DEEP LEARNING

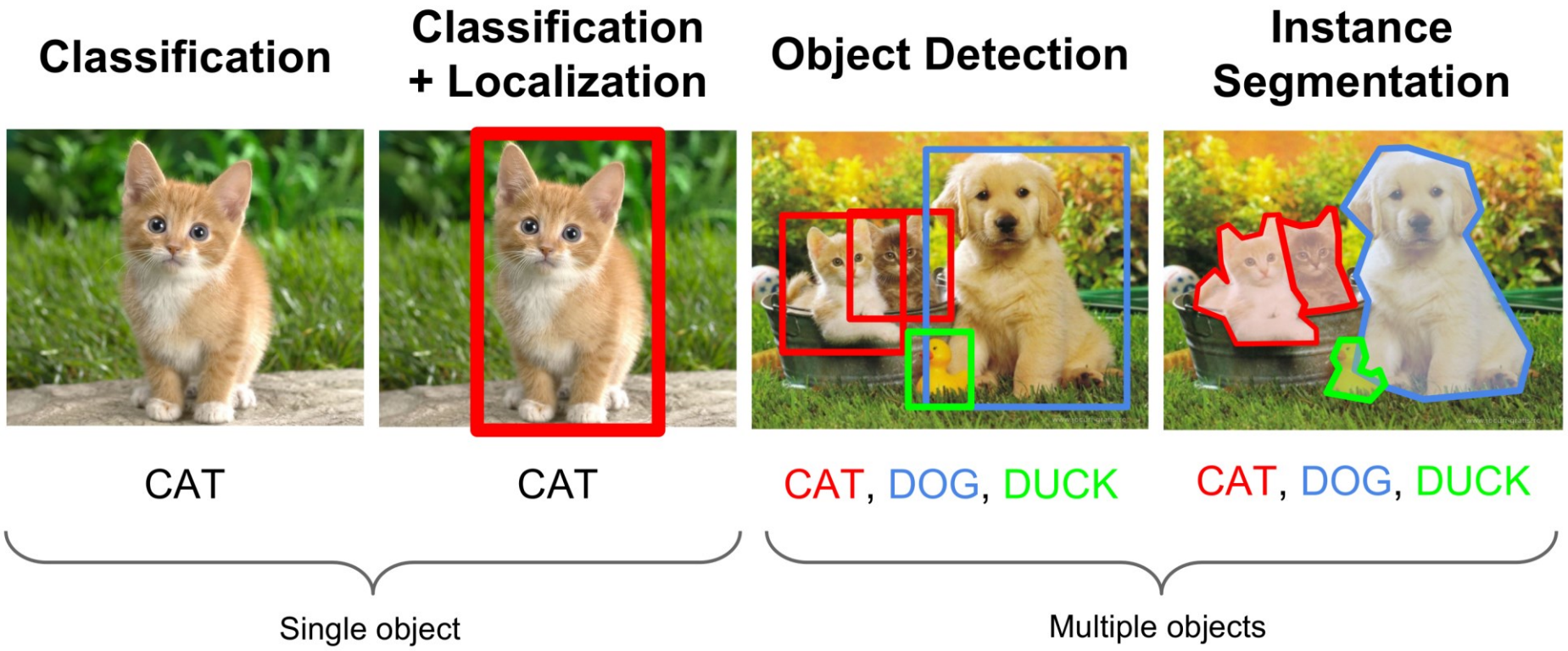
Object Detection & Segmentation

Ashish Pujari

Lecture Outline

1. Image Labeling
2. R-CNN
3. YOLO
4. Segmentation

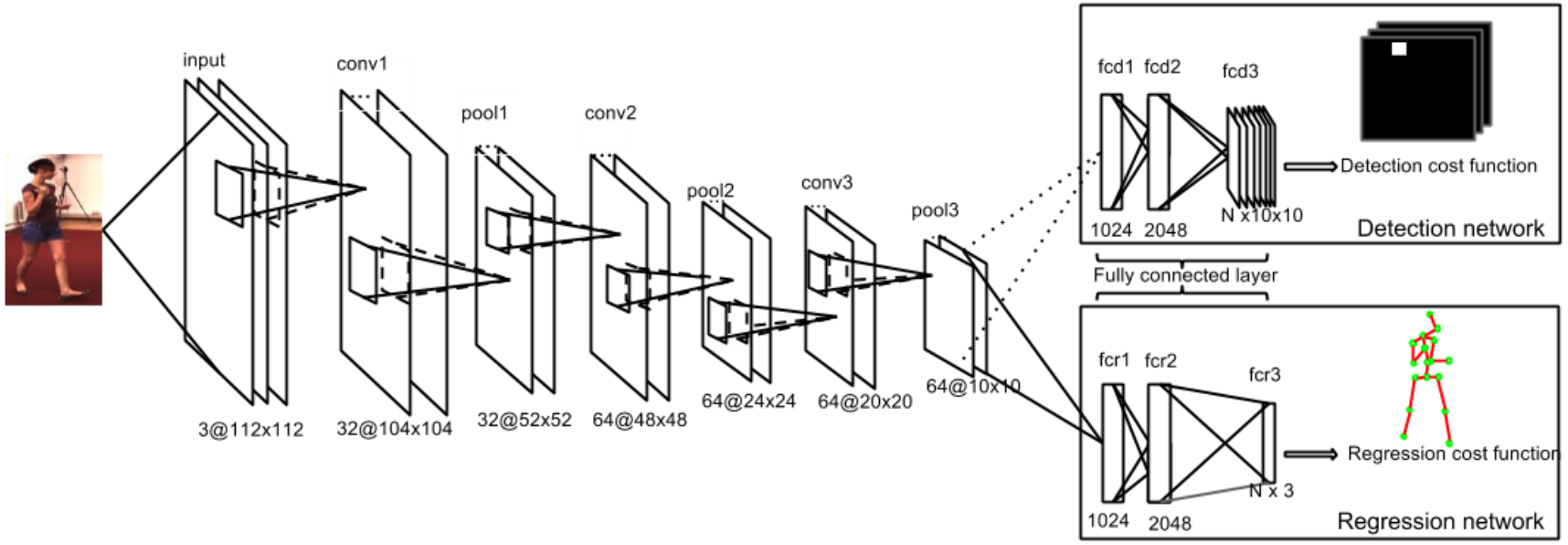
Computer Vision



http://cs231n.stanford.edu/

Multi-Task Learning (MTL)

- MTL is intended to impose knowledge sharing while solving multiple correlated tasks simultaneously - Localization, Detection
- By sharing representations between related tasks – e.g. classification and bounding box segmentation, we can enable our model to generalize better on our original task



https://arxiv.org/pdf/1601.00400.pdf

http://ruder.io/multi-task/

Image Labels

- The network identifies where the object is and puts a bounding box around it
- Add four more numbers to the Target **Y**, which identify the x and y coordinates of the upper left corner and the height and width of the box *(bx, by, bh, bw)*.

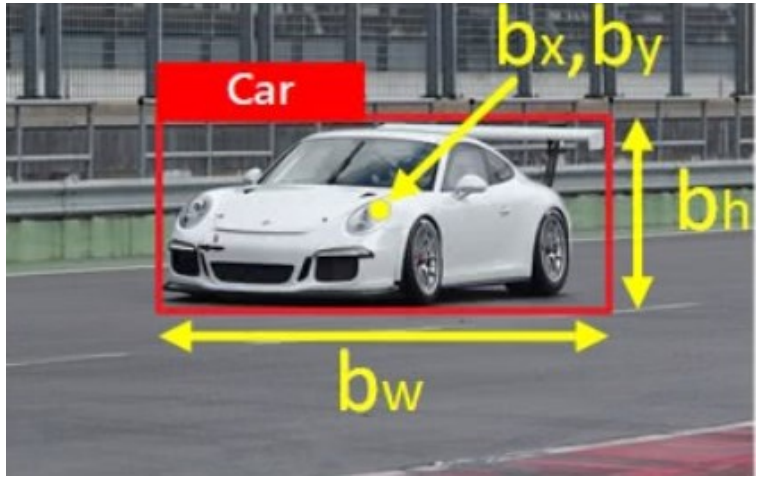
$$Y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

confidence that an object exists

Bounding box coordinates

Classification probabilities

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



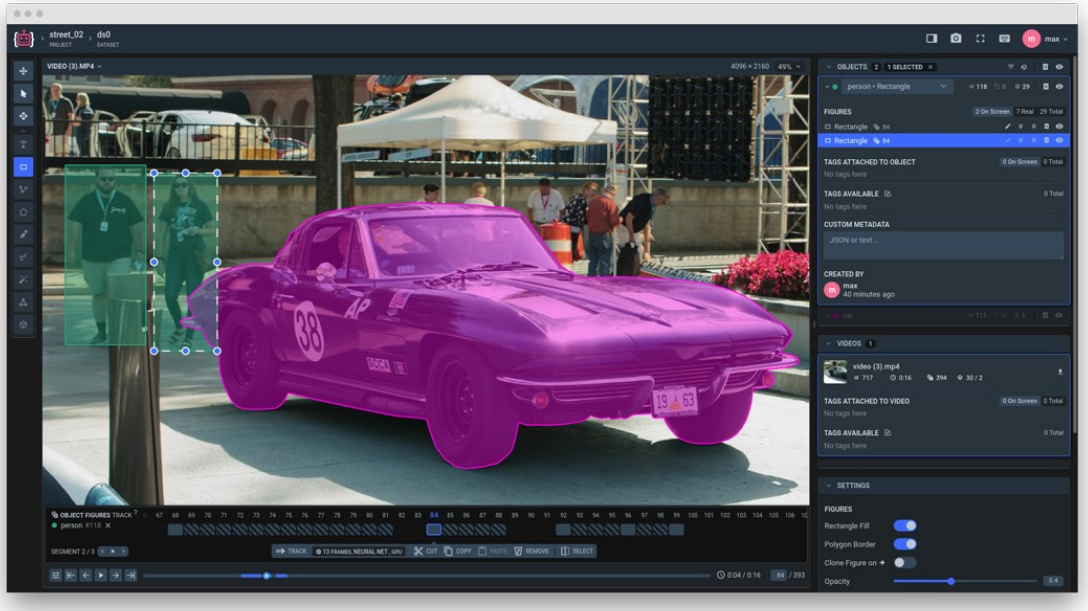
Car Detected

$$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

No Object

Image Labeling

https://pypi.org/project/labelImg/



https://supervise.ly/

- LabelImg

```
pip install labelImg
```

Bounding Box Parameters

- Each bounding box consists of 4 other parameters (x,y,w,h) corresponding to $(center\ coordinate(x,y), width, height)$ of a bounding box.
- Combining with the confidence score, each bounding box consists of 5 parameters

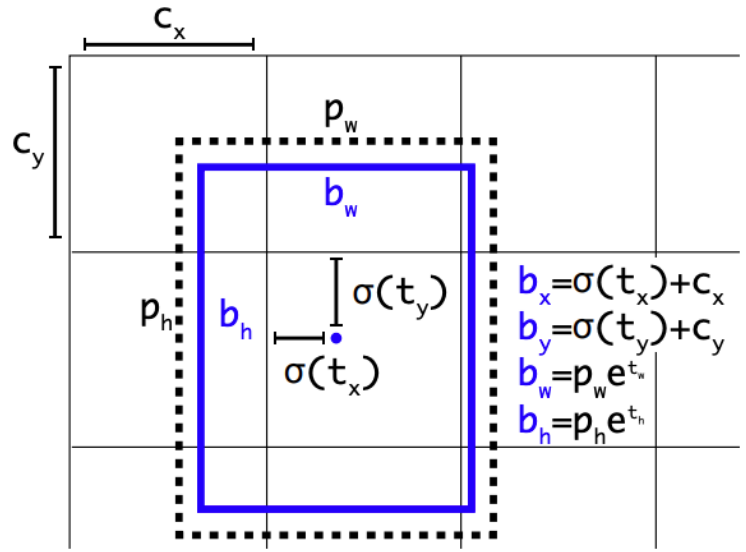
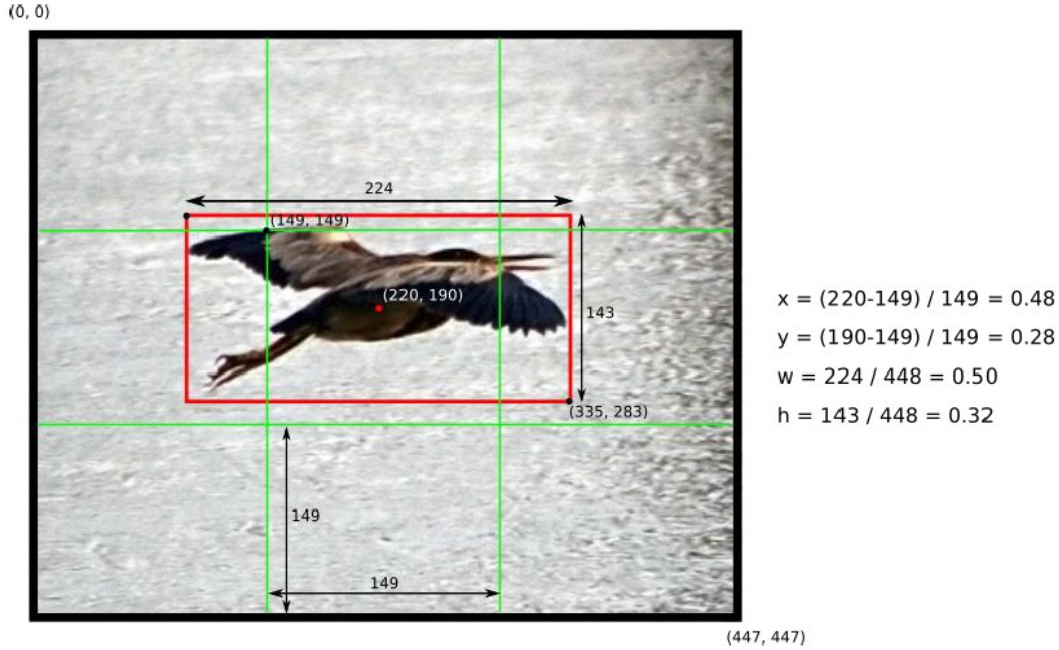
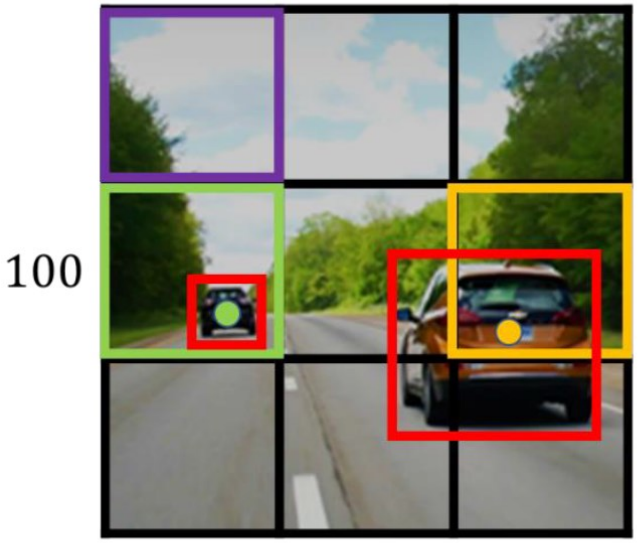


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].

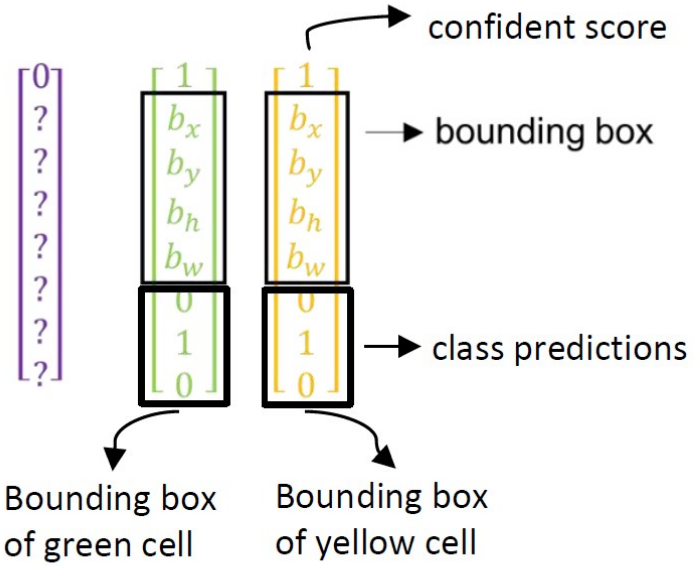


Labeled Vectors



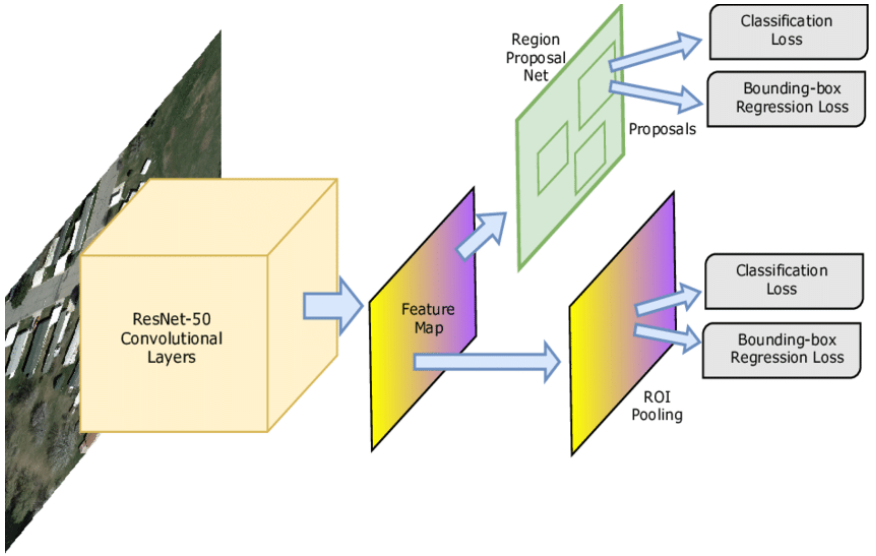
Labels for training for each grid cell:

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



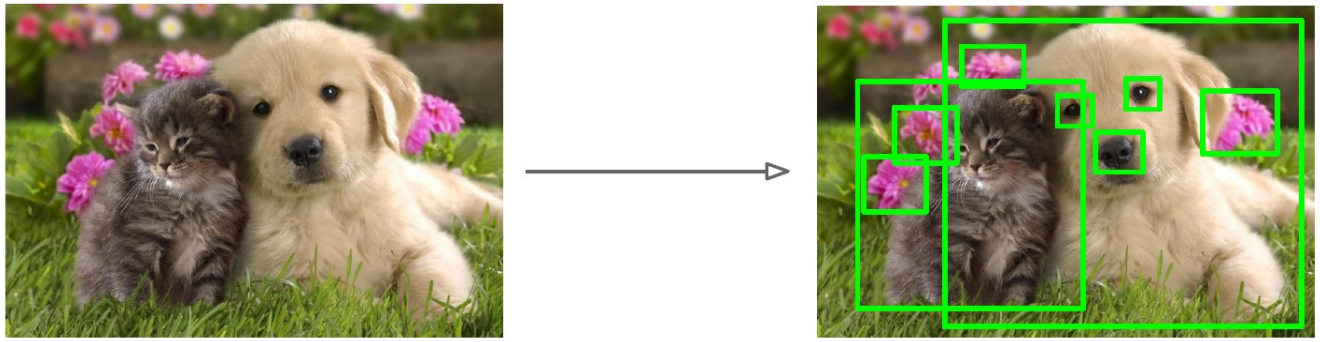
Region-Based CNN (R-CNN)

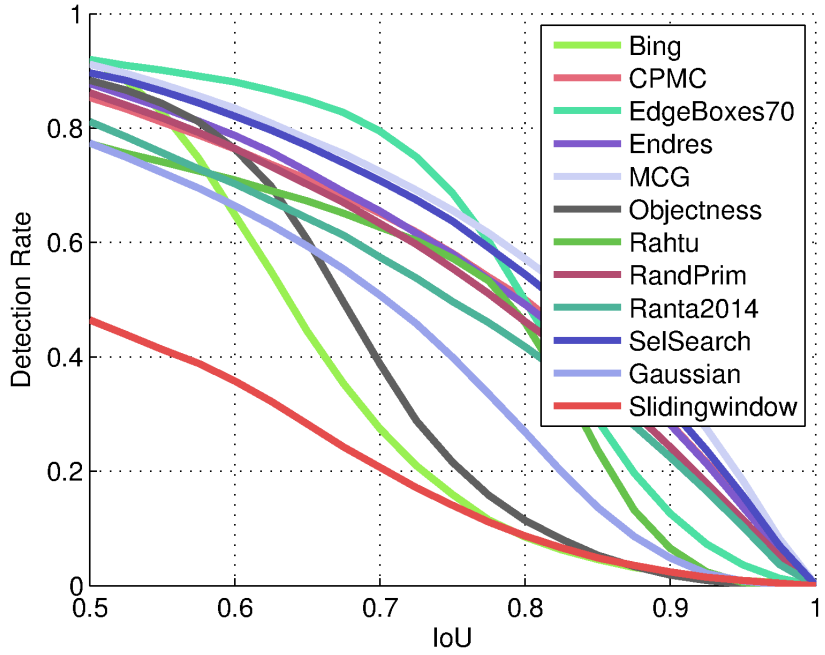
- R-CNNs solve the problem of drawing bounding boxes over all of the objects
- Step 1 - Region proposal step
  - Model proposes a set of regions of interests by select search or regional proposal network
  - proposed regions are sparse as the potential bounding box candidates can be infinite
- Step 2 - Classification step
  - A classifier only processes the region candidates
- Algorithms
  - R-CNN, Fast R-CNN, Faster R-CNN, R-FCN, Mask R-CNN



Region Proposal Step

- Region/object proposal algorithms aim to identify a small set of regions such that each object in the image is approximately delineated by at least one proposed region
- Find “blob like” image regions that are likely to contain objects

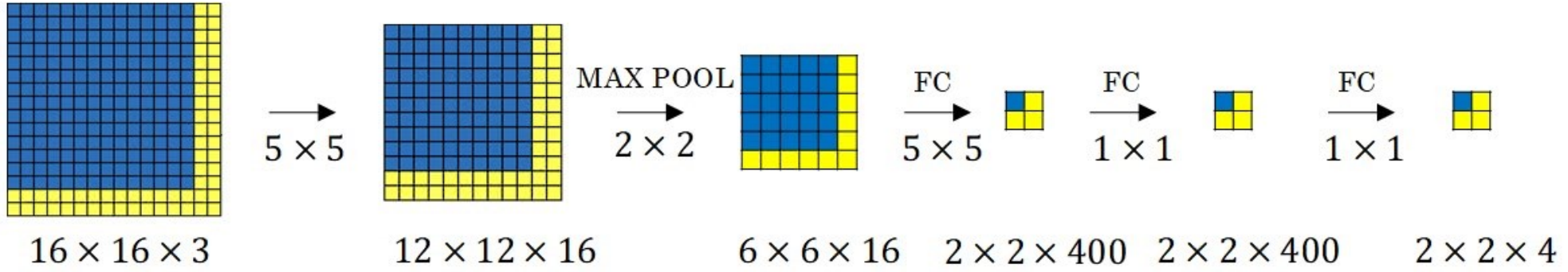




https://pdollar.wordpress.com/tag/detection/

Classification Step - Sliding Window

- Train a CNN to detect objects within an image and use windows of different sizes that we slide on top of it. For each window, we perform a prediction
- For a lot of windows, computational cost is high. The solution is to use sliding window detection computed convolutionally rather than using dense layers



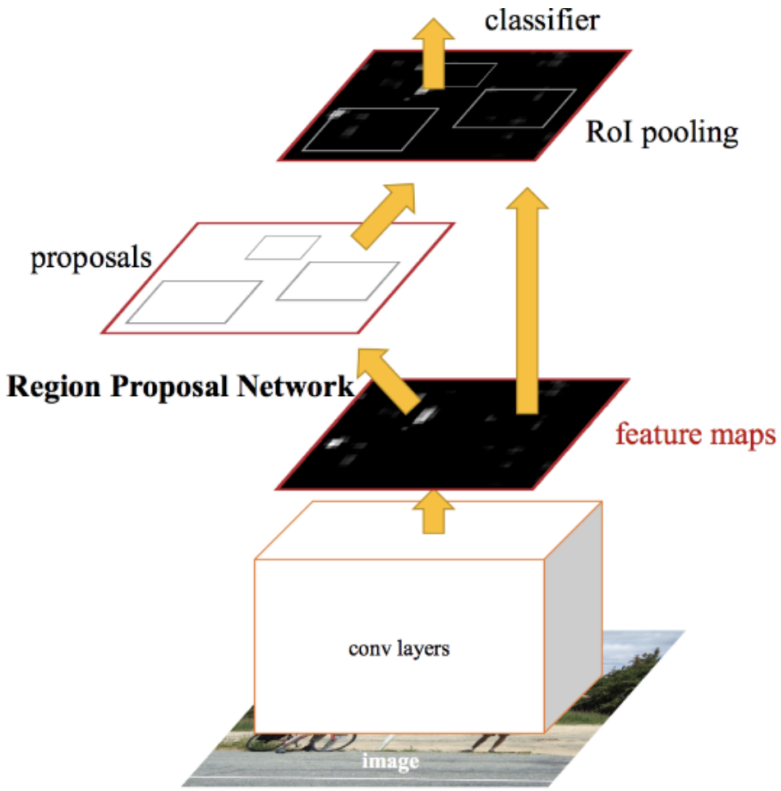
In the image above, we have the input image with shape 16 x 16 pixels and 3 color channels(RGB). Then the **convolutional sliding window** is shown in the upper left blue square with size 14 x 14 and stride of 2.
The upper left corner in the output gives you the upper left 14 x 14 image result if any of the 4 types of target objects are detected in the 14 x 14 section.

RCNN Variations

Algorithm	Approach	Prediction Time	Limitations
RCNN	Uses selective search to generate regions. Extracts around 2000 regions from each image.	40-50 seconds	High computation time as each region is passed to the CNN separately also it uses three different model for making predictions.
Fast RCNN	Each image is passed only once to the CNN and feature maps are extracted. Selective search is used on these maps to generate predictions. Combines all the three models used in RCNN together.	2 seconds	Selective search is slow and hence computation time is still high.
Faster RCNN	Replaces the selective search method with region proposal network which made the algorithm much faster.	0.2 seconds	Object proposal takes time as multiple systems are stacked. The performance of steps depends on how the previous step has performed.

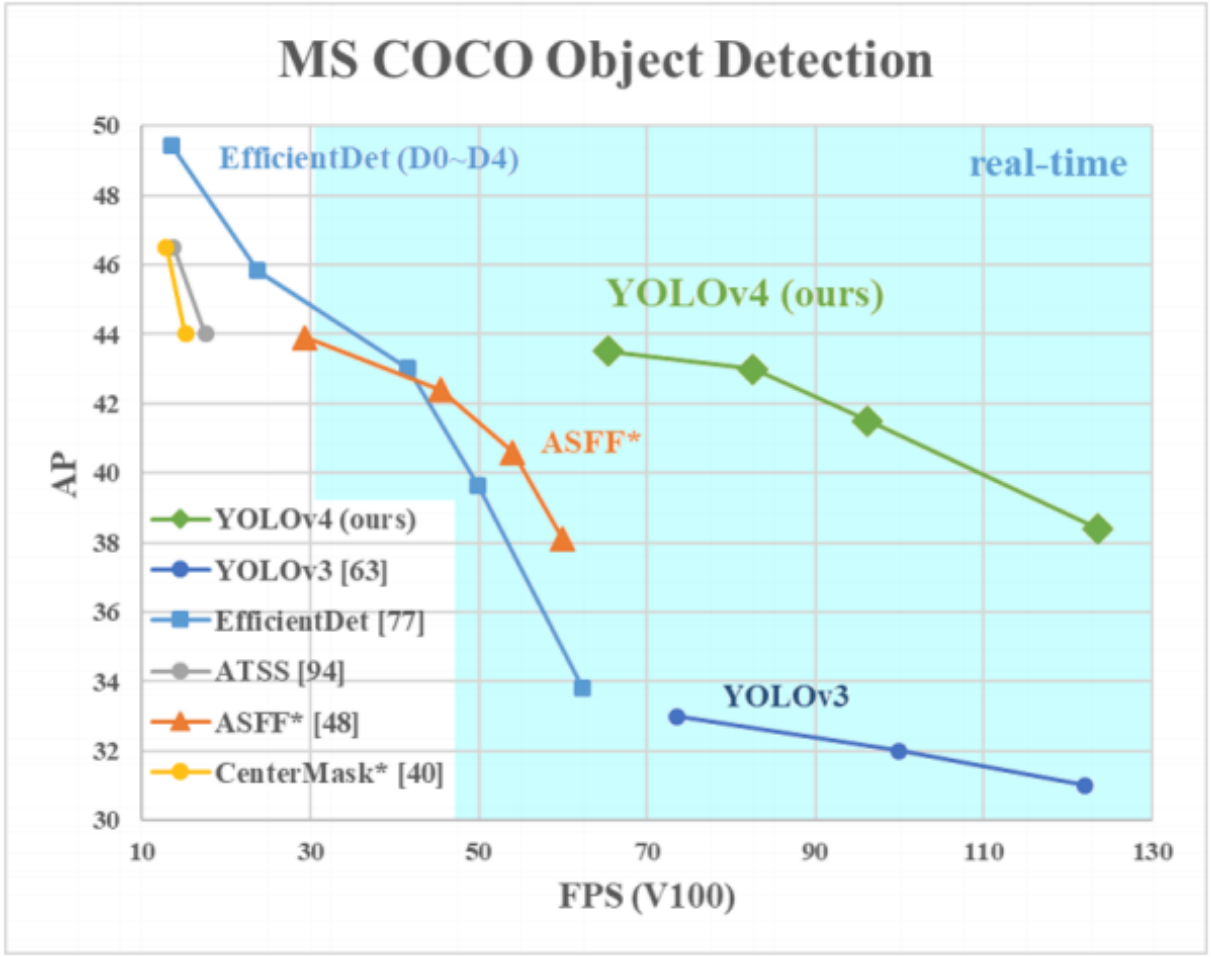
RCNN Pitfalls

- Do not look at the complete image in one go, but focus on parts of the image sequentially.
- Since they have multiple components and require many passes through a single image to extract all the objects, they can be very slow

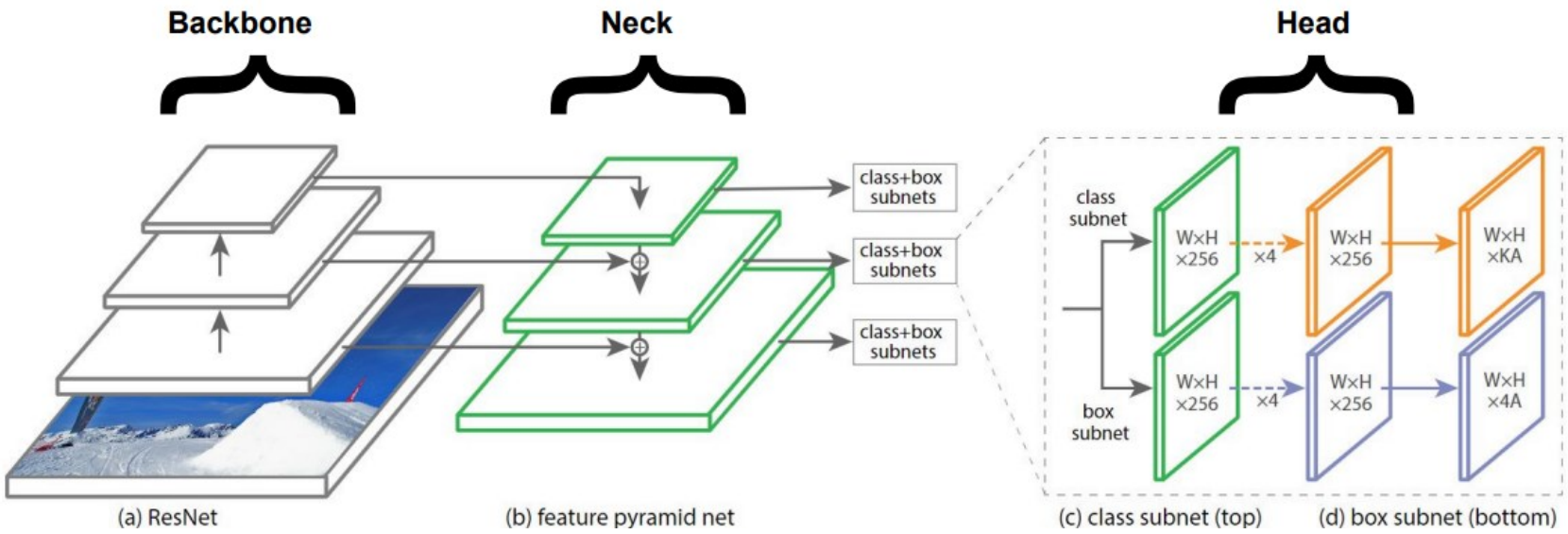


YOLO (2015)

- YOLO (You Only Look Once) is a one-stage object detector and produces SOTA results in real time.
- It avoids the region proposal step and only predicts over a limited number of bounding boxes.
- Performs thresholding to remove multiple detected instances
- Performs non-max suppression to refine the boxes more



Object Detection Architecture



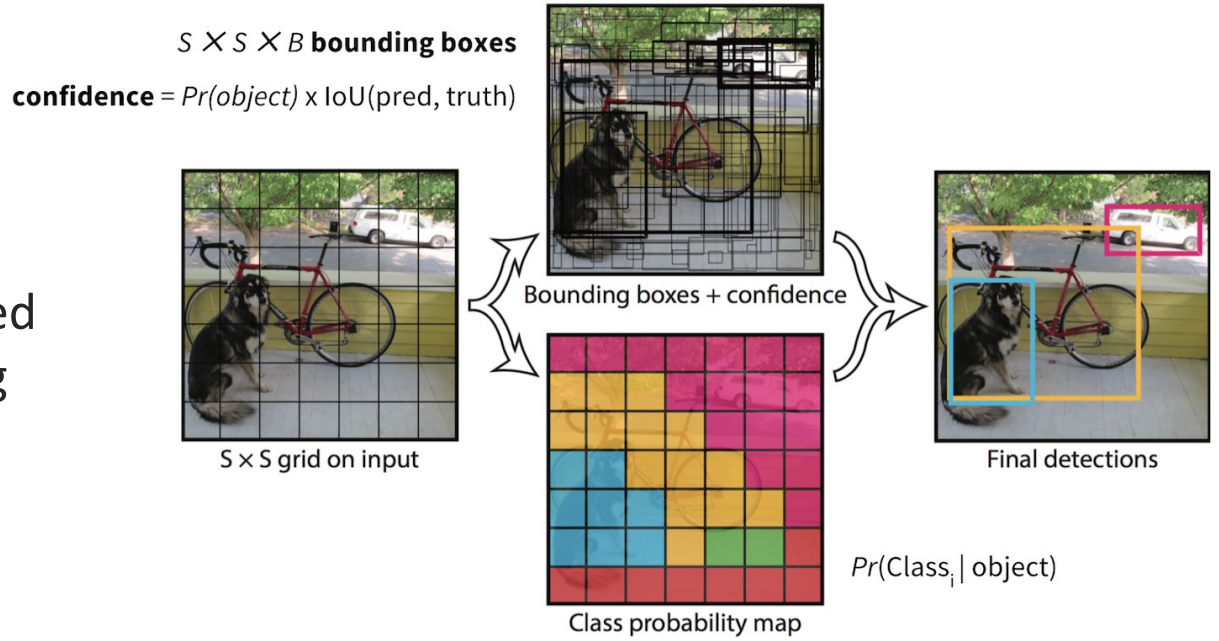
- Backbone - Models such as ResNet, DenseNet, VGG, etc. are used as feature extractors and fine-tuned on the detection dataset.
- Neck - Extra layers that go in between the backbone and head. They are used to extract different feature maps of different stages of the backbone. E.g. FPN[1], PANet[2], Bi-FPN[3], etc..
- Head - Network in charge of the detection (classification and regression) of bounding boxes.

https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50

YOLO Workflow

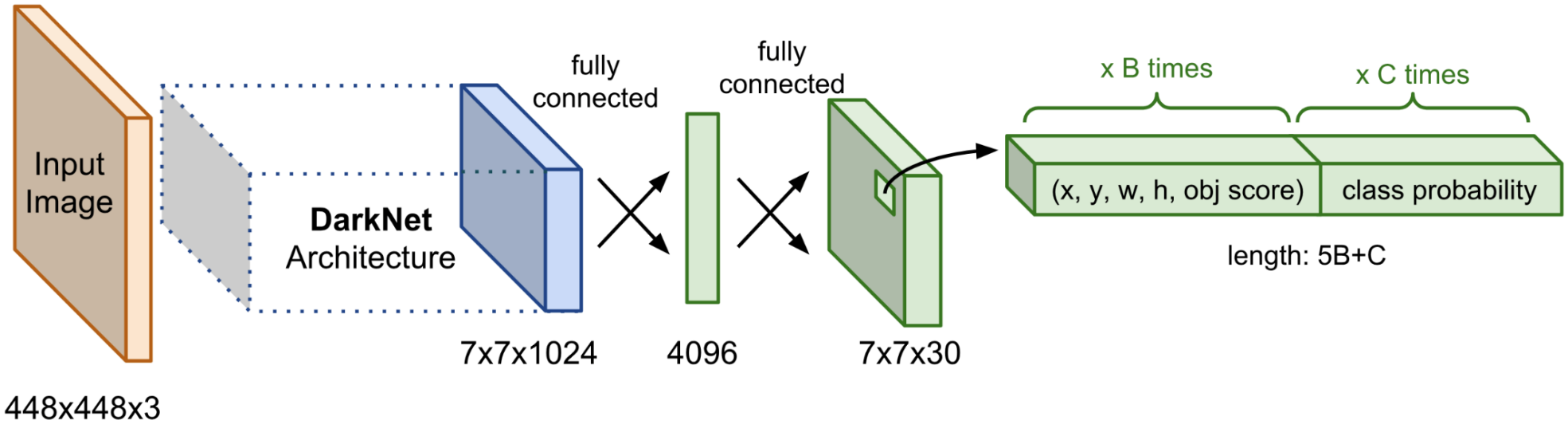
- YOLO divides the input image into an S×S grid.
- For each cell it predicts:
  - A) Location of the B bounding
  - B) A box confidence score
  - C) A probability of the object class conditioned on the existence of an object in the bounding box

- The final layer of the pre-trained CNN is modified to output a prediction tensor of size $S \times S \times (5B+K)$.



YOLO Architecture

- Base model like GoogLeNet with inception module replaced by 1x1 and 3x3 conv layers.
- Final prediction of shape $S \times S \times (5B+K)$ is produced by two fully connected layers over the whole conv feature map.



https://arxiv.org/pdf/1506.02640.pdf

YOLO Loss

- YOLO uses sum-squared error as it is easy to optimize, and the loss function composes of:
- **Classification loss**
  - If an object is detected, the classification loss at each cell is the squared error of the class conditional probabilities for each class
- **Localization loss**
  - The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object.
- **Confidence loss**
  - If an object is detected in the box, the confidence loss is the measure of "objectness" of the box

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$ where

$\mathbb{1}_i^{obj} = 1$ if an object appears in cell i, otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right]$$ where

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2\right]$$

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

λ_{coord} increase the weight for the loss in the boundary box coordinates.

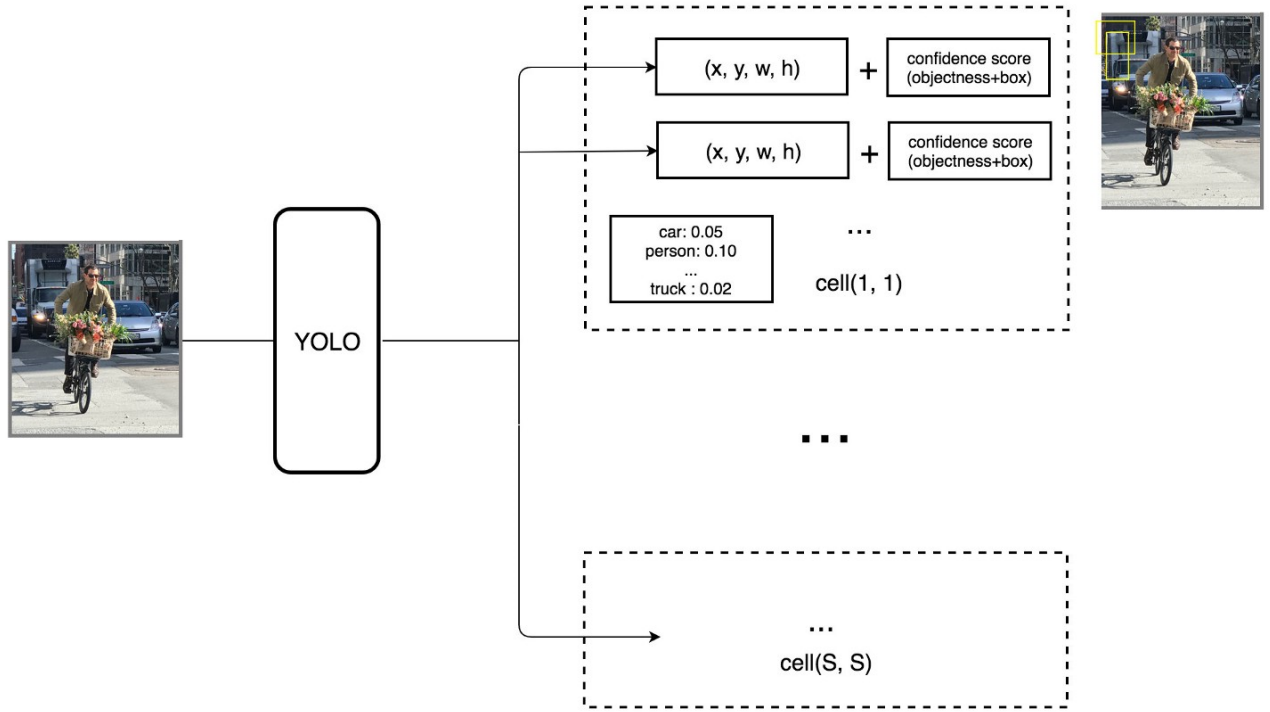
$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$ where

$\hat{C}_i$ is the box confidence score of the box j in cell i.

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

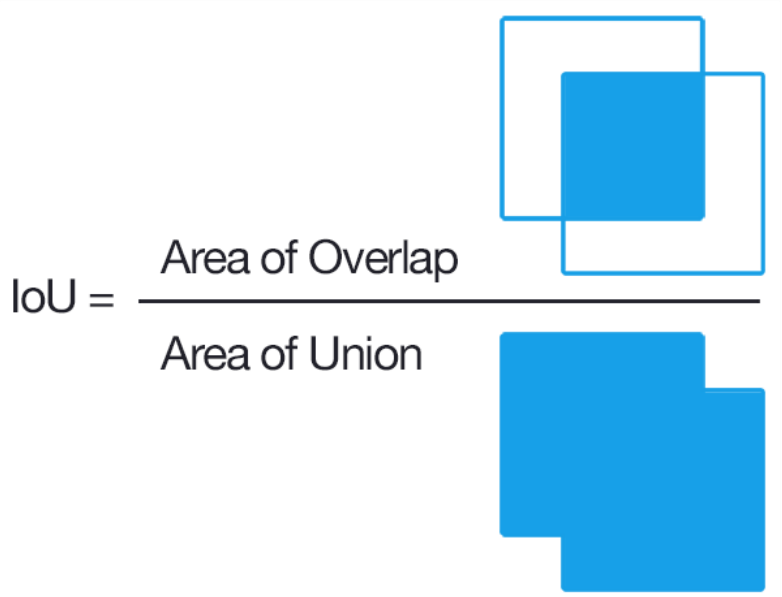
YOLO Example

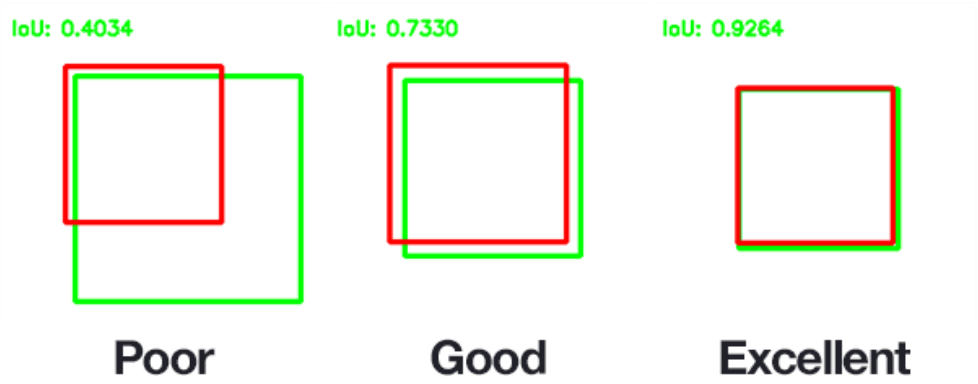
- For PASCAL VOC, YOLO uses 7×7 grids (S×S), 2 boundary boxes (B) and 20 classes (C)
- Each cell has 20 conditional class probabilities
- So, YOLO's prediction has a shape of (S, S, B×5 + C) = (7, 7, 2×5 + 20) = (7, 7, 30)



https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

Intersection over Union





https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Non-maximum Suppression (NMS)

- NMS filters the proposals based on some criteria

Non-maximum Suppression (NMS)

Algorithm 1 Non-Max Suppression

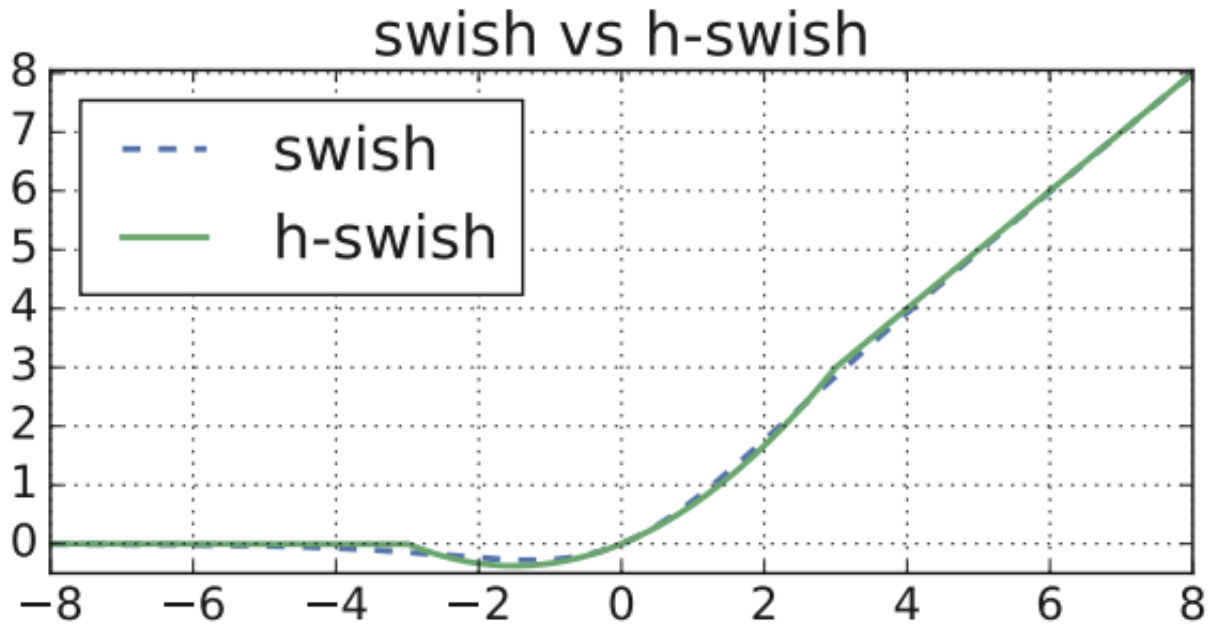
procedure NMS(B,c)
    $B_{nms} \leftarrow \emptyset$
    for $b_i \in B$ **do**
        $discard \leftarrow$ False
        for $b_j \in B$ **do**
            if same$(b_i, b_j) > \boldsymbol{\lambda_{nms}}$ **then**
                if score$(c, b_j) >$ score(c, b_i) **then**
                    $discard \leftarrow$ True
        if not $discard$ **then**
            $B_{nms} \leftarrow B_{nms} \cup b_i$
    return B_{nms}

Hard Swish Activation

- Swish is x times the Sigmoid function
  - Discovered through NAS
  - Like ReLU, Swish is bounded below but unbounded above
  - Unlike ReLU, Swish is smooth and non-monotonic
- Hard Swish replaces the computationally expensive sigmoid with a piecewise linear analogue

$$swish(x) = \frac{x}{(1 + e^{-x})}$$

$$h\_swish(x) = \frac{x\ ReLU6(x+3)}{6}$$

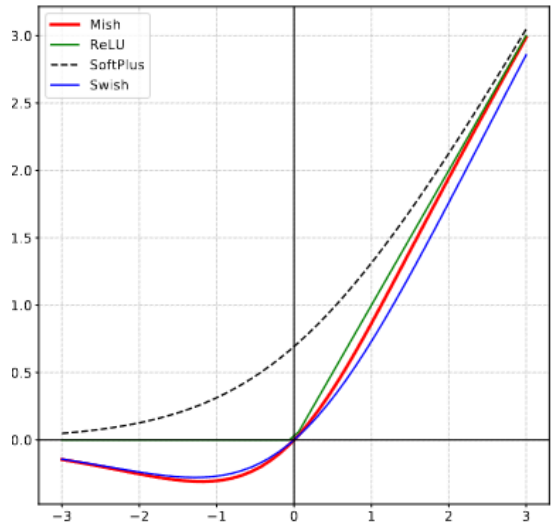


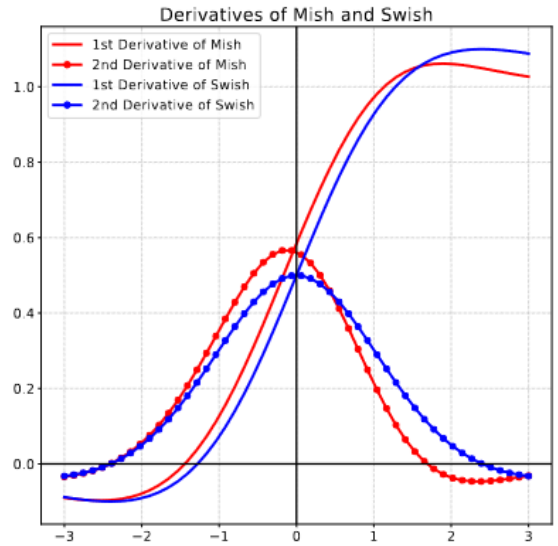
Mish Activation

- Mish
  - Closely related to Swish
  - Due to the preservation of a small amount of negative information, Mish eliminated by design the preconditions necessary for the Dying ReLU phenomenon
  - Outperformed Leaky ReLU on YOLOv4

$$mish(x) = x.\tanh(softplus(x))$$

$$mish(x) = x.\tanh(\ln(1 + e^x))$$



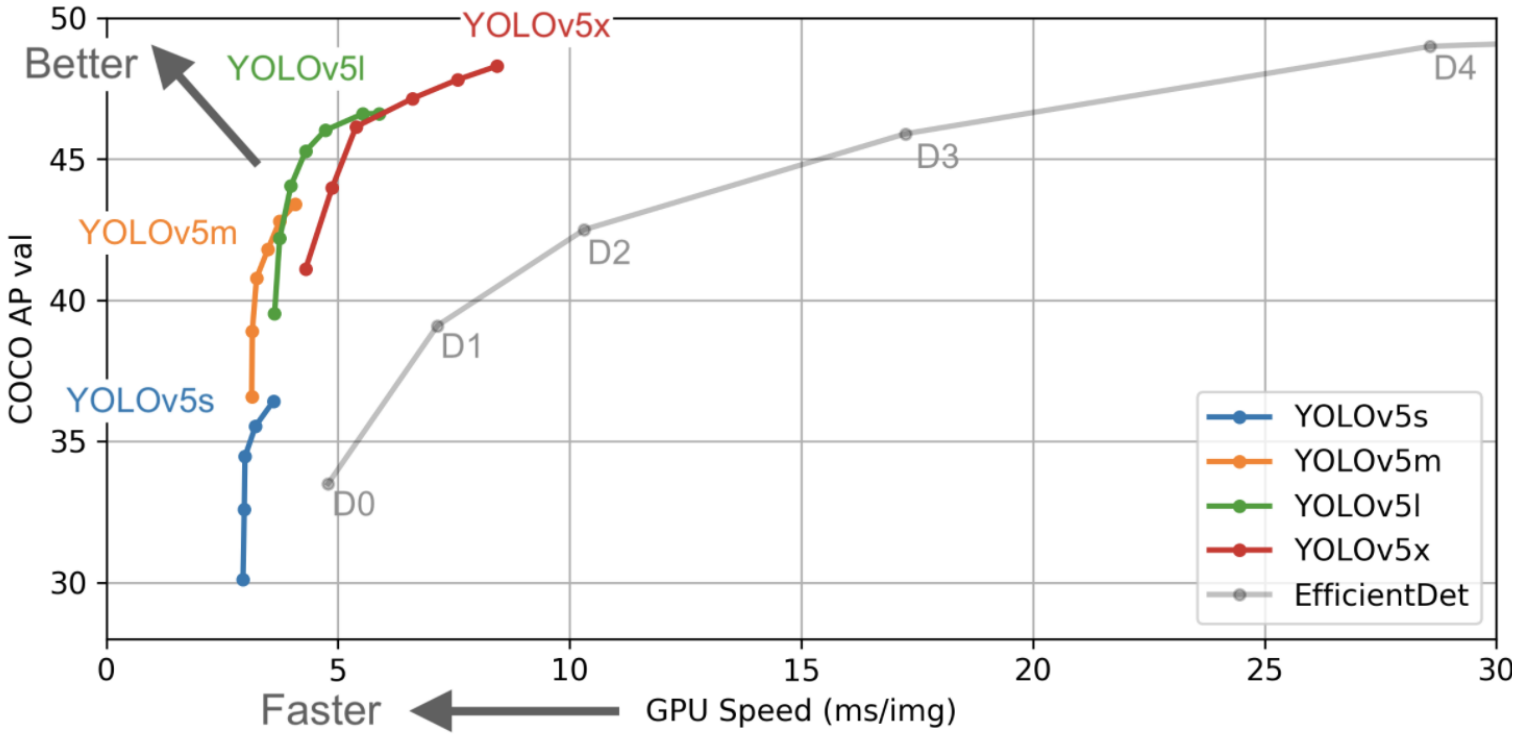


https://www.bmvc2020-conference.com/assets/papers/0928.pdf

YOLO Versions

Version	Architecture	Main Improvements
YOLO	Grid division is responsible for detection, confidence loss;	Allows real-time detection, struggles with small objects.
YOLO V2	Anchor with K-means added, two-stage training, full convolutional network;	Better with small objects.
YOLO V3	Modern CNNs as backbone. Multi-scale detection by using FPN;	Even better with small objects. Reduces computational complexity.
YOLO V4	CSP Networks, MISH activation function, data enhancement Mosaic/Mixup, GIOU(Generalized Intersection over Union) loss function; Dropbox Regularization.	Better image augmentation
YOLO V5	Flexible control of model size, application of Hardswish activation function, and data enhancement.	Ease of use and deploy on edge devices Quick to run experiments

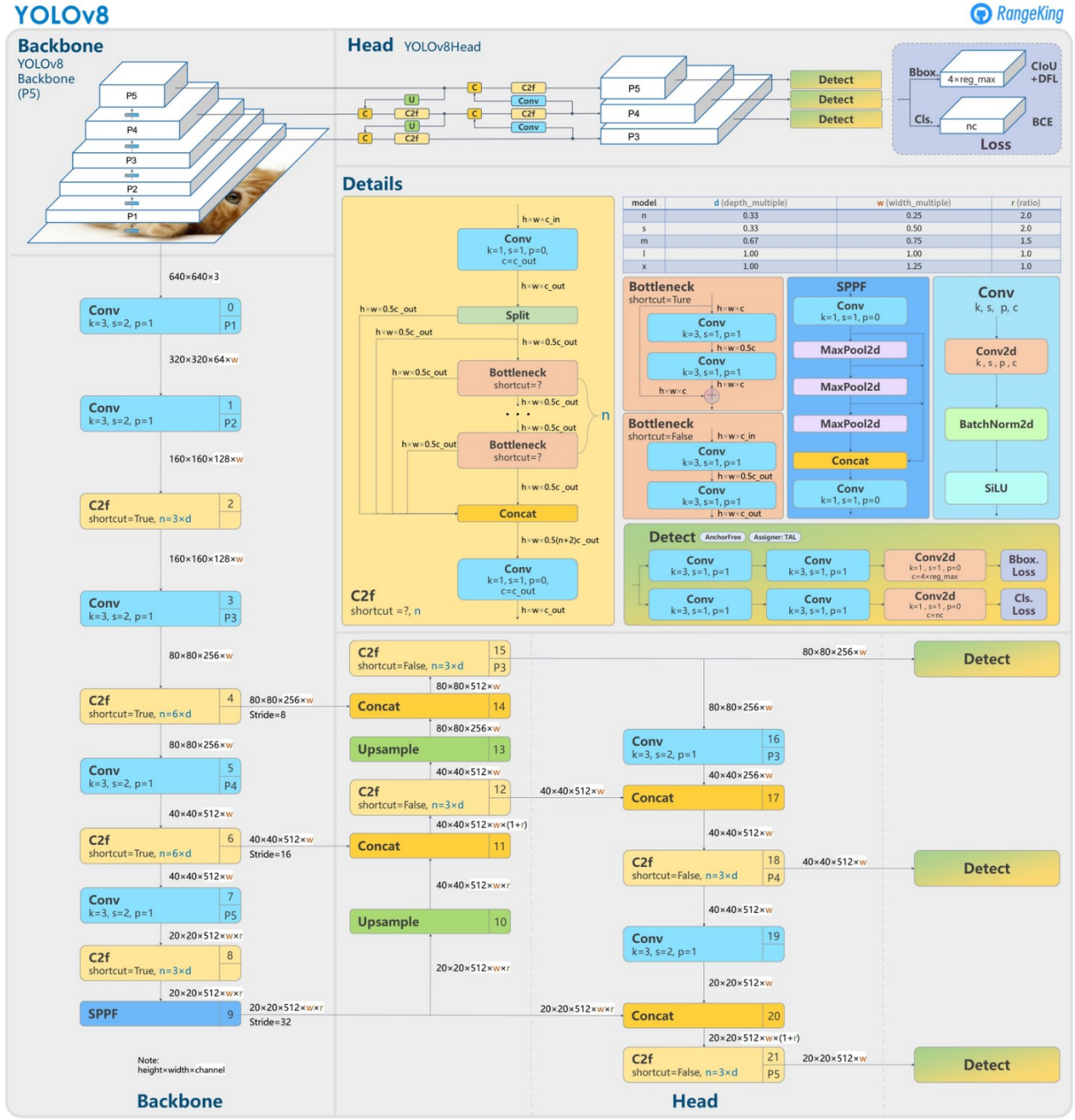
YOLO Versions



Source: Roboflow

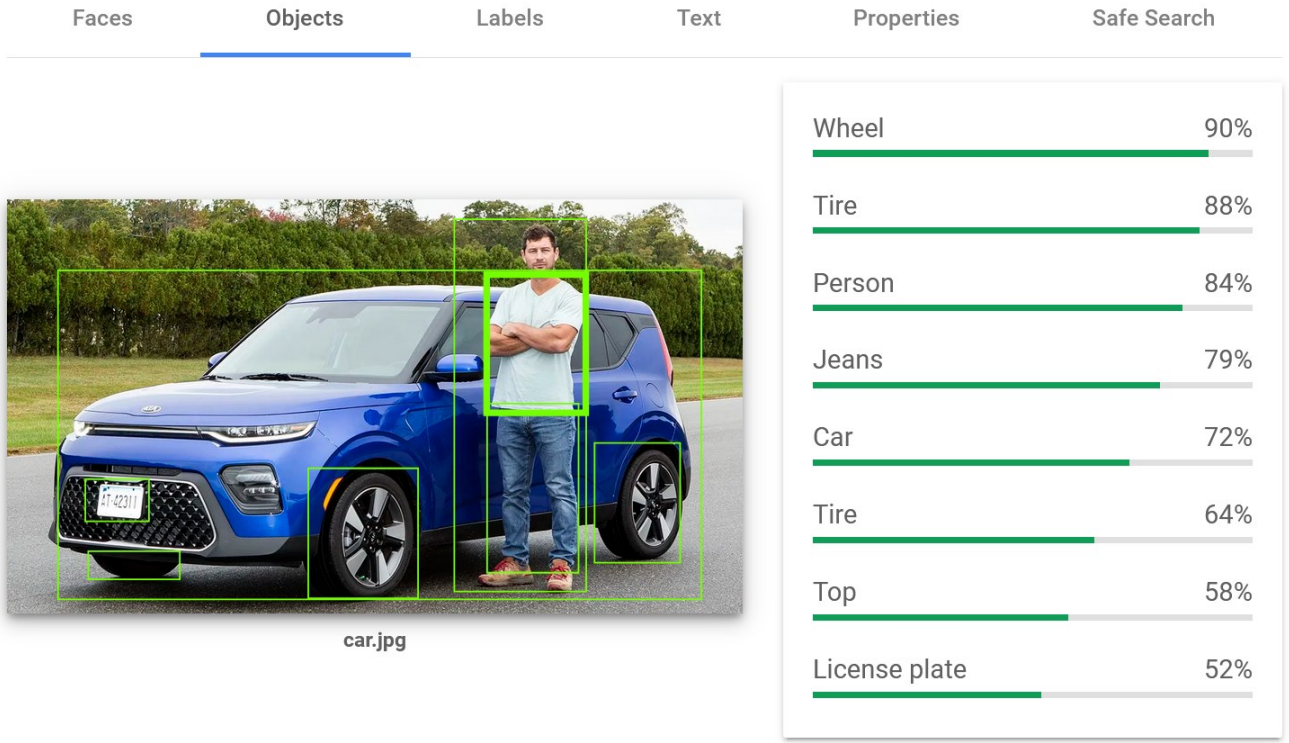
Name	Year	Type	Dataset	mAP	Inference rate (fps)
R-CNN	2014	2D	Pascal VOC	66%	0.02
Fast R-CNN	2015		Pascal VOC	68.80%	0.5
Faster R-CNN	2016		COCO	78.90%	7
YOLOv1	2016		Pascal VOC	63.40%	45
YOLOv2	2016		Pascal VOC	78.60%	67
SSD	2016		Pascal VOC	74.30%	59
RetinaNet	2018		COCO	61.10%	90
YOLOv3	2018		COCO	44.30%	95.2
YOLOv4	2020		COCO	65.70%	62
YOLOv5	2021		COCO	56.40%	140
YOLOR	2021		COCO	74.30%	30
YOLOX	2021		COCO	51.20%	57.8
Complex-YOLO	2018	3D	KITTI	64.00%	50.4
Complexer-YOLO	2019		KITTI	49.44%	100
Wen et al.	2021		KITTI	73.76%	17.8
RAANet	2021		NuScenes	62.00%	

YOLOv8


YOLOv8
RangeKing
Backbone
YOLOv8 Backbone (P5)
Head YOLOv8Head
Detect
Bbox. 4×reg_max CIoU +DFL
Cls. nc BCE
Loss
Details
C2f shortcut =?, n
Bottleneck shortcut=Ture
Bottleneck shortcut=False
SPPF
Conv k, s, p, c
Conv2d
BatchNorm2d
SiLU
Detect AnchorFree Assigner: TAL
Bbox. Loss
Cls. Loss
model | d (depth_multiple) | w (width_multiple) | r (ratio)
n | 0.33 | 0.25 | 2.0
s | 0.33 | 0.50 | 2.0
m | 0.67 | 0.75 | 1.5
l | 1.00 | 1.00 | 1.0
x | 1.00 | 1.25 | 1.0
640×640×3
Conv k=3, s=2, p=1 0 P1
320×320×64×w
Conv k=3, s=2, p=1 1 P2
160×160×128×w
C2f shortcut=True, n=3×d 2
160×160×128×w
Conv k=3, s=2, p=1 3 P3
80×80×256×w
C2f shortcut=True, n=6×d 4
80×80×256×w Stride=8
Conv k=3, s=2, p=1 5 P4
40×40×512×w
C2f shortcut=True, n=6×d 6
40×40×512×w Stride=16
Conv k=3, s=2, p=1 7 P5
20×20×512×w×r
C2f shortcut=True, n=3×d 8
20×20×512×w×r
SPPF 9
20×20×512×w×r Stride=32
Note: height×width×channel
Backbone
C2f shortcut=False, n=3×d 15 P3
Concat 14
Upsample 13
C2f shortcut=False, n=3×d 12
Concat 11
Upsample 10
Conv k=3, s=2, p=1 16 P3
Concat 17
C2f shortcut=False, n=3×d 18 P4
Conv k=3, s=2, p=1 19
Concat 20
C2f shortcut=False, n=3×d 21 P5
Detect
Head

Cloud Object Detection API

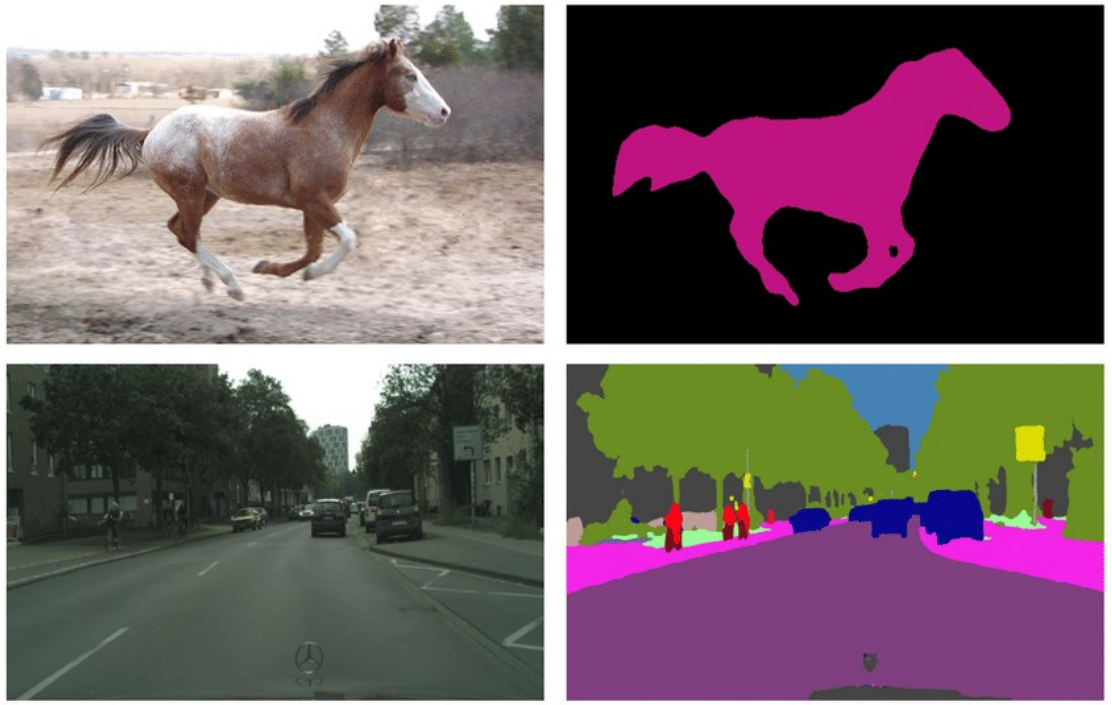
1. Create Service Account
2. Add API permissions
3. Download credentials
4. Install cloud libraries
5. Invoke API



https://cloud.google.com/vision/docs/drag-and-drop

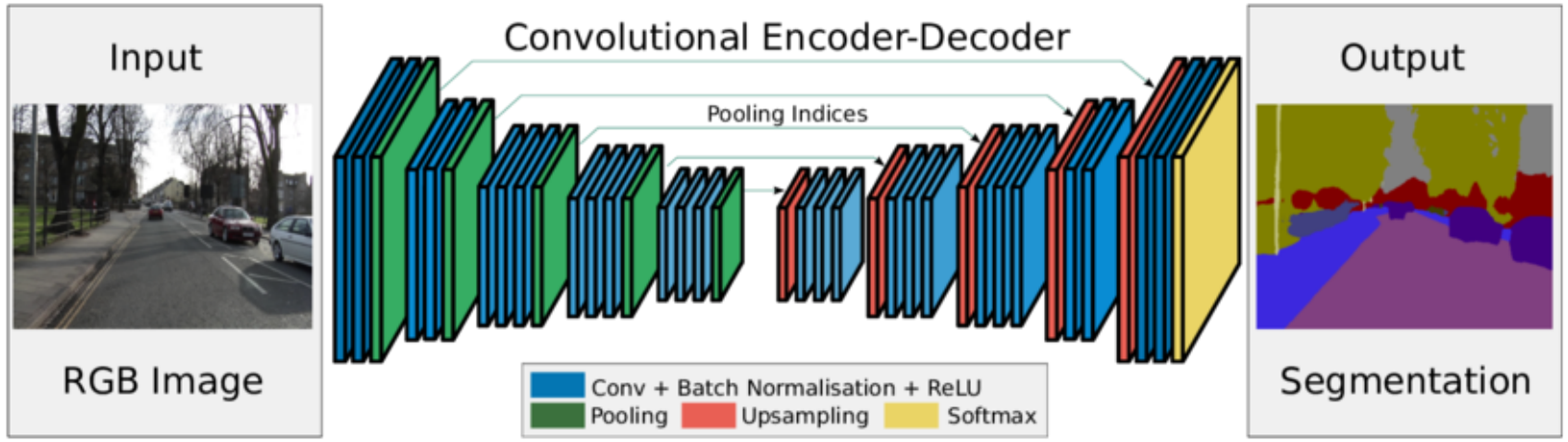
Segmentation

- Requires discrimination at pixel level and a mechanism to project the discriminative features learnt at different stages of the encoder onto the pixel space.



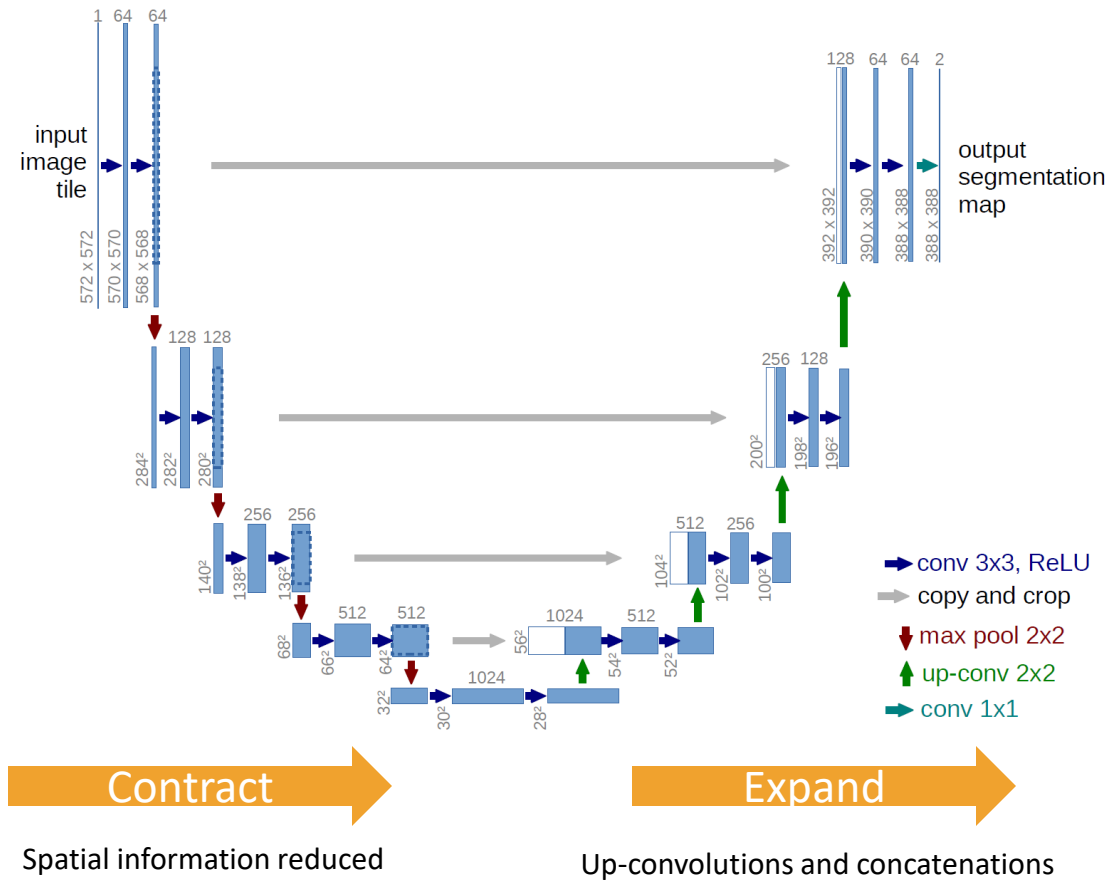
Encoder-Decoder Architecture

- Encoder network is topologically identical to multi-layered CNN.
- Decoder maps the low-resolution encoder feature maps to full input resolution feature maps for pixel-wise classification. Followed by a pixel-wise classification layer



U-Net (2015)

- Developed at University of Freiburg for biomedical image segmentation
- Architecture
  - **Encoder** : usually a pre-trained classifier to encode the input image into feature representations at multiple levels.
  - **Decoder**: up-sampling and concatenation followed by regular convolutions.
  - It semantically projects the discriminative features (lower resolution) learned by the encoder onto the pixel space (higher resolution) to get a dense classification.

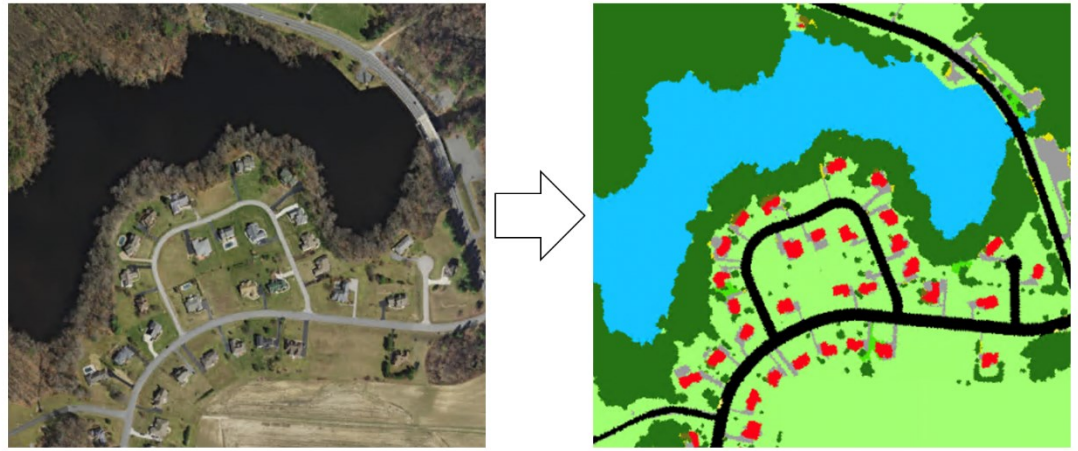


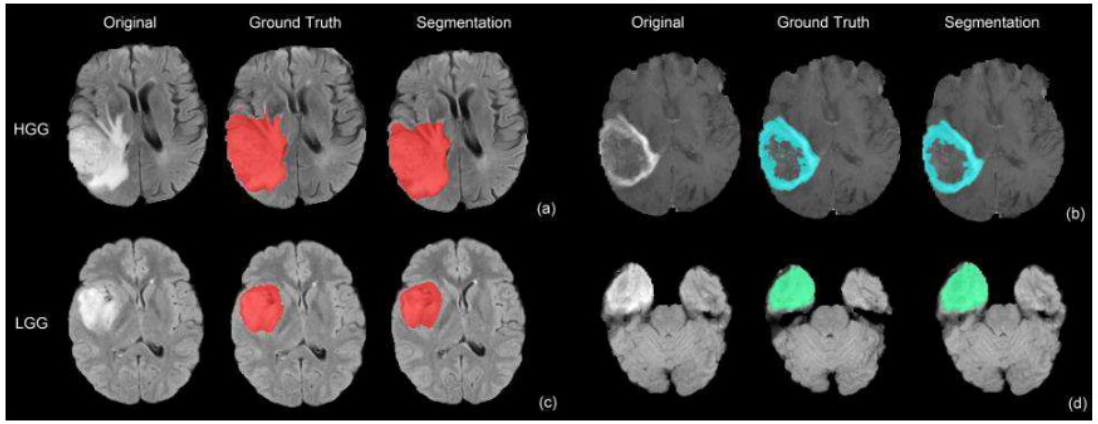
https://arxiv.org/abs/1505.04597

https://developers.arcgis.com/python/guide/how-unet-works/

U-Net (2015)

- Architecture
  - Contracting path (down-sampling encoder), where the width and heights of the feature maps are shrunk while the channel expands by a factor of 2 until it reaches 1024 (*typically the maximum recommended level for CNNs*)
  - Bottleneck as a “turning point”
  - Expanding path (Up-sampling decoder), where widths and heights of the feature maps are expanded to the mask’s dimension
  - Large number of feature channels in the upsampling part, allow the network to propagate context information to higher resolution layers

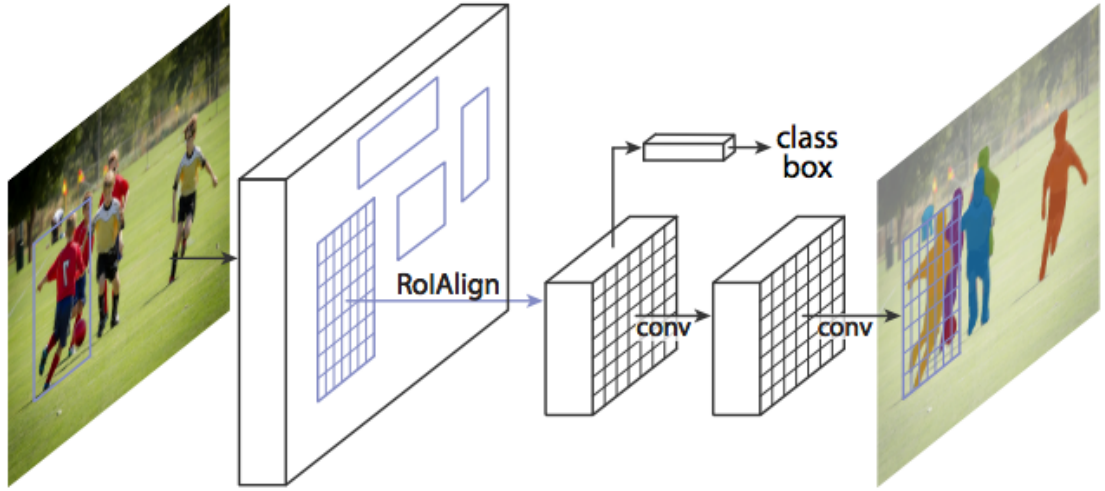




https://developers.arcgis.com/python/guide/how-unet-works/

Mask RCNN (2017)

- Adds a branch to Faster R-CNN that outputs a binary mask that says whether or not a given pixel is part of an object
- The branch (in white), is a Fully Convolutional Network (FCN) on top of a CNN based feature map
  - **Inputs**: CNN Feature Map.
  - **Outputs**: Matrix with 1s on all locations where the pixel belongs to the object and 0s elsewhere (this is known as a binary mask).



https://arxiv.org/abs/1703.06870
https://github.com/matterport/Mask_RCNN