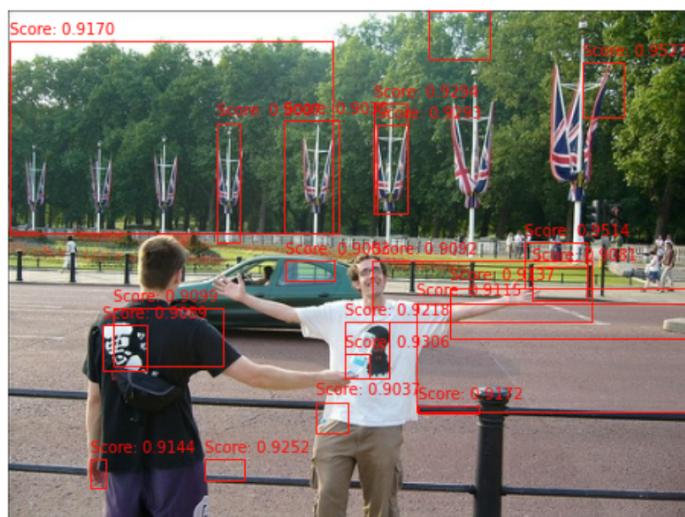
```
36/36 [=====] - 6s 183ms/step
(2293,)
```

```
[62] detections = [(region, score) for region, score in zip(regions, scores) if score > confidence_thresh]
len(detections)
```

21

تعداد ۲۱ ناحیه در تصویر `predict` شد

```
[63] detection_boxes = [box for box, score in detections]
detection_scores = [score for box, score in detections]
visualize(image, detection_boxes, detection_scores, annotate=True, annotation='Score: {:.4f}')
```



پس از پایان آموزش شبکه حال میتوانیم با استفاده از آن استنتاج کنیم.
تابع بالا پروپوزال های تصویر ورودی را به دست می آورد سپس آن ها را میبرد و به صورت یک لیست برمیگرداند.

NMS (non maximum suppression)

ممکن است چند پروپوزال نزدیک به هم برای یک ناحیه ارائه شده باشد، ما باید بهترین آن ها را انتخاب کنیم

```
▶ def non_max_suppression(detections, threshold=0.2):
```

```

bounding_boxes = [box for box, score in detections]
confidence_score = [score for box, score in detections]

picked_boxes = []
picked_score = []

#####
# Your code goes here.
# https://zditect.com/code/solve-the-problem-of-overlapping-two-frames-of-image-target-detection.html

# If no bounding boxes, return empty list
if len(bounding_boxes) == 0:
    return [], []

# Bounding boxes
boxes = np.array(bounding_boxes)

# coordinates of bounding boxes
start_x = boxes[:, 0]
start_y = boxes[:, 1]
end_x = boxes[:, 2]
end_y = boxes[:, 3]

# Confidence scores of bounding boxes
score = np.array(confidence_score)

# Compute areas of bounding boxes
areas = (end_x - start_x + 1) * (end_y - start_y + 1)

# Sort by confidence score of bounding boxes
order = np.argsort(score)

# Iterate bounding boxes
while order.size > 0:
    # The index of largest confidence score
    index = order[-1]

    # Pick the bounding box with largest confidence score
    picked_boxes.append(bounding_boxes[index])
    picked_score.append(confidence_score[index])

    # Compute ordinates of intersection-over-union (IOU)
    x1 = np.maximum(start_x[index], start_x[order[:-1]])
    x2 = np.minimum(end_x[index], end_x[order[:-1]])
    y1 = np.maximum(start_y[index], start_y[order[:-1]])
    y2 = np.minimum(end_y[index], end_y[order[:-1]])

    # Compute areas of intersection-over-union
    w = np.maximum(0.0, x2 - x1 + 1)
    h = np.maximum(0.0, y2 - y1 + 1)
    intersection = w * h

    # Compute the ratio between intersection and union
    ratio = intersection / (areas[index] + areas[order[:-1]] - intersection)

    left = np.where(ratio < threshold)
    order = order[left]

#####

return picked_boxes, picked_score

```

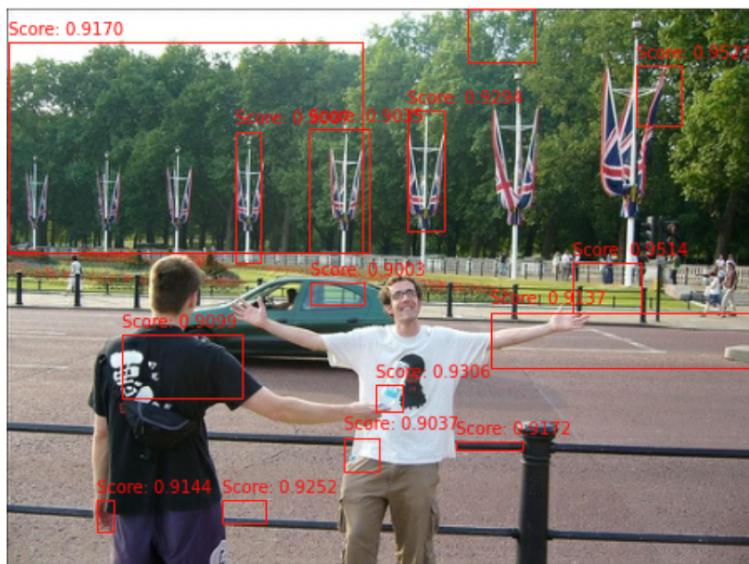
اگر تعدادی score های متفاوت با هم تداخل زیادی داشته باشند، این تابع بهترین آن ها را انتخاب میکند.

```
[65] detection_boxes, detection_scores = non_max_suppression(detections)
    len(detection_boxes)
```

15

با اعمال `non max suppression` تعداد نواحی به ۱۵ عدد کاهش یافت.

```
visualize(image, detection_boxes, detection_scores, annotate=True, annotation='Score: {:.4f}')
```



این الگوریتم تا حدی توانست خودرو را پیدا کند اما اصلاً عملکرد خوبی نداشت. دلیل آن میتواند این باشد که در فرایند آموزش به غیر از تصویر ماشین، تصویر اشیاء دیگری مثل کشتی، قطار و انسان هم به عنوان **positive** به شبکه آموزش داده شده اند و چون همه آن ها یک کلاس دارند درواقع شبکه ترکیبی از ویژگی های این آبجکت ها را یادگرفته نه فقط ماشین را بنابراین نمیتواند هیچکدام از آن آبجکت ها را به خوبی شناسایی کند.

دلیل دیگر این میتواند باشد که قسمت پروپوزال گیر، ناحیه ماشین را به عنوان پروپوزال نشناخته.

-۳

یک **object detector** مدرن معمولاً از دو بخش تشکیل شده است. یک **backbone** که معمولاً یک شبکه از پیش آموزش دیده روی **imagenet** است، و یک قسمت **head** که برای تشخیص کلاس و ناحیه اشیاء به کار میرود. برای **detector** هایی که روی **gpu** اجرا میشوند میتوان از شبکه های **VGG**, **ResNet**, **ResNeXt** و یا **denseNet** استفاده کرد.

برای **detector** هایی که روی **cpu** اجرا میشوند میتوان از شبکه های **ShuffleNet**, **MobileNet**, **SqueezeNet** یا **detectoR** استفاده کرد.

قسمت **head** میتواند دو نوع باشد. **object detector** هایی که مرحله ای و یا **object detector** های دو مرحله ای. از جمله **object detector** های دو مرحله ای میتوان به سری های **R-cnn** اشاره کرد. و از جمله هایی که مرحله ای میتوان به سری های **Yolo**, **SSD**, **RetinaNet** اشاره کرد.

برای **object detector** های مدرنی که در این سال ها استفاده میشوند اغلب یک سری لایه دیگر بین **backbone** و **head** میگذارند که برای جمع آوری نقشه های ویژگی از مراحل مختلف استفاده میشود، این قسمت **neck** نامیده میشود.

معمولًا بخش neck از چندین مسیر از پایین به بالا و چندین مسیر از بالا به پایین تشکیل شده است. از شبکه هایی که این مکانیسم را دارند میتوان به FPN, PAN, BiFPN, NAS-FPN اشاره کرد.

bag of freebies

معمولًا یک object detector مرسوم به صورت آفلاین آموزش داده میشود، محققان همیشه دوست داشته اند که از این مزیت استفاده کنند و روش هایی برای آموزش توسعه بدهند که بدون نیاز به افزایش هزینه استنتاج میتوانند باعث شوند object detector بتواند به دقت های بالاتری دست یابد.

ما این روش ها را که فقط استراتژی آموزش را تغییر میدهند و یا فقط باعث افزایش هزینه آموزش میشوند را bag of freebies مینامیم.

روشی که معمولًا object detector ها انتخاب میکنند و با تعریف bag of freebies همخوانی دارد روش augmentation است.

هدف data augmentation بالا بردن خاصیت تغییر پذیری تصاویر ورودی و درنتیجه بالا رفتن robustness مدل object detector در برابر تصاویری که از محیط های مختلف به دست آمده اند میشود.

bag of specials

به مژول های افزونه ای و روش های پس پردازشی که هزینه استنتاج را کمی افزایش میدهند اما در مقابل میتوانند دقت object detector را به طور قابل توجهی افزایش دهند bag of specials میگوییم.

به طور کلی این مژول ها برای ارتقا ویژگی های خاصی در مدل ها هستند، برای مثال بزرگ کردن receptive field، معرفی کردن مکانیسم ها توجه، و یا تقویت ظرفیت ادغام ویژگی و ... همچنین پس پردازش روشی برای غربالگری نتایج prediction مدل است.