YOLO

You only **look** once.

Review: Image Classification vs Object Detection

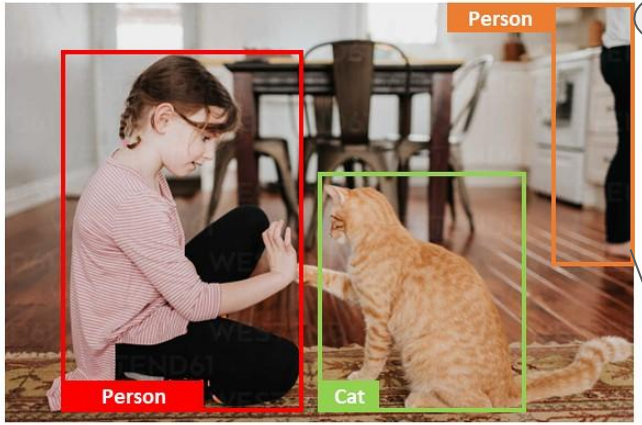
- Input: Image
- Output: Logits for Classification, Bounding Boxes for Detection

Image Recognition

```
tensor([[0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0],
```

Dog

Object Detection



Bounding box is vectors. Essentially the same thing!

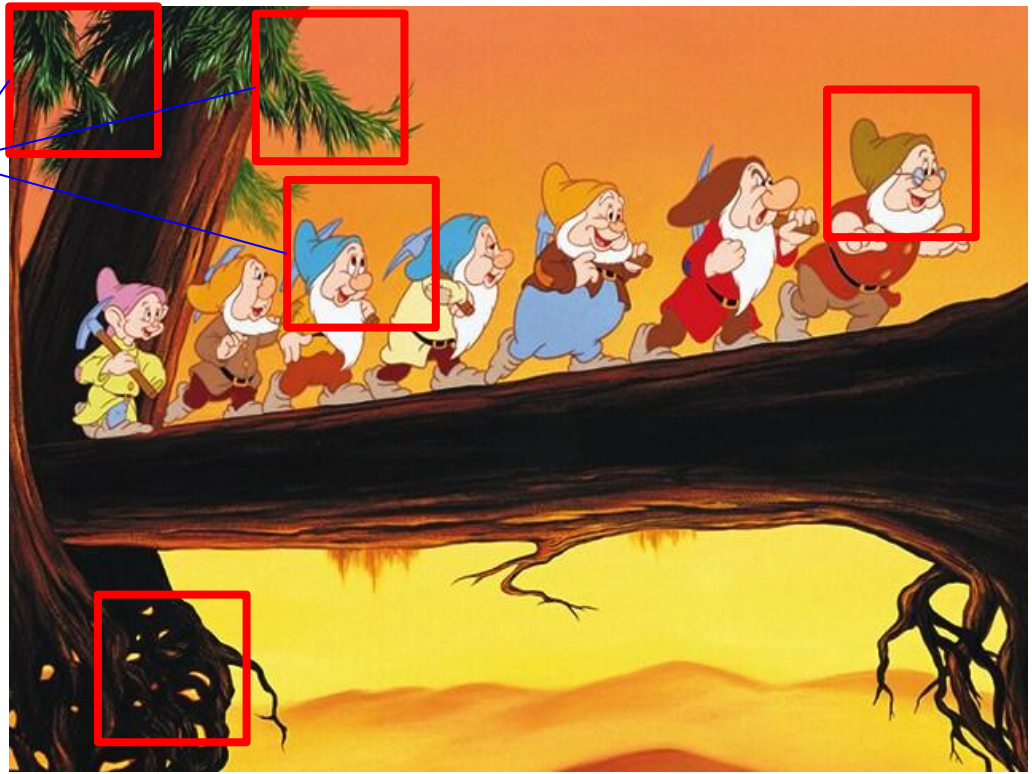
Presentation of ‘bounding box’

- (x, y, w, h)
- (p1, p2, p3, p4)
- (cx, cy, w, h)
- (x, y, w, h, angle)

…

Review: Object Detection as Classification

Binary Classification: If the area contains a head?



Review: Object Detection as Classification

A more precise way:
Sliding Window
Classification

The main idea of RCNN/ Fast RCNN

Problem: Huge time cost

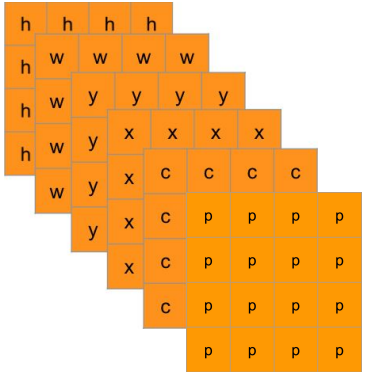
May use millions of sliding windows, if higher accuracy is needed.

Key Limitations Before YOLO

- Computationally expensive (THE biggest problem): Impossible for real-time applications because sliding window requires many excess box predictions.
- Complex pipeline: Require input standardization before training, and manual sample generation is also necessary
- Severe data imbalance for binary classification: Too much background, few objects (dwarf’s head)

YOLO's Idea

- Predict x, y, w, h, c, p instead of box vector + logits
  - $448 \times 448 \rightarrow$ CNN $\rightarrow$ FC $\rightarrow$ $S^2 \times (5, C)$
  - $p_i \in R^C$, where i is each grid cell.
  - Turn classification into a **regression** problem,
  - Detect multiple targets in one image
  - Use one box for the target in one certain region of image
  - Divide the image into $S \times S$ grid cells



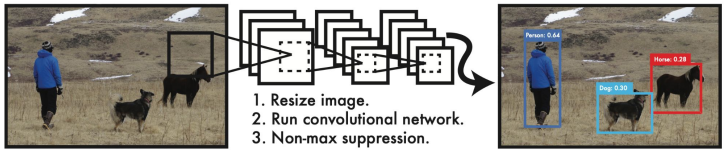
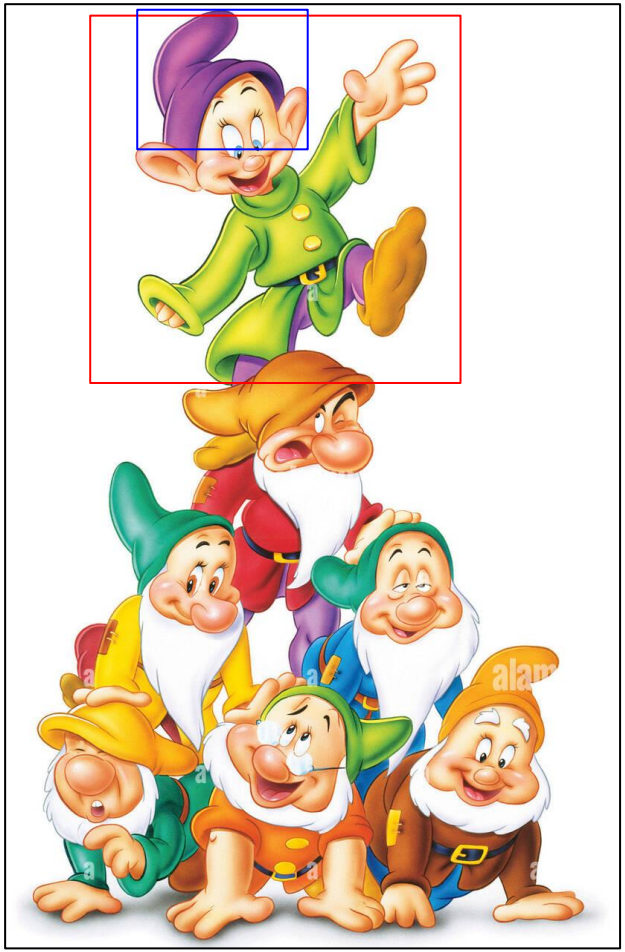


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

Multi-Object Grid

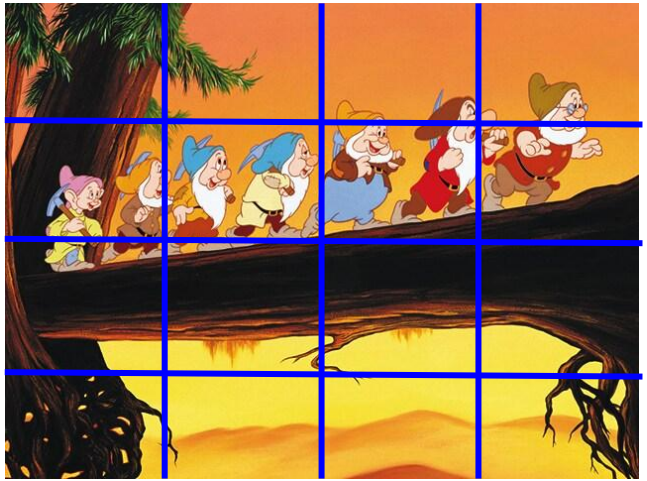
- Multiple boxes for every grid cell $(B = 2)$
  - i.e. 2 x, y, w, h, c outputs for 1 grid cell
  - The output would be $S^2 \times (B \times 5, C)$
- One box is responsible for each object
- Class probabilities $p_i(c)$ represent the probability that class c is present in grid cell i, with multiple true classes possible per cell



Non-Maximum Suppression (NMS)

Main idea for NMS:

- If 2 boxes have high overlap (IoU), they likely represent the same target.
- Keep the box with the highest confidence in <u>each grid cell.</u>



We'll get 16 boxes, but we only have 7 dwarfs…
How do we choose 7 boxes for 7 dwarfs?

- Clustering?
- **NMS!**

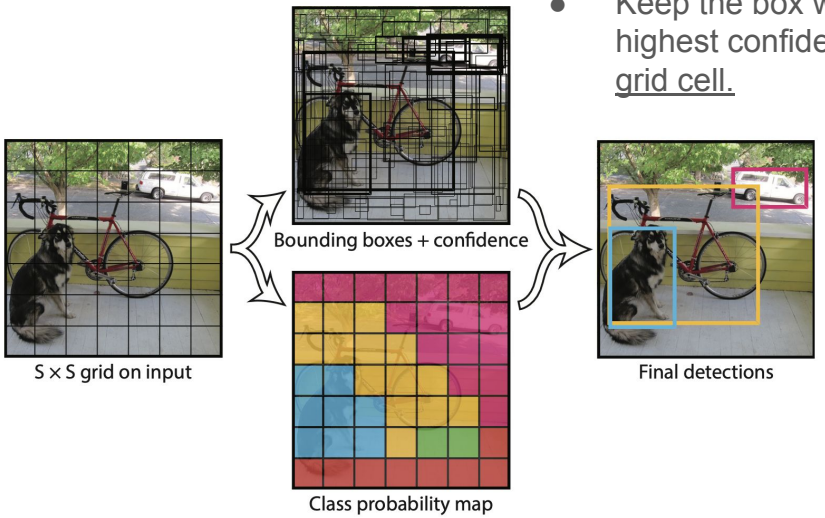
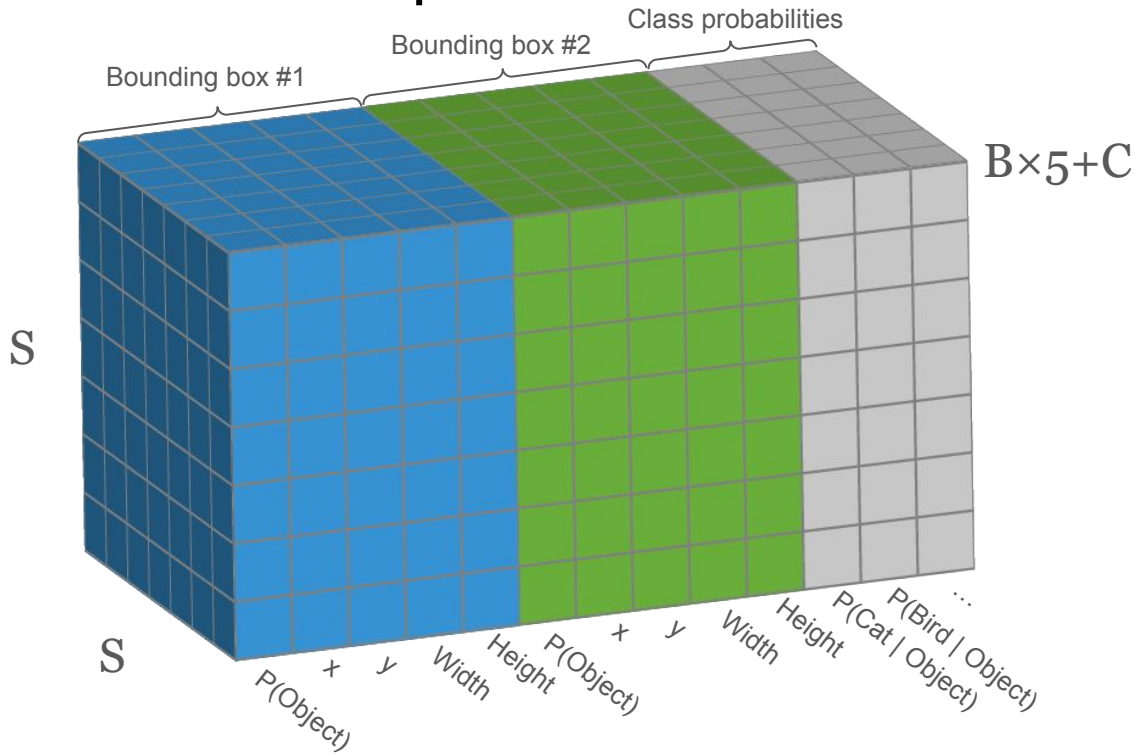


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

Network Overview: Outputs



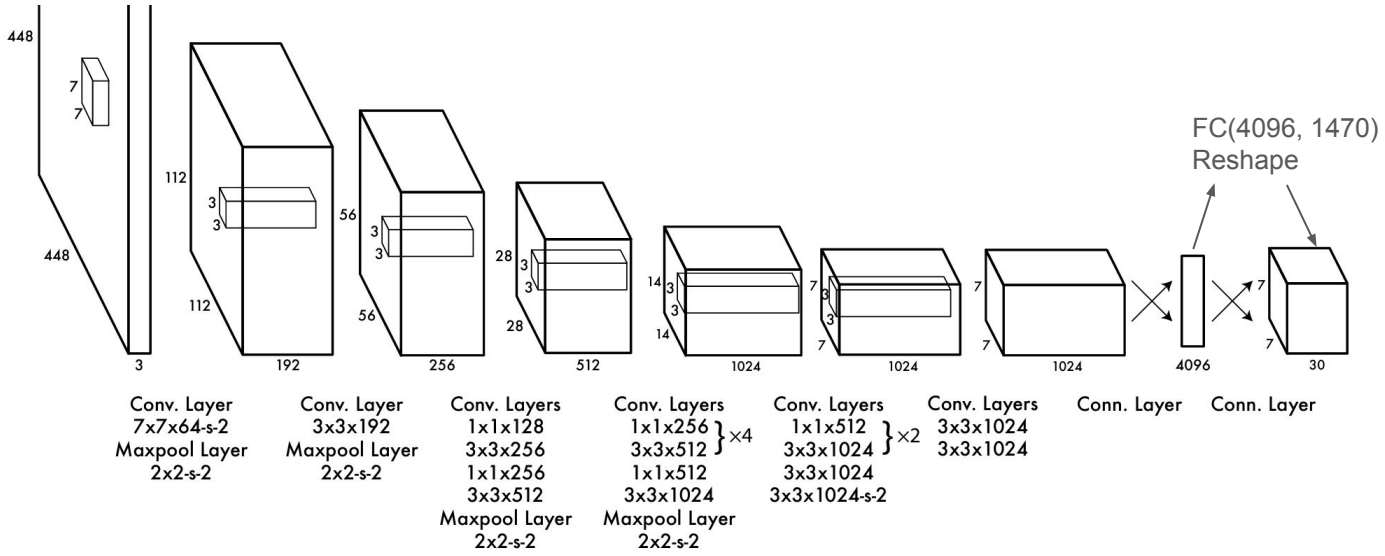
Note:

- (x, y) are relative to the bounds of the grid cell
- (w, h) are relative to the image

Network Overview: Architecture

Backbone: custom pretrained convolutional neural network

Theoretical concepts **only influence shape of inputs and outputs** of the layers.



Network Overview: Outputs

Pseudocode

For each **grid cell (S×S):**

Predict C **class probabilities** $p_{\text{cell}}(c)$

For each **potential bounding box (B):**

Predict box coordinates and object confidence as a single tensor x, y, w, h, c

Return $S \times S \times (B * 5 + C)$

Where

Confidence $c = \hat{P}(object)$ Whether **an** object exists in a bounding box

Class Probability $\hat{p}_i(c) = P_i(c|object) * \hat{P}_{\text{object}}$ Given that **an** object exists, what class

Network Overview: Inference

For every **cell** prediction $S \times S \times (B * 5 + C)$:

Discard bounding boxes with confidence c below classification threshold

Run NMS on bounding boxes with similar IOU

Label bounding boxes with class(es) with highest probability / top probabilities for their grid cell, $p_{\text{cell}}(c)$

Training: Losses

$$\mathcal{L} = \lambda_{\text{coord}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] \quad 1$$

$$+ \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \quad 2 \qquad \left(= \lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{noobj}} (1 - \hat{C}_i)^2 \right)$$

$$+ \lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad 3 \qquad \left(= -\lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{noobj}} \hat{C}_i^2 \right)$$

$$+ \sum_{i=1}^{S^2} 1_{i}^{\text{obj}} \sum_{c=1}^{C} (p_i(c) - \hat{p}_i(c))^2 \quad 4$$

1. **Bounding Boxes:**
   - **For every grid cell, get MSE of every bounding box's coordinates**
   - **Square root of width and height used to penalise bigger boxes less.**
   - **Scale up coordinate loss with** $\lambda_{\text{coord}} = 5$
2. **Object Detection:**
   - **For every grid cell, get MSE of every box's between predicted 'confidence' and ground truth of 1.**
   - **Confidence:** $\hat{C}_i = \hat{P}_{\text{object}} \times IOU \qquad C_i = 1$

Training: Losses

$$\mathcal{L} = \lambda_{\text{coord}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] \quad 1$$

$$+ \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \quad 2 \qquad \left(= \lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{noobj}} (1 - \hat{C}_i)^2 \right)$$

$$+ \lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad 3 \qquad \left(= -\lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{ij}^{\text{noobj}} \hat{C}_i^2 \right)$$

$$+ \sum_{i=1}^{S^2} 1_{i}^{\text{obj}} \sum_{c=1}^{C} (p_i(c) - \hat{p}_i(c))^2 \quad 4$$

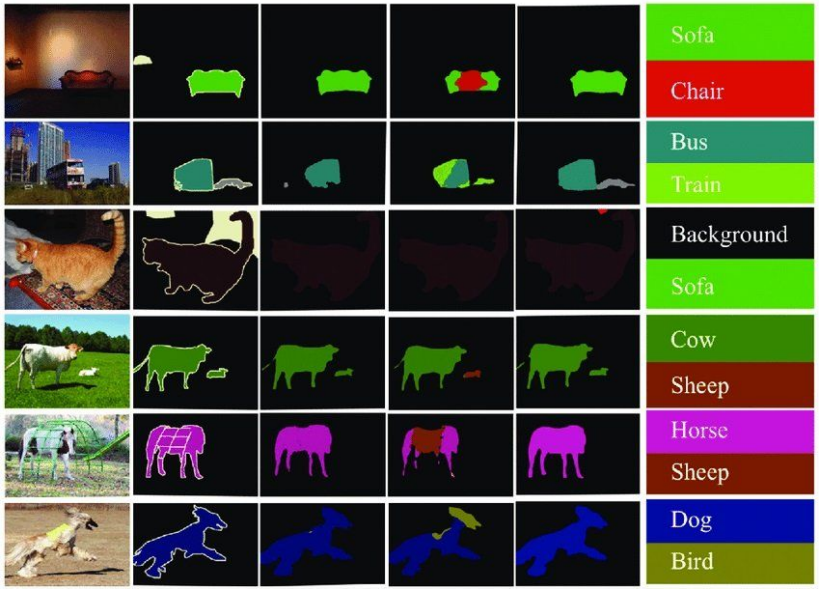
3. **False Positives:**
   - **For every grid cell, get MSE of every box's 'confidence' from ground truth of 0**
   - **Scale down with λ_{noobj} due to class imbalance – far more empty boxes $\lambda_{\text{noobj}} = 0.5$**
4. **Class probabilities:**
   - **For every grid cell, get MSE loss between predicted and true class probabilities.**
   - $\hat{p}_i(c) = \hat{P}_i(c|object)$

     $p_i(c) = 1$ **if class present in cell, else** 0

Training: Data

Pascal VOC (~10K images, ~20K objects, 20 classes)

S=7, B=2, C=20



Training: Layers

- *Conv* includes single convolution layer and an activation layer (introduced in v2),
- using the LeakyReLU (neg slope=0.1) except for the last activation function
- CNN pretrained on half-resolution ImageNet
- Dropout 0.5 to prevent layer coadaptation.
- Data augmentation – random scaling and translations of up to 20% of the original image size, and random adjustment the exposure and saturation by up to a factor of 1.5.

Training: LR Schedule

- 3-stage LR schedule as model diverges for high LR at early epochs

YOLO v1 Advantages

- Real-time (155 FPS YOLO Small / 45 FPS YOLO)
- Small YOLO was the fastest detection method on PASCAL at the time, with more than twice the mAP of the previous real-time detection.
- SOTA on certain benchmarks.
- Using YOLO to eliminate R-CNN errors boosts in mAP without much increase in cost.
- Generalises well to artwork, showing good generalisability.

	mAP	Combined	Gain
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	**66.9**	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	**75.0**	**3.2**

YOLO v1 Limitations

- Recall: Many targets can’t be found
  - This is since 2 bounding boxes are predicted per grid cell, making it hard for the network to discern large numbers of clustered objects.
- Localisation: The predicted bounding boxes are not accurate.
  - Biggest contribution to YOLO’s errors, bigger than other sources combines.
  - Struggles to generalise bounding boxes to novel object shapes, potentially due to coarseness introduced by downsampling input images to half resolution.
  - Loss function is still treats small errors in large boxes to be equivalent to small errors in small boxes despite square-rooting of width and height.

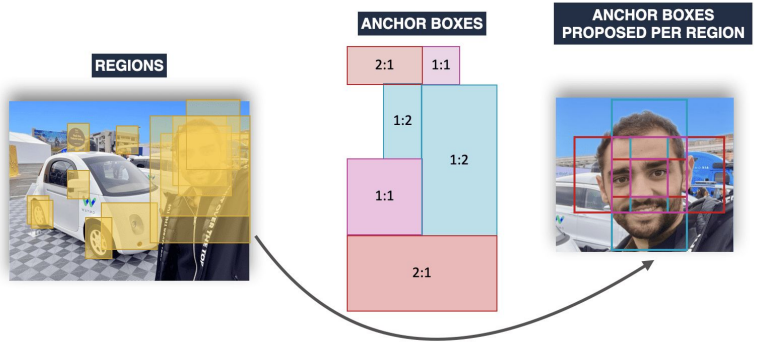
YOLO v2: General Improvements

- Object Knowledge**:** Named “YOLO-9000” as it goes from 20 to 9000 object classes.
- Batchnorm**:** On all convolutional layers, improving convergence and replacing dropout.
- Cropped Input**:** Cropped inputs to 416x416 so that the centre lies in the centre of a grid cell.
- Fully Convolutional: Eliminated fully connected layers.
- High Resolution Backbone**:** Backbone fine-tuned at 448×448 (source) instead of 224×224 by eliminating maxpool. Uses Darknet instead of GoogLeNet.

YOLO v2: **Key** Improvements

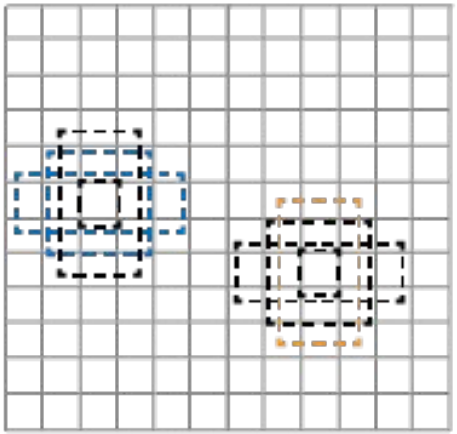
- **Anchor Boxes with Clustered Priors:** Uses anchor boxes with priors from k-means clustering.
- **Bounded Box Predictions:** Use of sigmoid parameterised offsets for box predictions, giving mode stable training.
- **Passthrough Layer:** Extracts image features at multiple scales through passthrough layer.
- **Multi-Scale Training:** Fully convolutional structure allows the model to be trained with images of various resolutions, making it robust to input size.
- **Hierarchical Class Labels**: Allows multiple and related true classes for each object using WordTree
- **DarkNet:** Uses a custom GoogLeNet-based model, Darknet-19 (19 conv layers), incorporating the latest advances (incl batchnorm), to speed up convergence

Review: Anchor Boxes

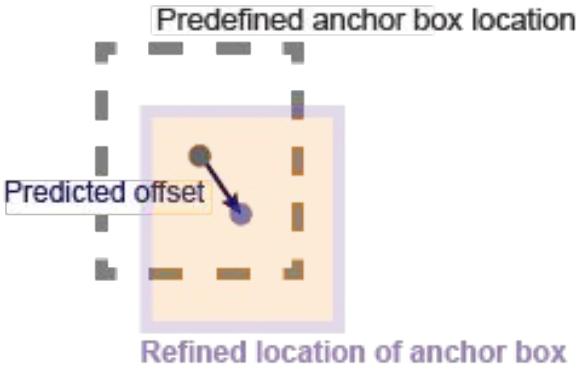


- Originated from R-CNN
- Predefined template of boxes, for model reference in training
- Instead of predicting coordinates, we predict offset *t* to get final bounding box

Ground truth image and bounding boxes

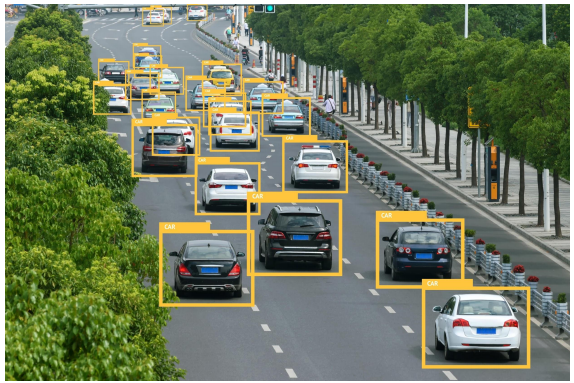


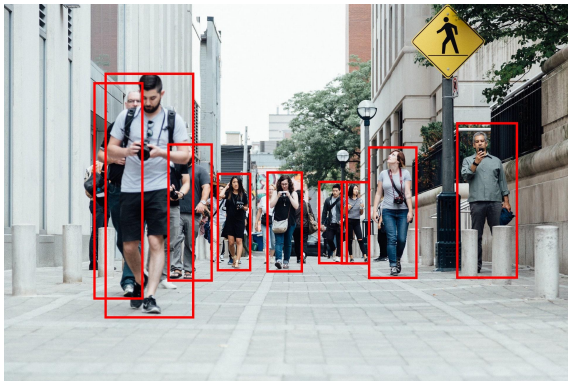
Anchor boxes at each predefined location in each feature map

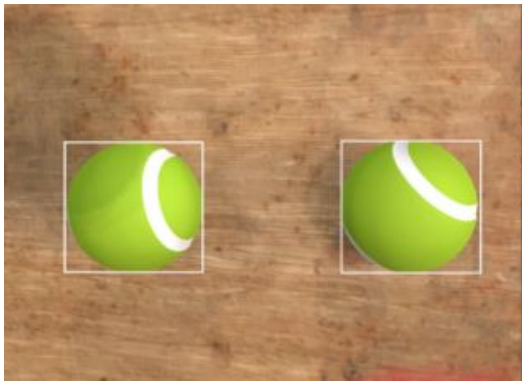


Why Anchor?

- Successful in R-CNNs
- Stabilizes training process by using knowledge of object shapes:
  - Short & wide boxes for vehicles
  - Tall & narrow boxes for people
  - Square-shaped boxes for balls
  - …







Prior Box Selection

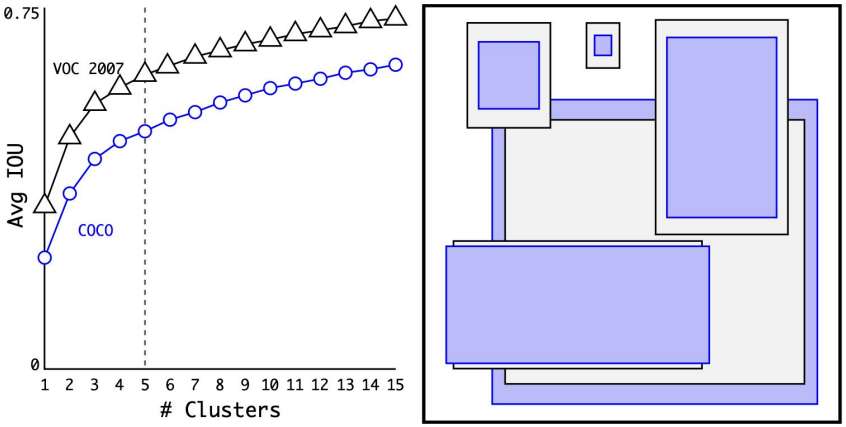


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

- In R-CNN, hand picked priors are used for the anchor boxes.
- Instead, v2 attempts to give the model a better starting point for training by using clustered ground truth boxes as priors.
- They use k-means with k = 5 for a good tradeoff between complexity and recall.
- Distance metric:
  $d(box, centroid) = 1 - IOU(box, centroid)$

Box Predictions

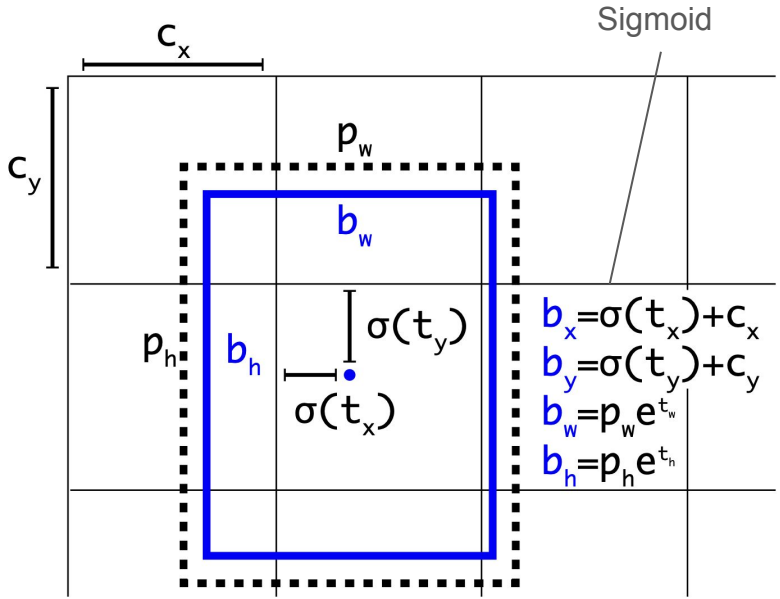
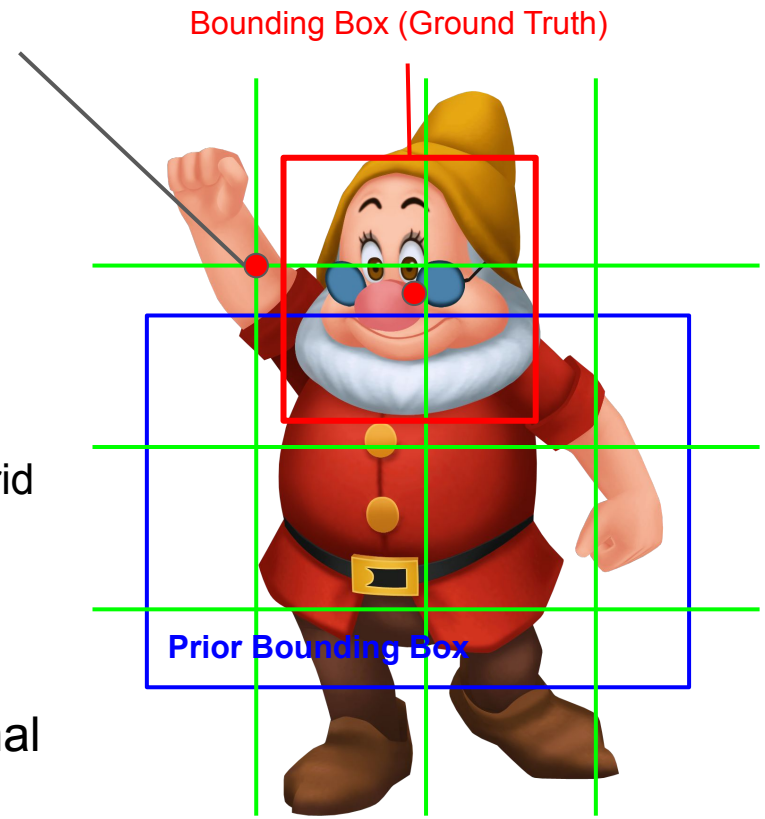


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

- v2 outputs $(t_x, t_y, t_w, t_h, t_o)$
- Instead of directly predicting box coordinates, predicts offsets relative to the current grid cell
- Exponential scaling for width and height
- Object confidence uses sigmoid

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

Final Bounding Boxes



t_x, t_y, t_w, t_h The offsets the model directly predicts

c_x, c_y The coordinates of the upper left corner of the grid

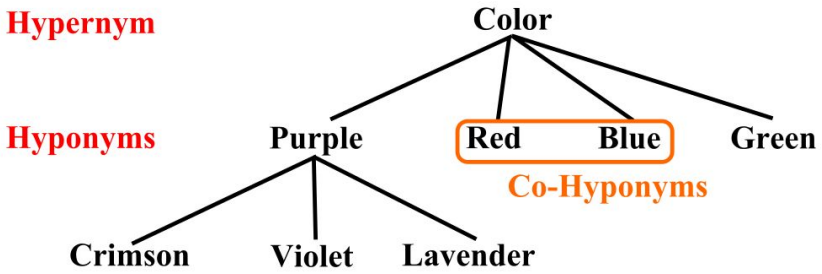
p_w, p_h Width and height of the prior (pre-defined)

- The model predicts offsets to give the final predictions:

$$b_x, b_y, b_w, b_h$$

WordTree

- Transform the flat Image1k dataset into a hierarchical tree structure – “WordTree”
- Use softmax classification on across hyponym classes



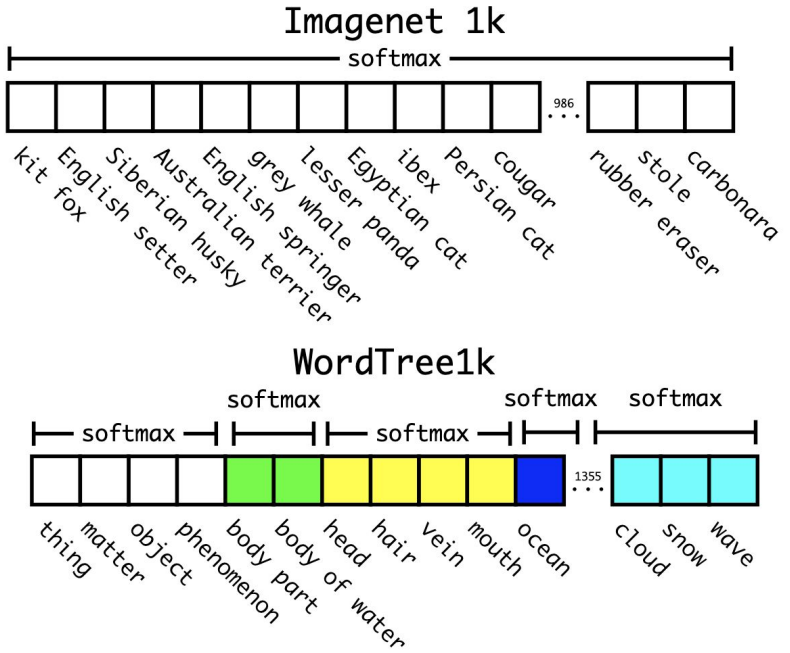
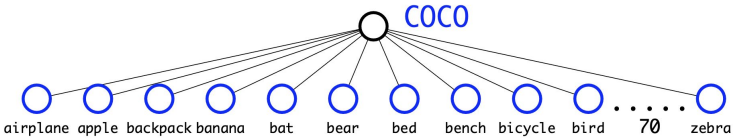
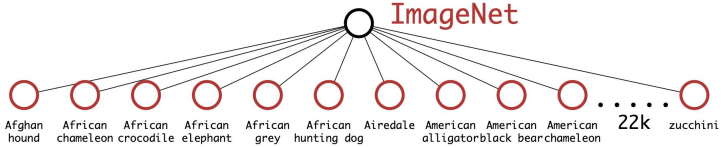


Figure 5: Prediction on ImageNet vs WordTree. Most ImageNet models use one large softmax to predict a probability distribution. Using WordTree we perform multiple softmax operations over co-hyponyms.

Dataset Combination with WordTree



- COCO dataset for object detection
- Only have 80 general classes



- ImageNet dataset for image classification
- Over 22k specific classes
- Run softmax on each level, to get best class for each level, then find best path

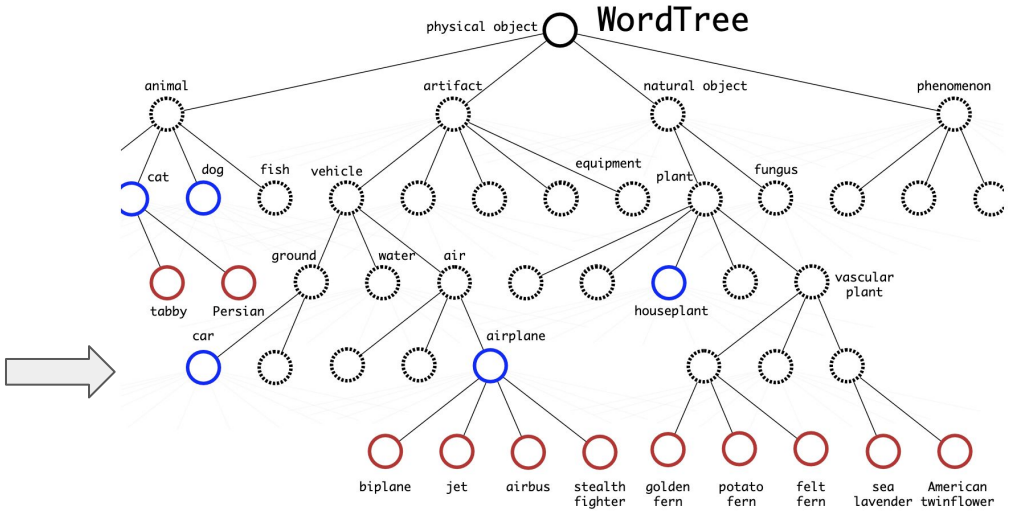


Figure 6: Combining datasets using WordTree hierarchy. Using the WordNet concept graph we build a hierarchical tree of visual concepts. Then we can merge datasets together by mapping the classes in the dataset to synsets in the tree. This is a simplified view of WordTree for illustration purposes.

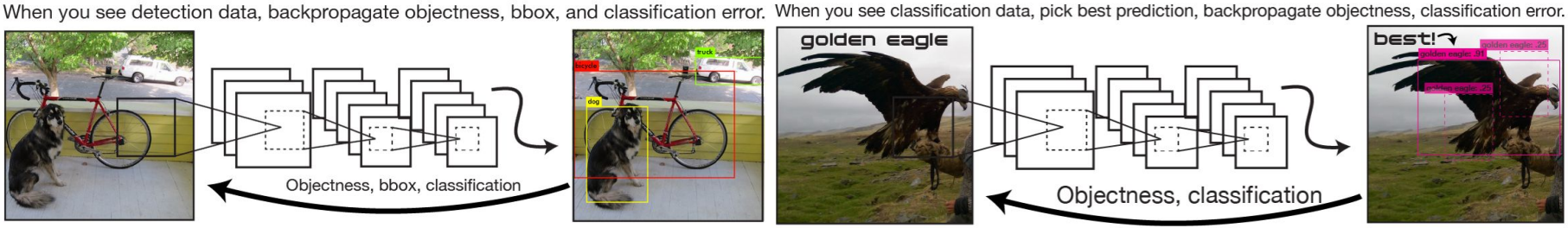
Hierarchical Classification

- The model progressively predicts along the tree
- Every node predict conditional probability
  - Pr(Norfolk terrier|dog), Pr(dog|animal), Pr(mammal|animal), Pr(animal|physical object)
  - Final Pr(Norfolk terrier) = Pr(Norfolk terrier|dog)*Pr(dog|animal)*Pr(mammal|animal)*Pr(animal|physical object)
- The classification task is to find a most probable path, instead of direct softmax on every class.

physical object -> animal -> mammal -> dog

Joint Training

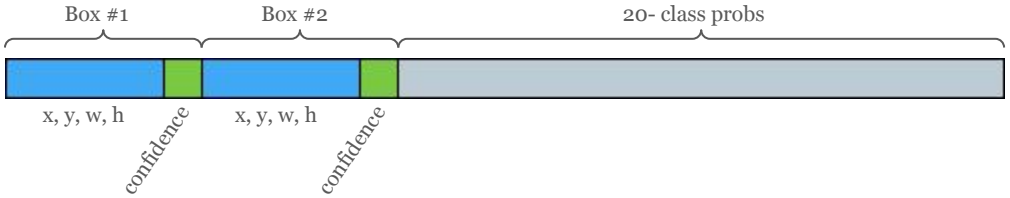
- Not all the images is used for object detection and classification simultaneously.
- During **joint training**, if the image has object detection labels (COCO), the model backpropagates the detection loss. If the input is a classification image (ImageNet), it backpropagates only the classification loss.



Network Overview: Output

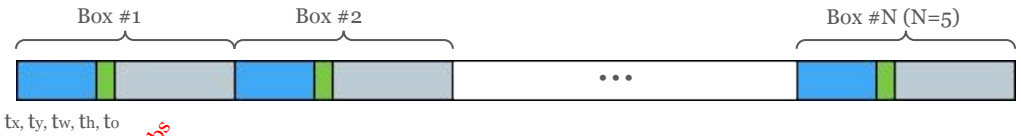
Instance: PASCAL VOC (single grid cell prediction)

YOLOv1



$S^2 \times (B \times 5 + C)$

YOLOv2



$S^2 \times (B \times (5 + C))$

Class probabilities softmaxed over each “synset” (~taxonomy level), giving multiple true classes

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

Network Overview: Backbone

"Standardized" **darknet-19** (inspired by VGG)

Table 6 for classification, modification needed for detection

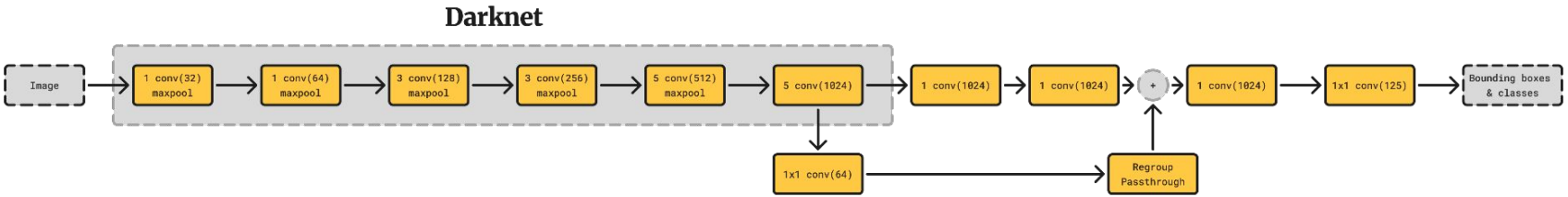
Classification head

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Table 6: Darknet-19.

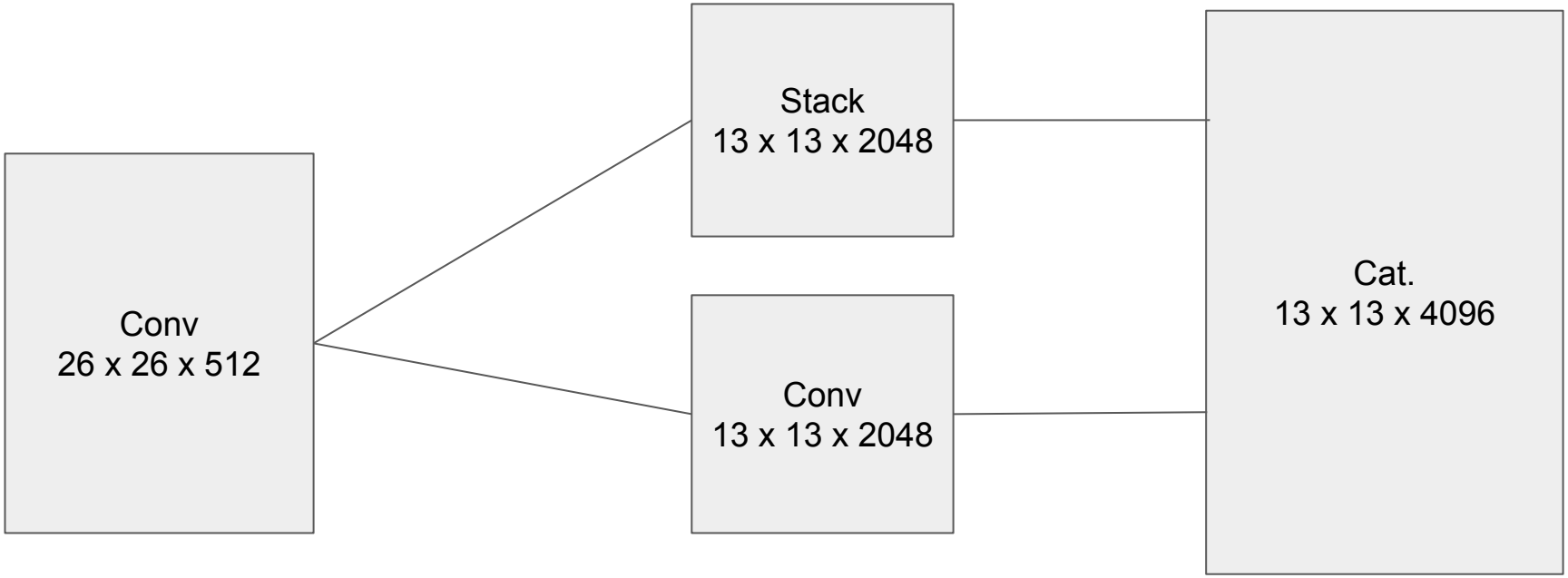
Network Overview: Architecture

Define **N conv(k):**
Cascading convolutions with 3×3, k out channels and 1×1, k/2 out channels, N(odd) conv layers
maxpool:
Max pooling with kernel size 2, stride 2



Fine Grained Features

- YOLO v1 predicted from a 13x13 feature map.
- YOLO v2 includes features from the previous 26x26 layer, stacks and concatenates, to facilitate better prediction of small objects.

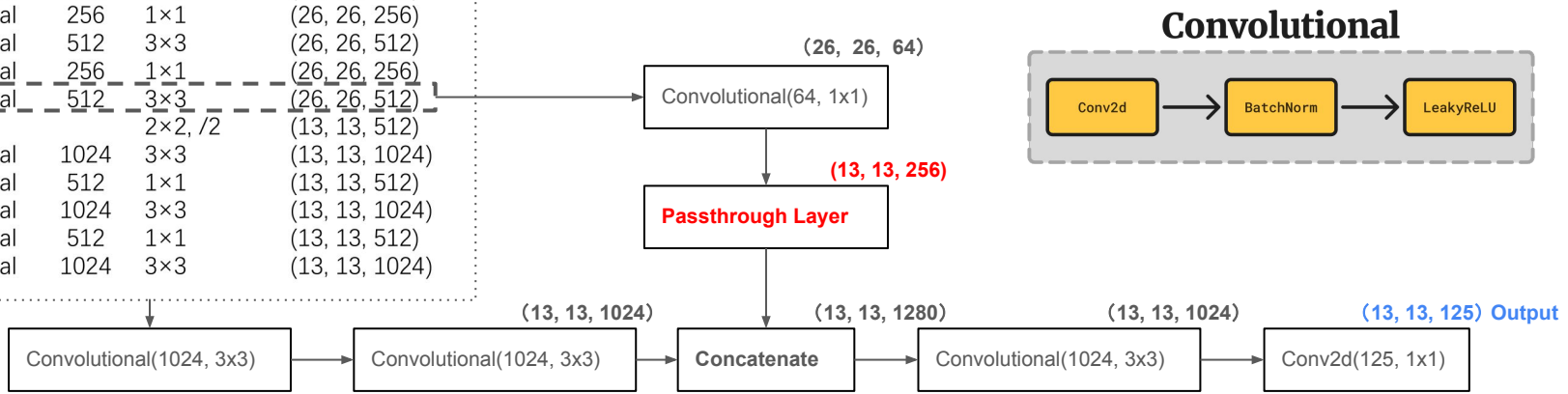


Network Overview: Detection Head

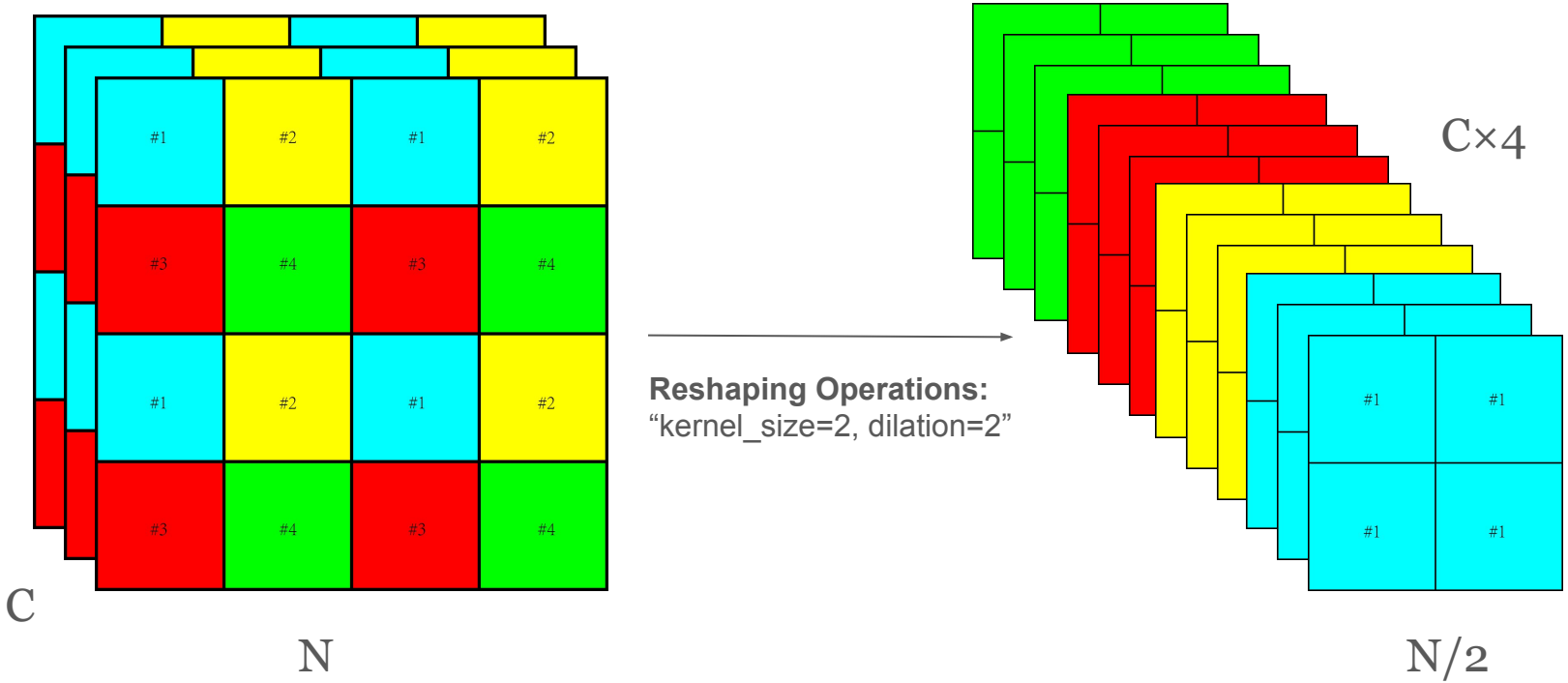
Darknet19

Type	Filters	Kernel size	Output
Convolutional	32	3×3	(416, 416, 32)
Maxpool		2×2, /2	(208, 208, 32)
Convolutional	64	3×3	(208, 208, 64)
Maxpool		2×2, /2	(104, 104, 64)
Convolutional	128	3×3	(104, 104, 128)
Convolutional	64	1×1	(104, 104, 64)
Convolutional	128	3×3	(104, 104, 128)
Maxpool		2×2, /2	(52, 52, 128)
Convolutional	256	3×3	(52, 52, 256)
Convolutional	128	1×1	(52, 52, 128)
Convolutional	256	3×3	(52, 52, 256)
Maxpool		2×2, /2	(26, 26, 256)
Convolutional	512	3×3	(26, 26, 512)
Convolutional	256	1×1	(26, 26, 256)
Convolutional	512	3×3	(26, 26, 512)
Convolutional	256	1×1	(26, 26, 256)
Convolutional	512	3×3	(26, 26, 512)
Maxpool		2×2, /2	(13, 13, 512)
Convolutional	1024	3×3	(13, 13, 1024)
Convolutional	512	1×1	(13, 13, 512)
Convolutional	1024	3×3	(13, 13, 1024)
Convolutional	512	1×1	(13, 13, 512)
Convolutional	1024	3×3	(13, 13, 1024)

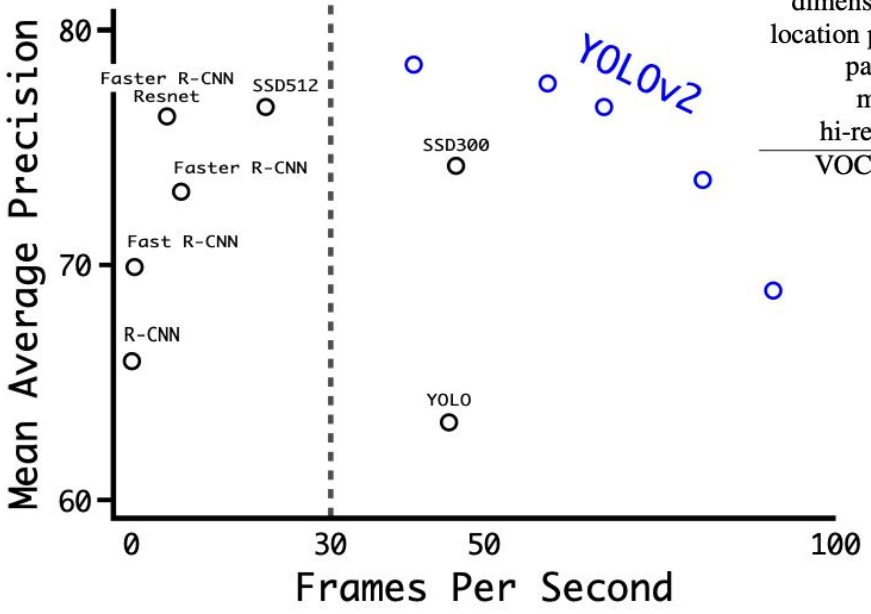
Training for detection. We modify this network for detection by removing the last convolutional layer and instead adding on three 3×3 convolutional layers with 1024 filters each followed by a final 1×1 convolutional layer with the number of outputs we need for detection. For VOC we predict 5 boxes with 5 coordinates each and 20 classes per box so 125 filters. We also add a passthrough layer from the final $3 \times 3 \times 512$ layer to the second to last convolutional layer so that our model can use fine grain features.



Passthrough: Stacking Example



YOLO v2 Advantages



	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

YOLO v2: Limitations

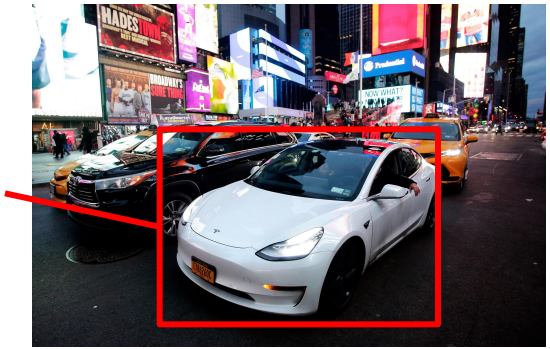
- Poor Small Object Detection
  - Single feature scale (13×13) and pass-through layer is not enough
- Softmax-based Classification
- Shallow CNN Backbone Darknet-19

YOLO v3: Key improvements

- **Pyramid Feature Scales**: Improving small-object detection even further
- **New Backbone Darknet-53**: Better feature extraction capabilities
- **Sigmoid for Classification**: Allowing multi-label classification with multiple true classes

Multi-Label Classification

Vehicle: 0.38
Car:0.31
Train:0.11
…

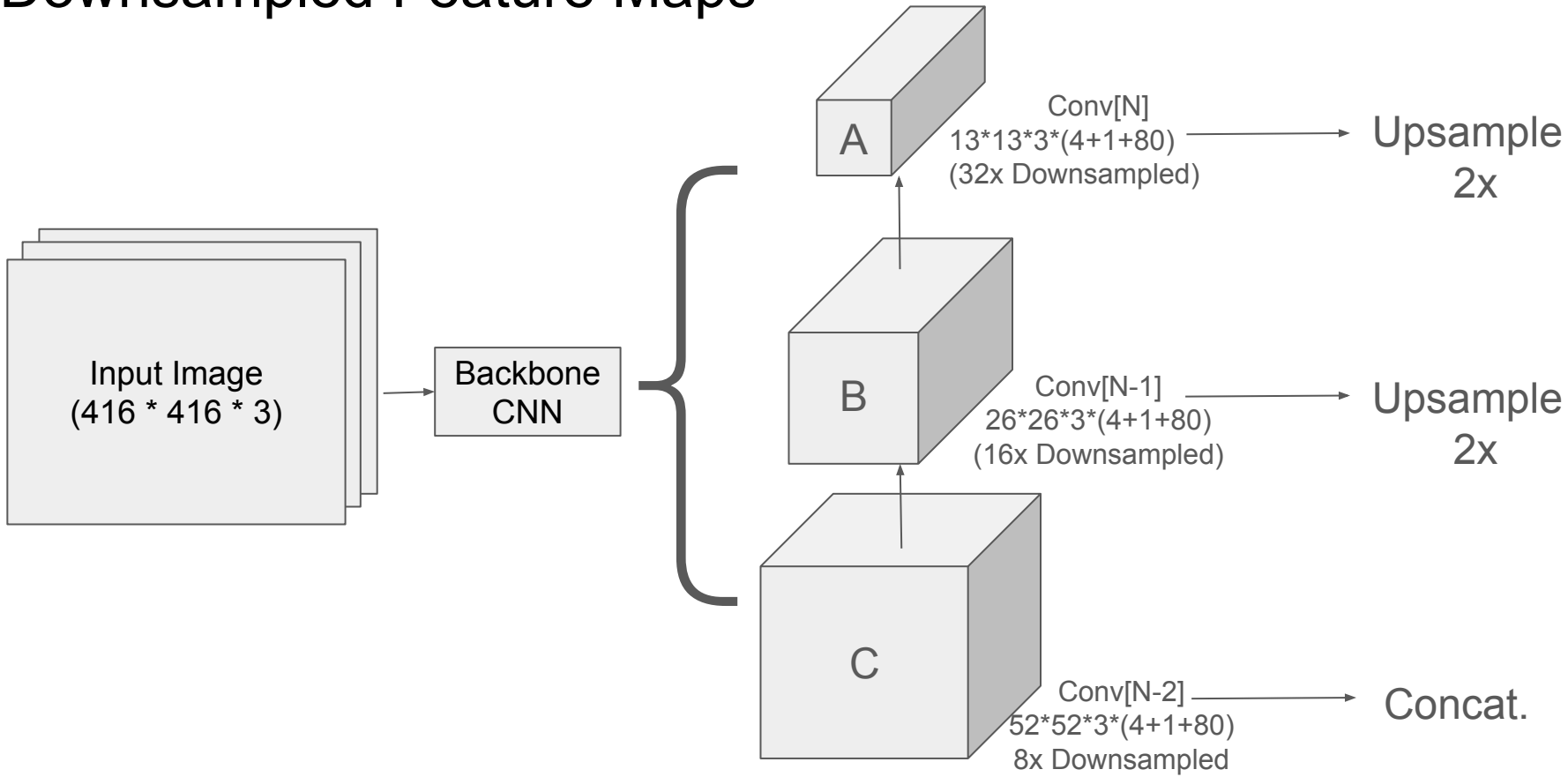


Tesla: 0.99
Car:0.97
Train:0.08
…

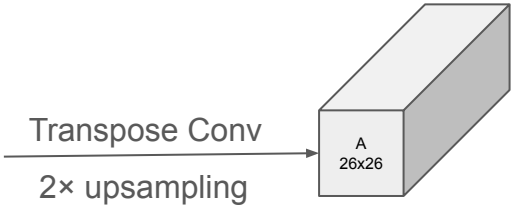
- In YOLO v1, each object only belongs to one class
- In YOLO v2, although hierarchical classification is used, it is still one-label classification (one path)

- With multi-label classification, the classifier predicts probabilities for every class independently
- Sigmoid activation to bound to [0,1]
- BCE Loss within each object class

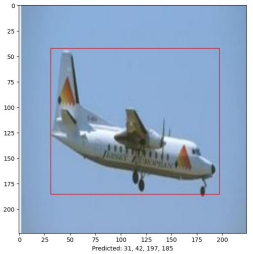
Downsampled Feature Maps

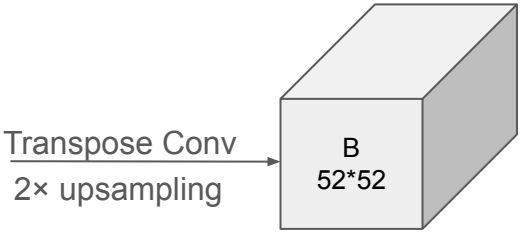


Upsampled Feature Maps

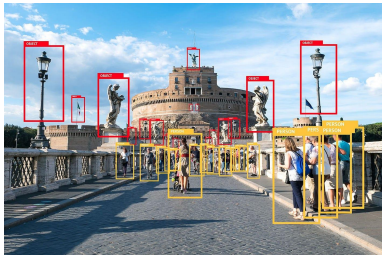


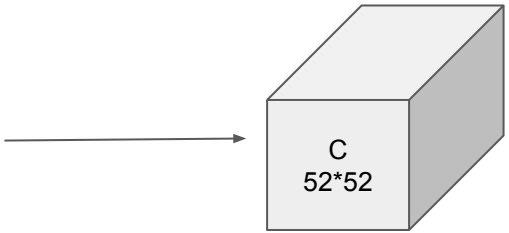
For large target
(Larger receptive field)



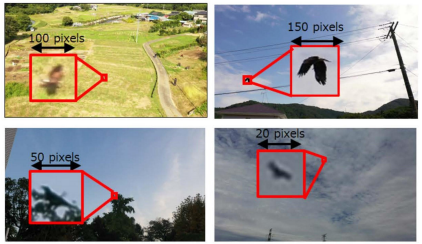


For medium target

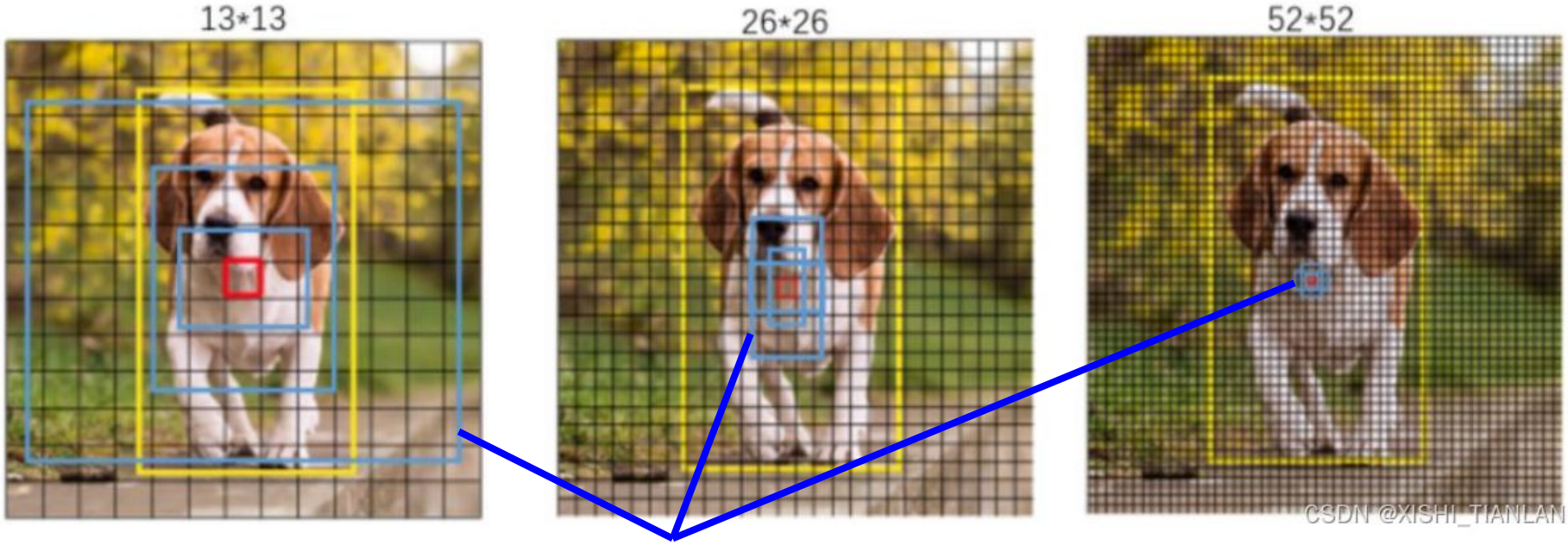




For small target
(Original Scaling)



Anchor Selection



Anchor box: default 9 boxes, 3 for each scale

Choose the box with highest IOU in the grid cell where the object center is located in.

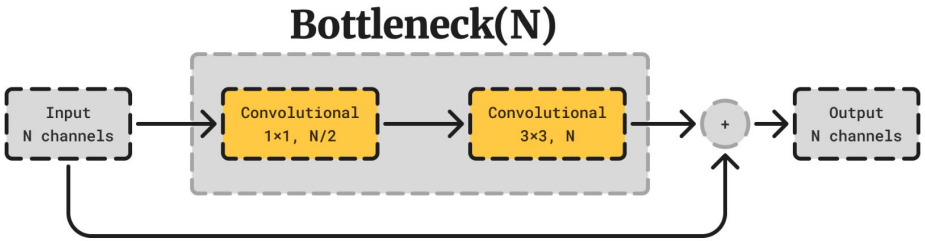
Network Overview: Backbone

"Residual" **darknet-53** (inspired by ResNet)

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
Classification head	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

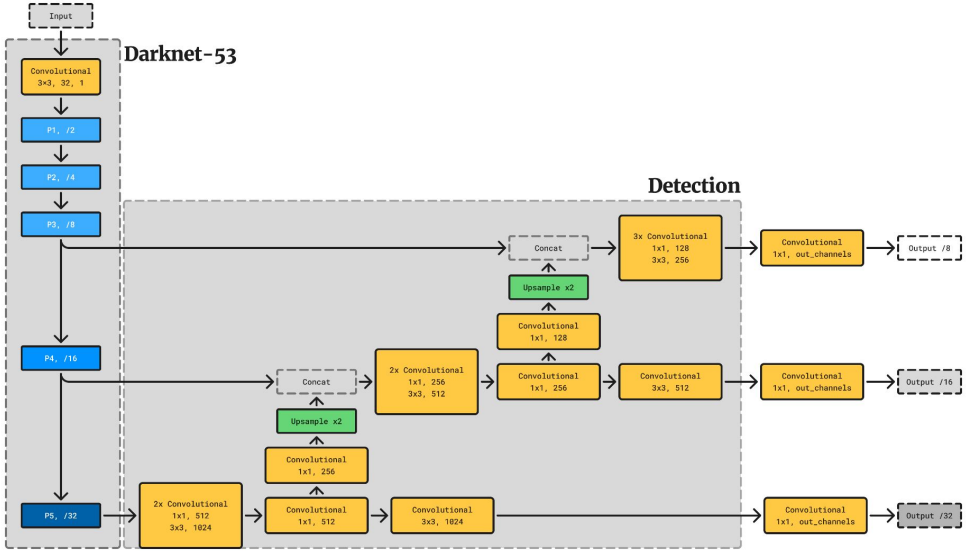
Network Overview: Residual Bottleneck




```
# darknet53 backbone
backbone:
  # [from, number, module, args]
  [
    [-1, 1, Conv, [32, 3, 1]], # 0
    [-1, 1, Conv, [64, 3, 2]], # 1-P1/2
    [-1, 1, Bottleneck, [64]],
    [-1, 1, Conv, [128, 3, 2]], # 3-P2/4
    [-1, 2, Bottleneck, [128]],
    [-1, 1, Conv, [256, 3, 2]], # 5-P3/8
    [-1, 8, Bottleneck, [256]],
    [-1, 1, Conv, [512, 3, 2]], # 7-P4/16
    [-1, 8, Bottleneck, [512]],
    [-1, 1, Conv, [1024, 3, 2]], # 9-P5/32
    [-1, 4, Bottleneck, [1024]], # 10
  ]
```

Code source:
https://github.com/ultralytics/yolov3/blob/master/models/yolov3.yaml

Network Overview: Multiscale Detection Head

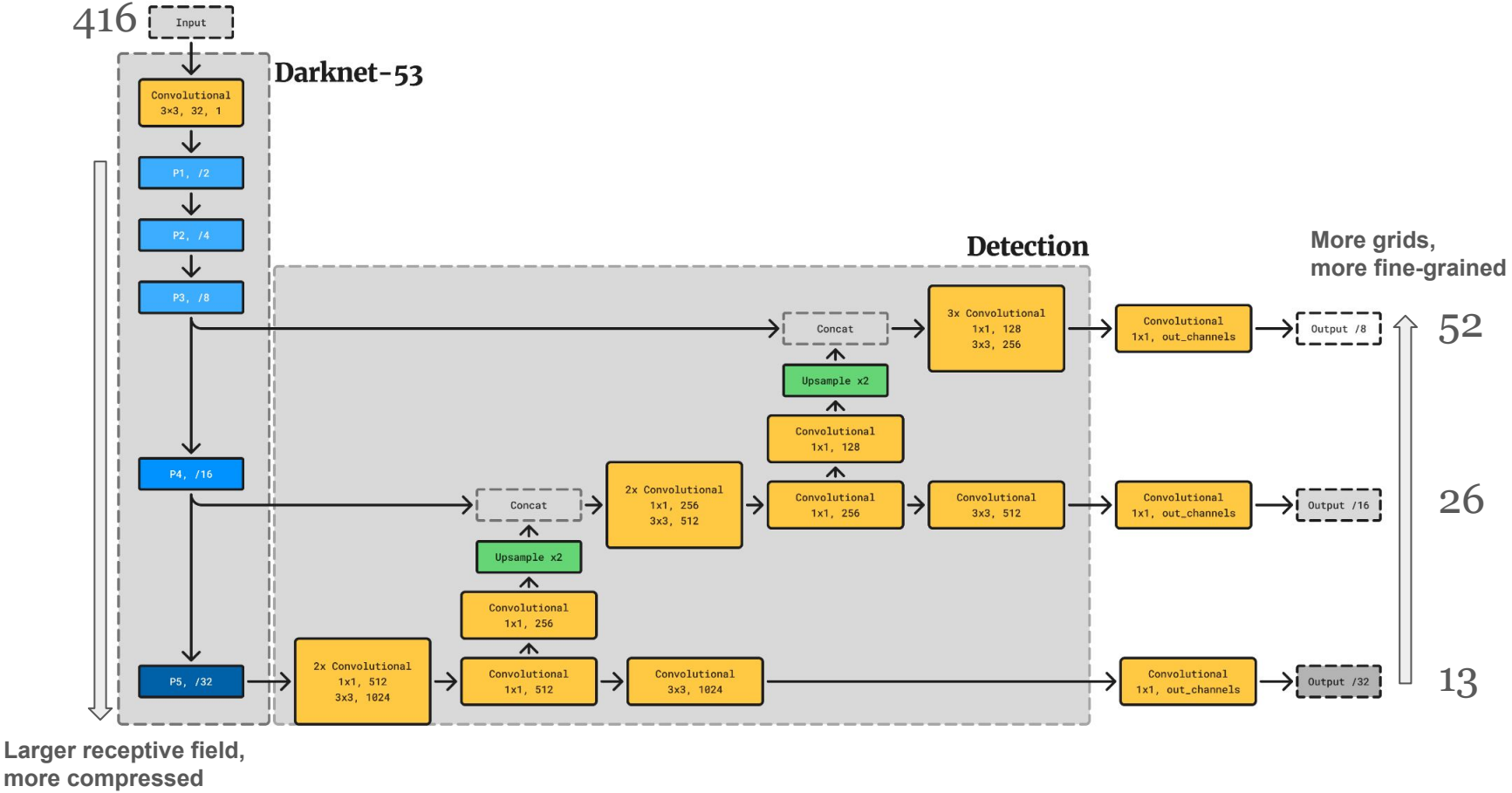



```
# YOLOv3 head
head: [
    [-1, 1, Bottleneck, [1024, False]],
    [-1, 1, Conv, [512, 1, 1]],
    [-1, 1, Conv, [1024, 3, 1]],
    [-1, 1, Conv, [512, 1, 1]],
    [-1, 1, Conv, [1024, 3, 1]], # 15 (P5/32-large)

    [-2, 1, Conv, [256, 1, 1]],
    [-1, 1, nn.Upsample, [None, 2, "nearest"]],
    [[-1, 8], 1, Concat, [1]], # cat backbone P4
    [-1, 1, Bottleneck, [512, False]],
    [-1, 1, Bottleneck, [512, False]],
    [-1, 1, Conv, [256, 1, 1]],
    [-1, 1, Conv, [512, 3, 1]], # 22 (P4/16-medium)

    [-2, 1, Conv, [128, 1, 1]],
    [-1, 1, nn.Upsample, [None, 2, "nearest"]],
    [[-1, 6], 1, Concat, [1]], # cat backbone P3
    [-1, 1, Bottleneck, [256, False]],
    [-1, 2, Bottleneck, [256, False]], # 27 (P3/8-small)

    [[27, 22, 15], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
  ]
```


416
Input
Darknet-53
Convolutional
3×3, 32, 1
P1, /2
P2, /4
P3, /8
P4, /16
P5, /32
2x Convolutional
1x1, 512
3x3, 1024
Convolutional
1x1, 512
Convolutional
3x3, 1024
Detection
Upsample x2
Convolutional
1x1, 256
Concat
2x Convolutional
1x1, 256
3x3, 512
Convolutional
1x1, 256
Convolutional
3x3, 512
Upsample x2
Convolutional
1x1, 128
Concat
3x Convolutional
1x1, 128
3x3, 256
Convolutional
1x1, out_channels
Output /32
Convolutional
1x1, out_channels
Output /16
Convolutional
1x1, out_channels
Output /8
More grids,
more fine-grained
52
26
13
Larger receptive field,
more compressed

Results

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Table 2. **Comparison of backbones.** Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

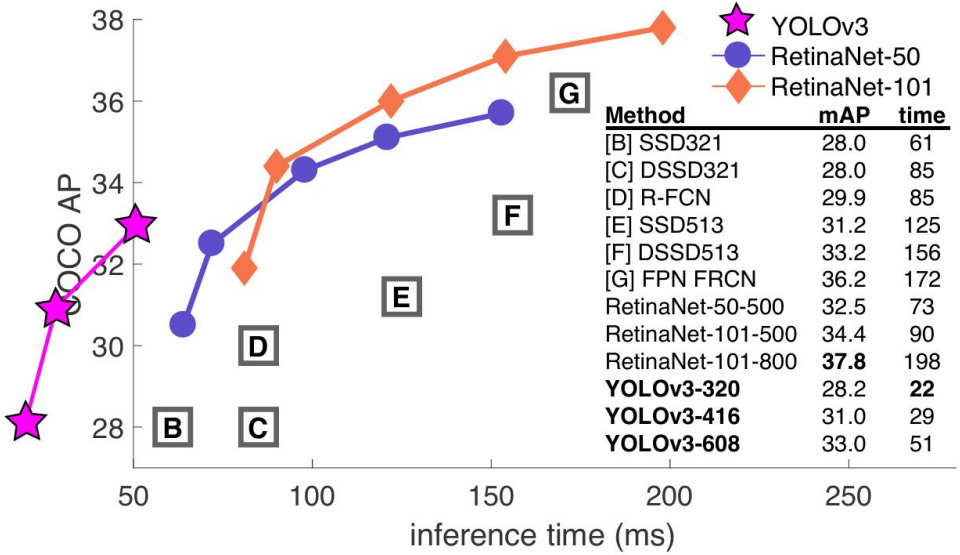


Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

YOLO v4, v5, v6, v7……

- v4: Ablated various methods from the literature: better data augmentation (mosaic, CutMix), better hyperparameter search, spatial attention, experimenting with losses (e.g IoU vs GIoU) and much more.

- **Key improvement**: Cross-stage Partial Connections (CSP) – mixing of scales across feature map layers.

- Best model achieves SOTA on real-time detection.

YOLO v4, v5, v6, v7……

- v6: More ablations implementing the latest research

- **Key results from best model:**

  - Single head with output p - combining classification score, objectness and IOU - for every bounding box (Task alignment training - TAL).

  - Better weighting of negatives using VariFocal loss:

$$VFL(p,q) = -q(q\log(p) + (1-q)\log(1-p)) \text{ if } q > 0$$

$$VFL(p,q) = -\alpha p^{\gamma}\log(1-p)$$

YOLO v4, v5, v6, v7……

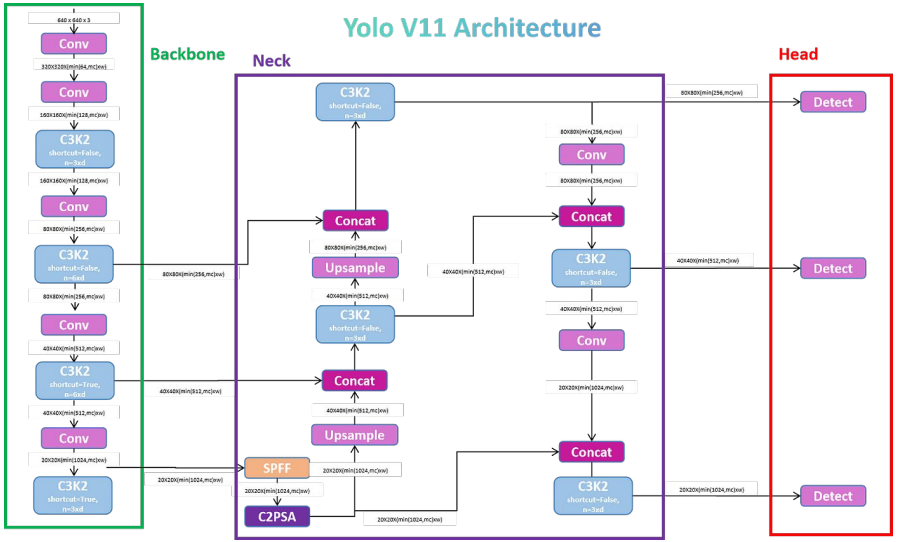
- v7: Big step up:

Table 1: Comparison of baseline object detectors.

Model	#Param.	FLOPs	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
YOLOv4 [3]	64.4M	142.8G	640	49.7%	68.2%	54.3%	32.9%	54.8%	63.7%
YOLOR-u5 (r6.1) [81]	46.5M	109.1G	640	50.2%	68.7%	54.6%	33.2%	55.5%	63.7%
YOLOv4-CSP [79]	52.9M	120.4G	640	50.3%	68.6%	54.9%	34.2%	55.6%	65.1%
YOLOR-CSP [81]	52.9M	120.4G	640	50.8%	69.5%	55.3%	33.7%	56.0%	65.4%
YOLOv7	36.9M	104.7G	640	**51.2%**	**69.7%**	**55.5%**	**35.2%**	**56.0%**	**66.7%**
improvement	-43%	-15%	-	+0.4	+0.2	+0.2	+1.5	=	+1.3
YOLOR-CSP-X [81]	96.9M	226.8G	640	52.7%	**71.3%**	57.4%	36.3%	57.5%	68.3%
YOLOv7-X	71.3M	189.9G	640	**52.9%**	71.1%	**57.5%**	**36.9%**	**57.7%**	**68.6%**
improvement	-36%	-19%	-	+0.2	-0.2	+0.1	+0.6	+0.2	+0.3
YOLOv4-tiny [79]	6.1	6.9	416	24.9%	42.1%	25.7%	8.7%	28.4%	39.2%
YOLOv7-tiny	6.2	5.8	416	**35.2%**	**52.8%**	**37.3%**	**15.7%**	**38.0%**	**53.4%**
improvement	+2%	-19%	-	+10.3	+10.7	+11.6	+7.0	+9.6	+14.2
YOLOv4-tiny-3l [79]	8.7	5.2	320	30.8%	47.3%	32.2%	**10.9%**	31.9%	51.5%
YOLOv7-tiny	6.2	3.5	320	**30.8%**	**47.3%**	**32.2%**	10.0%	**31.9%**	**52.2%**
improvement	-39%	-49%	-	=	=	=	-0.9	=	+0.7
YOLOR-E6 [81]	115.8M	683.2G	1280	55.7%	73.2%	60.7%	40.1%	**60.4%**	69.2%
YOLOv7-E6	97.2M	515.2G	1280	**55.9%**	**73.5%**	**61.1%**	**40.6%**	60.3%	**70.0%**
improvement	-19%	-33%	-	+0.2	+0.3	+0.4	+0.5	-0.1	+0.8
YOLOR-D6 [81]	151.7M	935.6G	1280	56.1%	73.9%	61.2%	**42.4%**	60.5%	69.9%
YOLOv7-D6	154.7M	806.8G	1280	56.3%	73.8%	61.4%	41.3%	60.6%	70.1%
YOLOv7-E6E	151.7M	843.2G	1280	**56.8%**	**74.4%**	**62.1%**	40.8%	**62.1%**	**70.6%**
improvement	=	-11%	-	+0.7	+0.5	+0.9	-1.6	+1.6	+0.7

YOLO v11

- v11: **2025 SOTA** model by Ultralytics: C3k2 (Cross Stage Partial with kernel size 2) block, SPPF (Spatial Pyramid Pooling - Fast), and C2PSA (Convolutional block with Parallel Spatial Attention)
- We went from **simple** to **complex** from v1 to v11, but key ideas remain:



- **Fully convolutional**
- **Mixing of feature scales (CSP)**
- **Unified loss**
- **Multi-scale predictions**

Thank you!