EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Image Analysis II

Object Detection – RCNN/YOLO/SSD

Radovan Fusek

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Classic Sliding Window

Robust Real-Time Face Detection

PAUL VIOLA
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
viola@microsoft.com

MICHAEL J. JONES
Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA 02139, USA
mjones@merl.com

Received September 10, 2001; Revised July 10, 2003; Accepted July 11, 2003

Histograms of Oriented Gradients for Human Detection

Navneet Dalal and Bill Triggs
INRIA Rhône-Alps, 655 avenue de l'Europe, Montbonnot 38334, France
{Navneet.Dalal,Bill.Triggs}@inrialpes.fr, http://lear.inrialpes.fr

Cascade Object Detection with Deformable Part Models*

Pedro F. Felzenszwalb
University of Chicago
pff@cs.uchicago.edu

Ross B. Girshick
University of Chicago
rbg@cs.uchicago.edu

David McAllester
TTI at Chicago
mcallester@ttic.edu

Two Stage

Fast R-CNN

Ross Girshick
Microsoft Research
rbg@microsoft.com

Rich feature hierarchies for accurate object detection and semantic segmentation
Tech report (v5)

Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik
UC Berkeley
{rbg,jdonahue,trevor,malik}@eecs.berkeley.edu

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

One Stage

You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

SSD: Single Shot MultiBox Detector

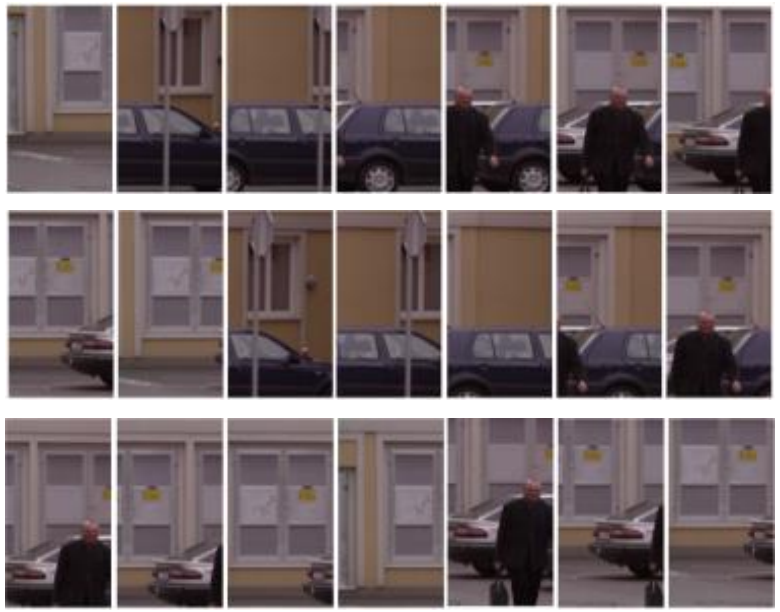
Wei Liu[1], Dragomir Anguelov[2], Dumitru Erhan[3], Christian Szegedy[3],
Scott Reed[4], Cheng-Yang Fu[1], Alexander C. Berg[1]

[1]UNC Chapel Hill [2]Zoox Inc. [3]Google Inc. [4]University of Michigan, Ann-Arbor
[1]wliu@cs.unc.edu, [2]drago@zoox.com, [3]{dumitru,szegedy}@google.com,
[4]reedscot@umich.edu, [1]{cyfu,aberg}@cs.unc.edu

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Region-Based CNNs (R-CNNs)

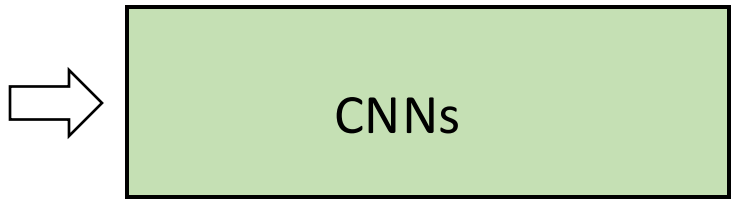
- Classical way (how to localize/detect object) is based on sliding window technique



EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Region-Based CNNs (R-CNNs)

- Disadvantages of sliding window with the use off very deep CNNs for object detection
  - many different image regions
  - each region is used as an input for CNNs
    - computational cost – overlapping regions (stride parameters)
    - duplicated operations



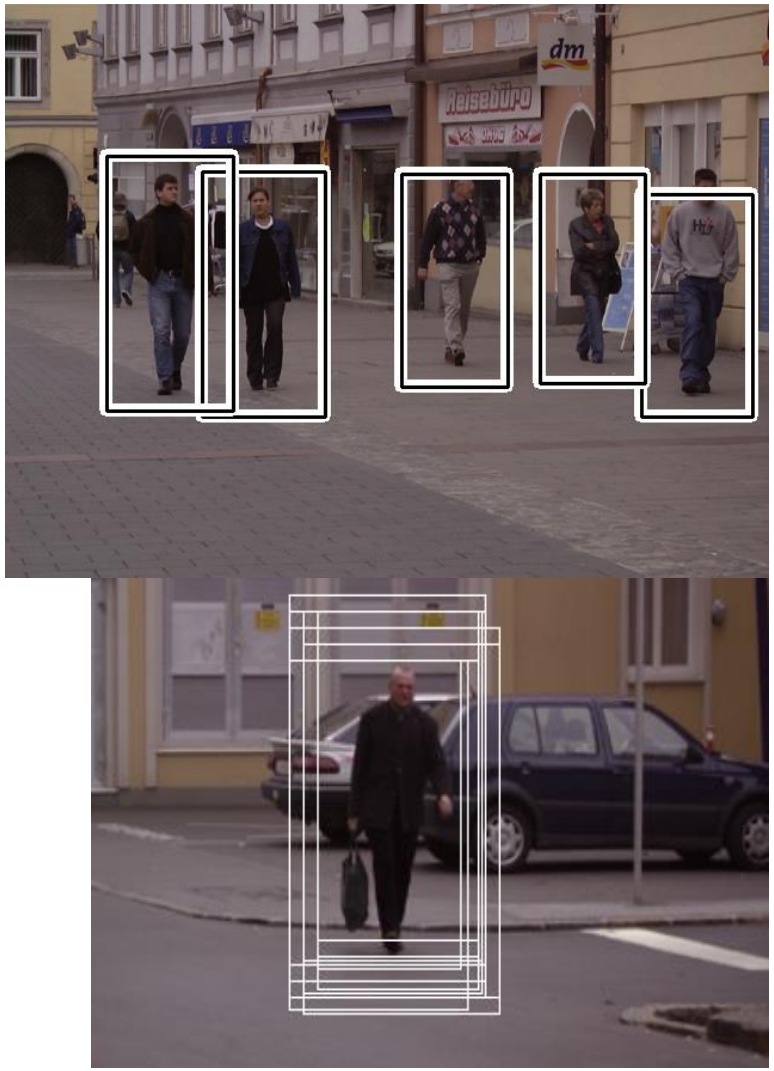
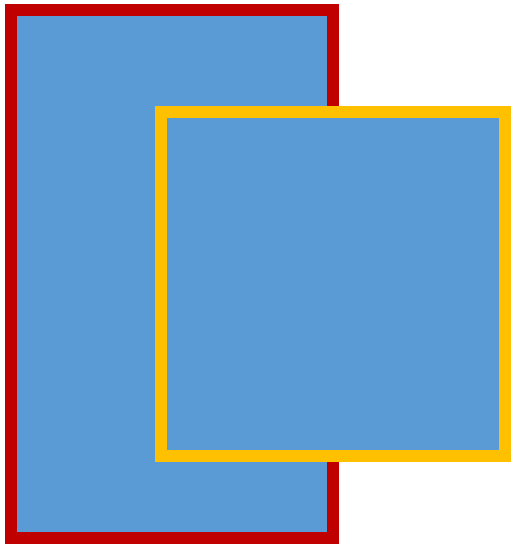


Figure 3: Non-Maximum Suppression (NMS). a) Shows the typical output of an object detection model containing multiple overlapping boxes. b) Shows the output after NMS.

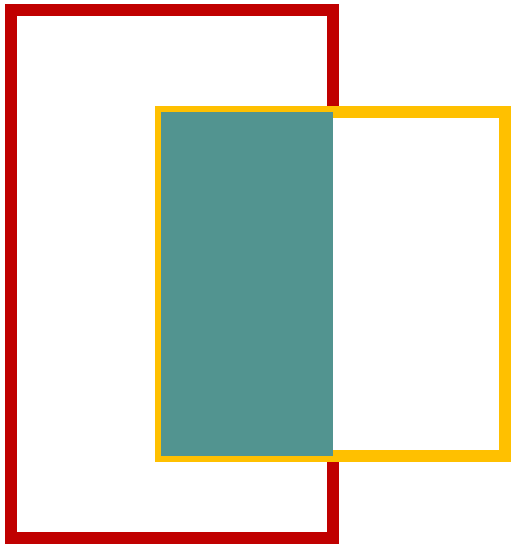
https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

Intersection, Union

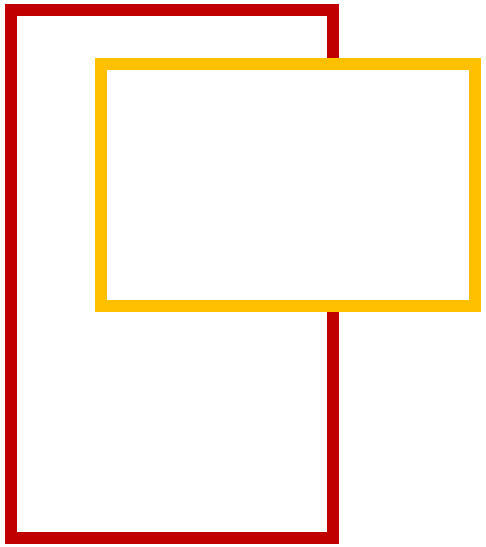
Union



Intersection



IoU = ?

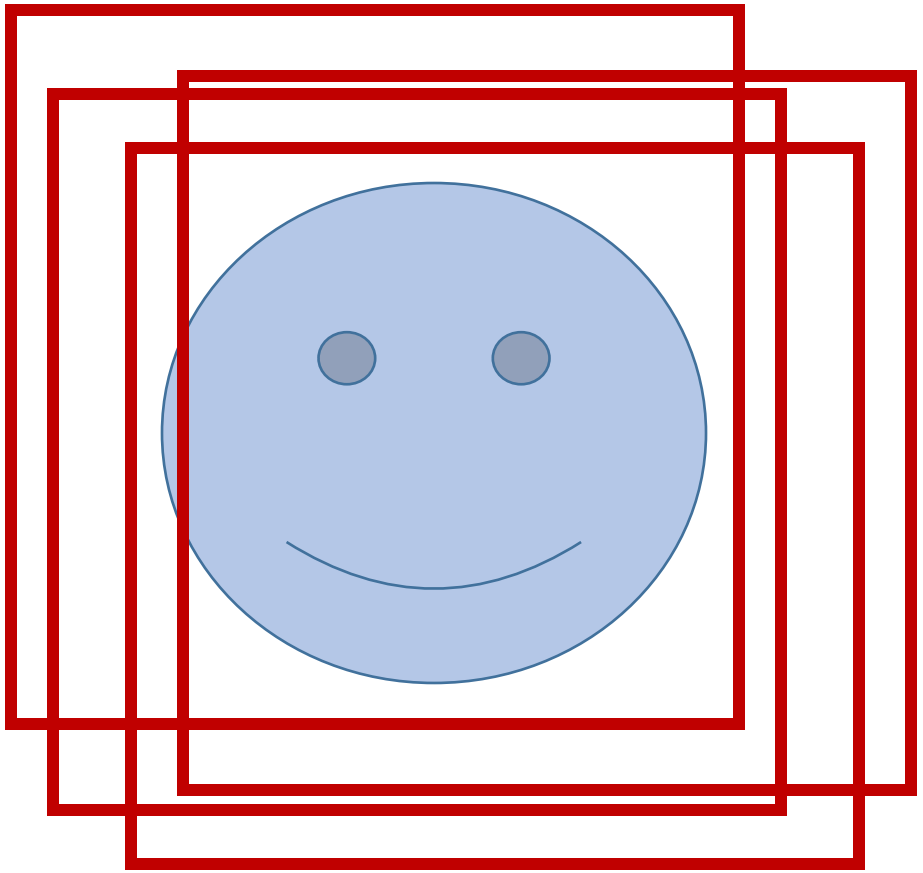


IoU = ?

IoU = Area of Overlap / Area of Union

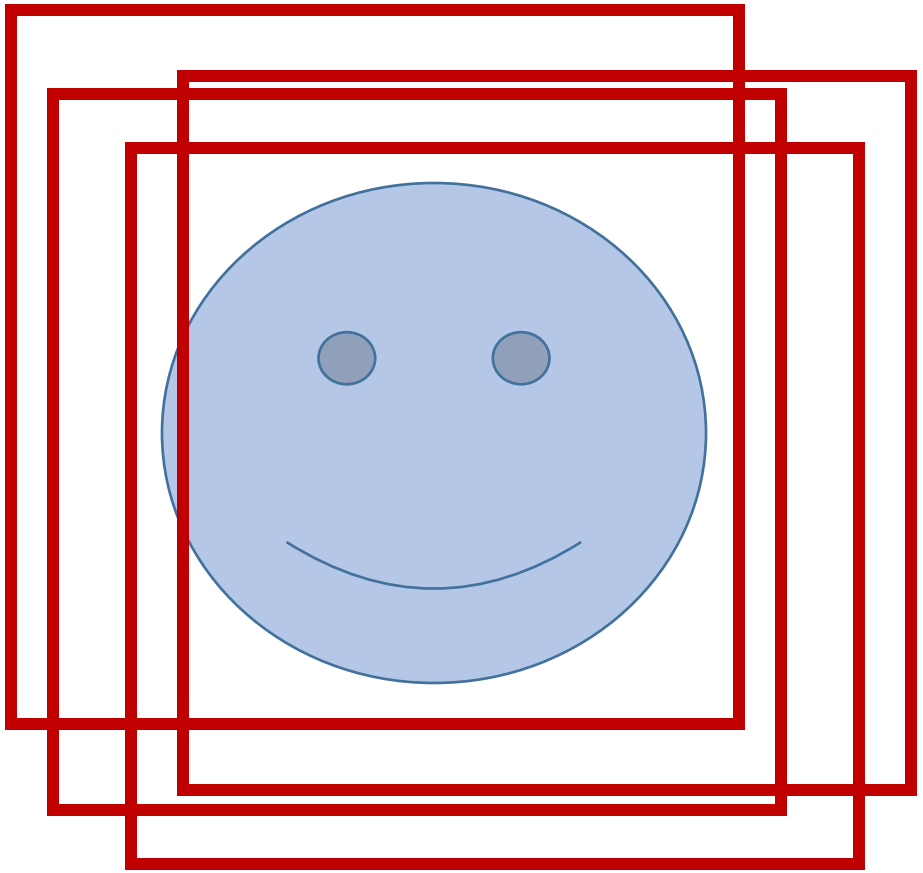
EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Non-max Suppresion



How select only one box?

Non-max Suppresion



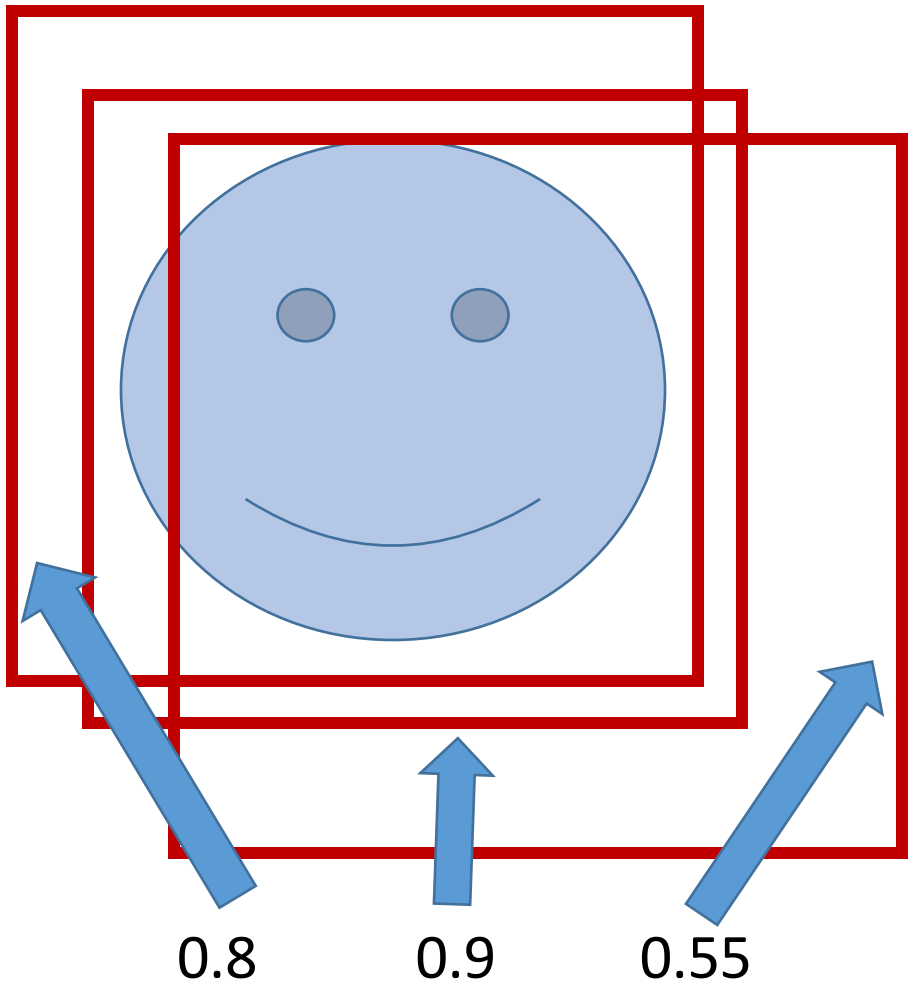
How select only one box?

1. Discard all boxes with confidence smaller or equal to 0.6

2. Select the box with largest confidence

3. Discard all remaining box with IoU greater or equal to 0.5

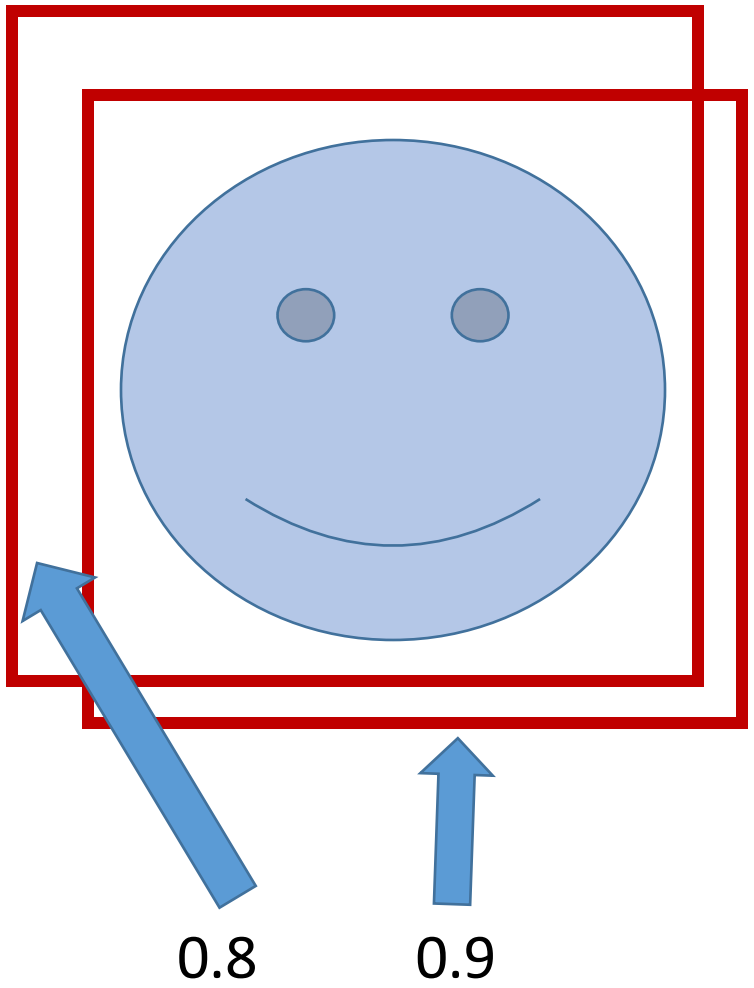
Non-max Suppresion



How select only one box?

1. Discard all boxes with confidence smaller or equal to 0.6

Non-max Suppresion

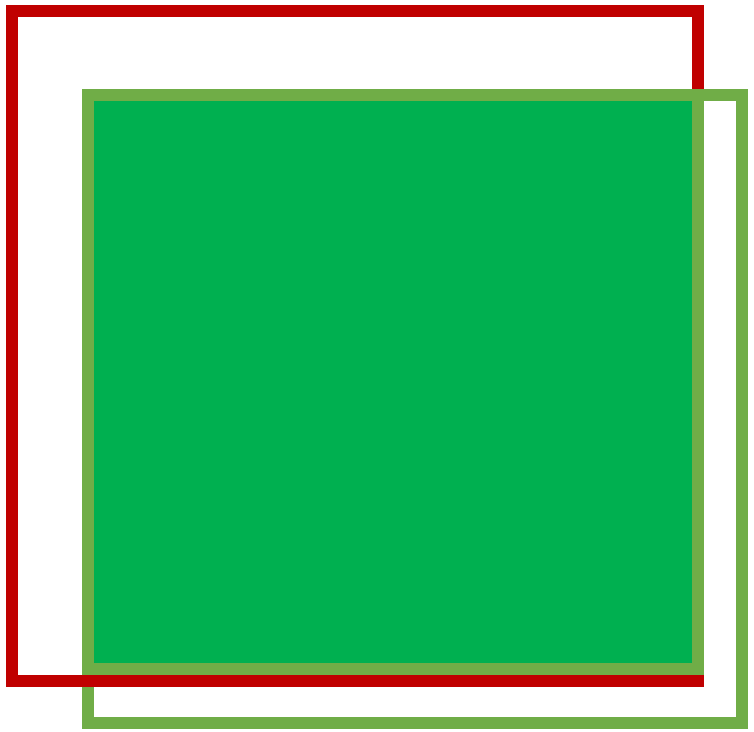


How select only one box?

1. Discard all boxes with confidence smaller or equal to 0.6

2. Select the box with largest confidence

Non-max Suppresion



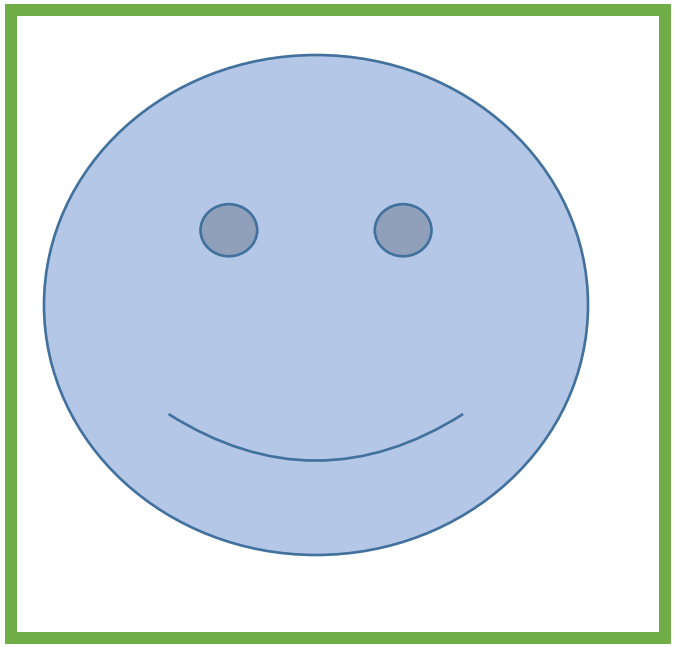
How select only one box?

1. Discard all boxes with confidence smaller or equal to 0.6

2. Select the box with largest confidence

3. Discard all remaining box with IoU greater or equal to 0.5

Non-max Suppresion



How select only one box?

1. Discard all boxes with confidence smaller or equal to 0.6

2. Select the box with largest confidence

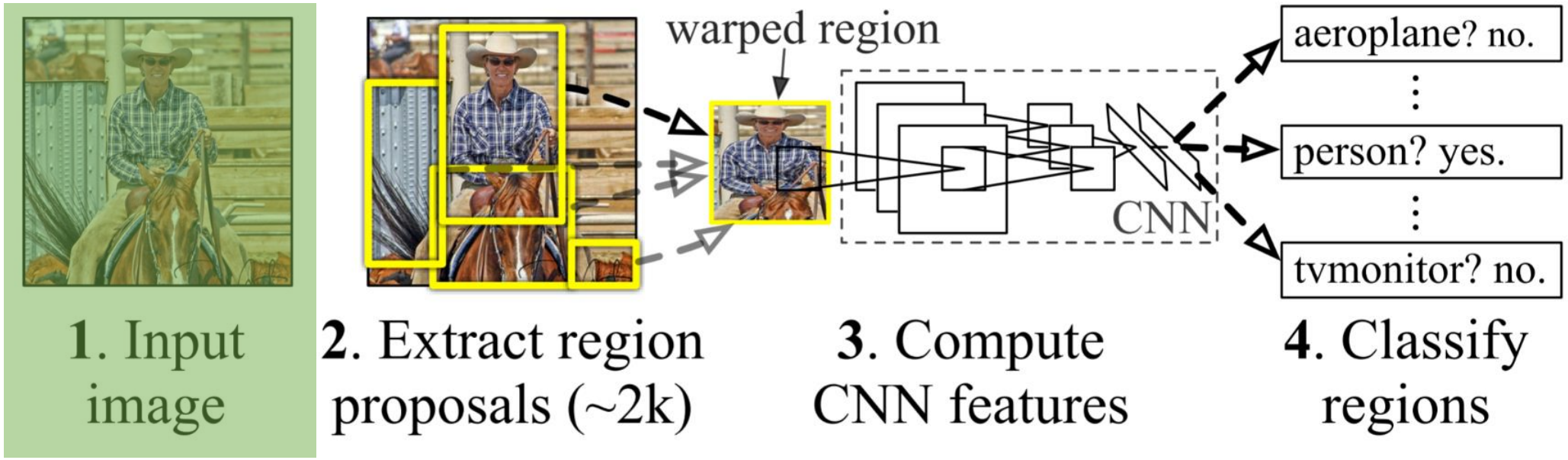
3. Discard all remaining box with IoU greater or equal to 0.5

Region-Based CNNs (R-CNNs)

- **R-CNN**
  - Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580–587).
- **Fast R-CNN**
  - Girshick, R. (2015). Fast r-cnn. *Proceedings of the IEEE international conference on computer vision* (pp. 1440–1448).
- **Faster R-CNN**
  - Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: towards real-time object detection with region proposal networks. *Advances in neural information processing systems* (pp. 91–99).
- **Mask R-CNN**
  - He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. *Proceedings of the IEEE international conference on computer vision* (pp. 2961–2969).

Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**
- **(1) takes an input image**

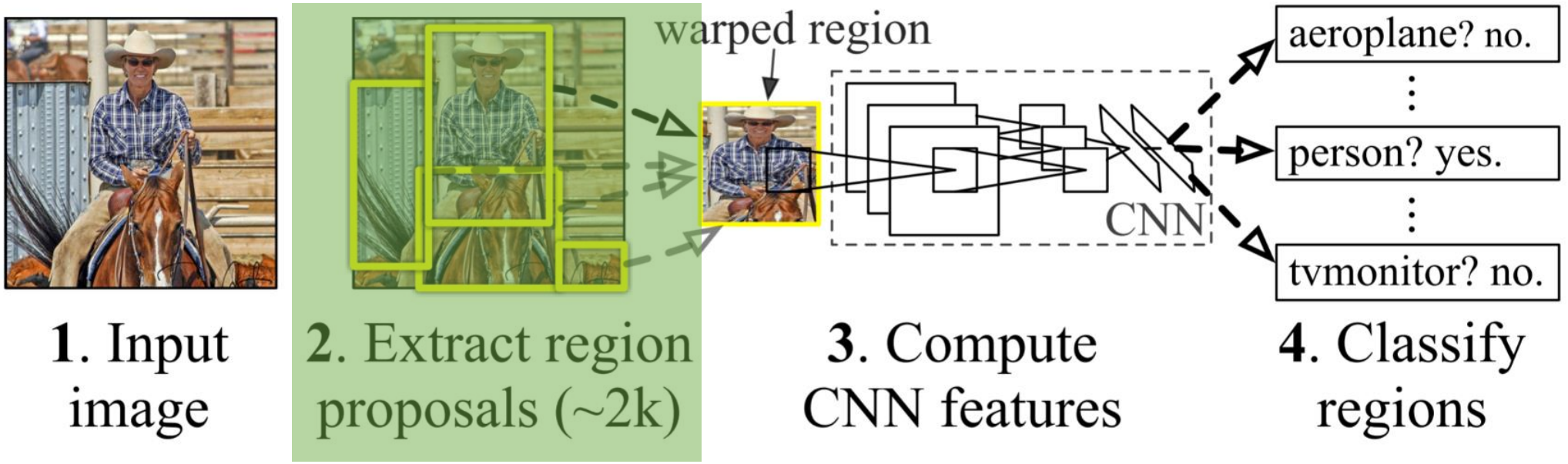


EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**

- (1) takes an input image
- **(2) extracts around 2000 bottom-up regions using selective search**
  - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013

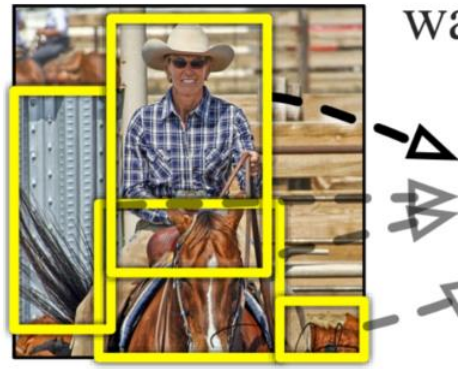


EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

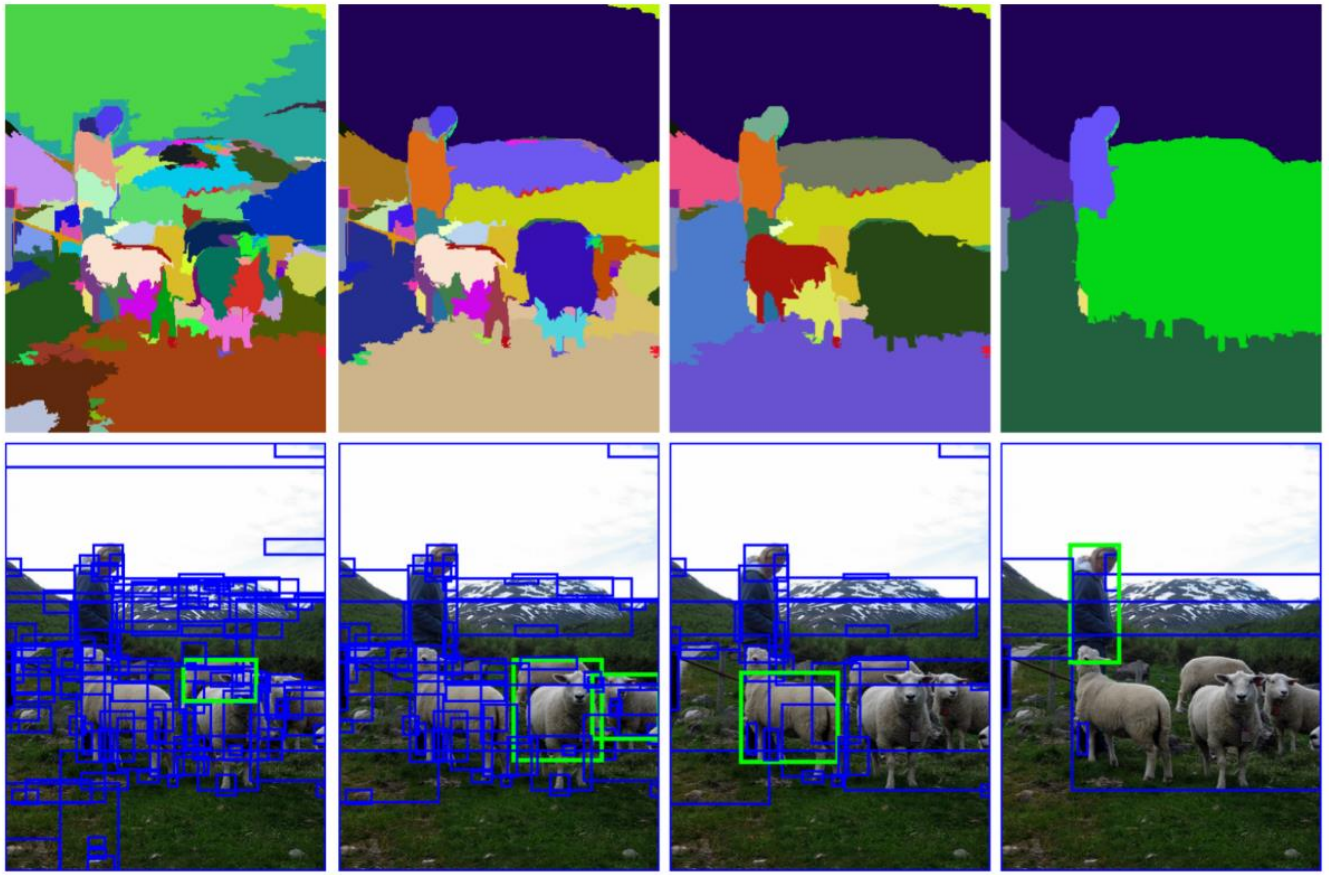
Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**

- (1) takes an input image
- **(2) extracts around 2000 bottom-up regions using selective search**
  - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013



2. Extract region proposals (~2k)

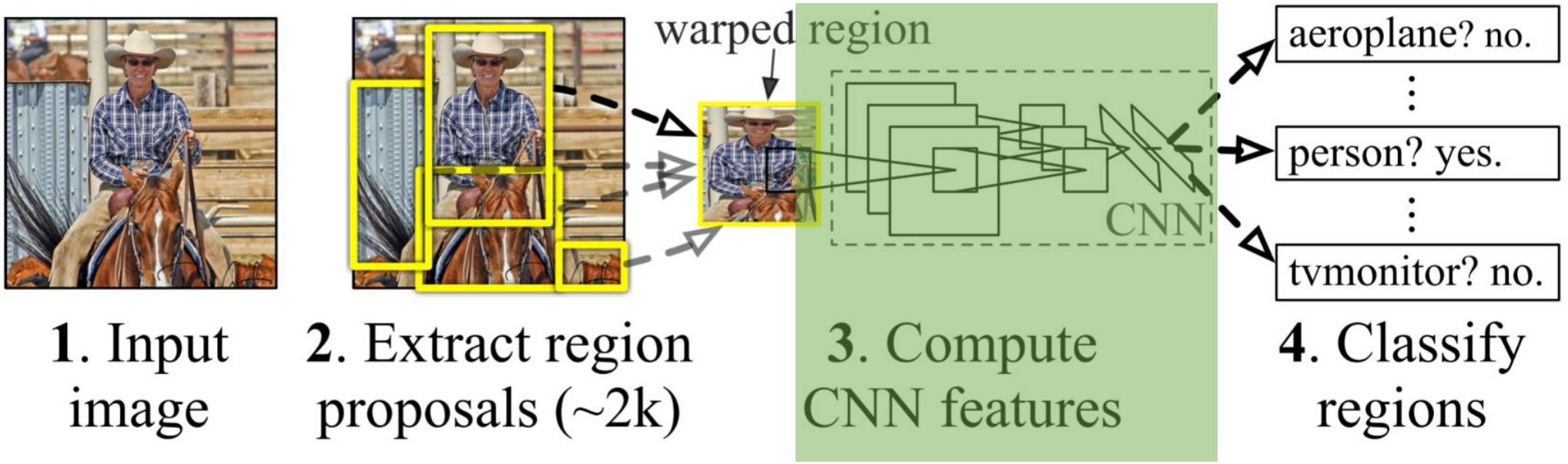


EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**

- (1) takes an input image
- (2) extracts around 2000 bottom-up regions using selective search
  - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013
- **(3) computes features for each region using a large convolutional neural network (CNN)**
  - AlexNet is used to compute the features

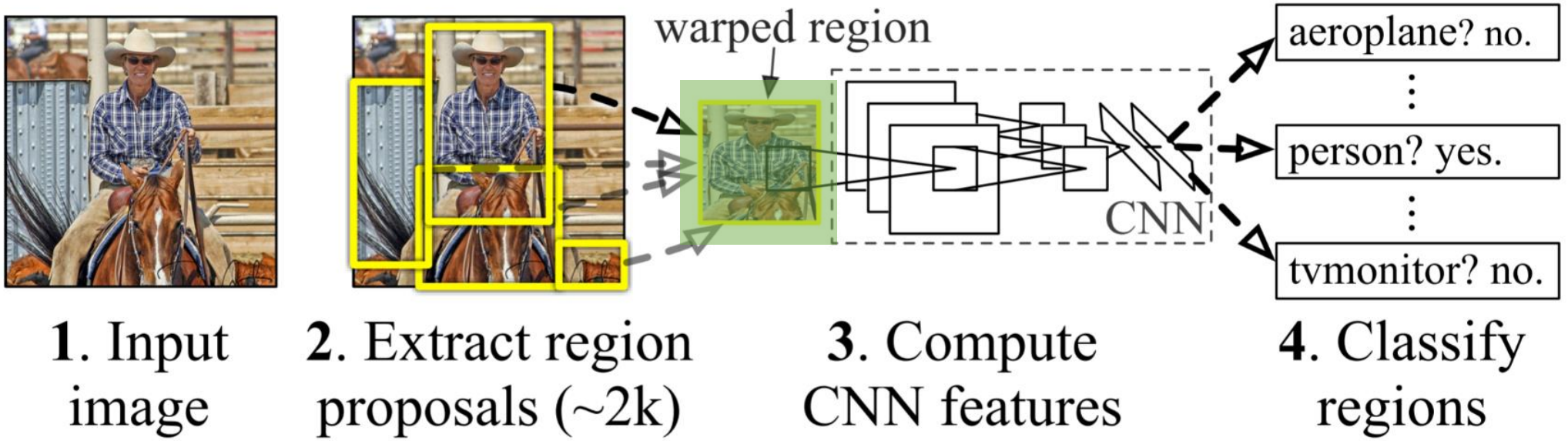


EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**

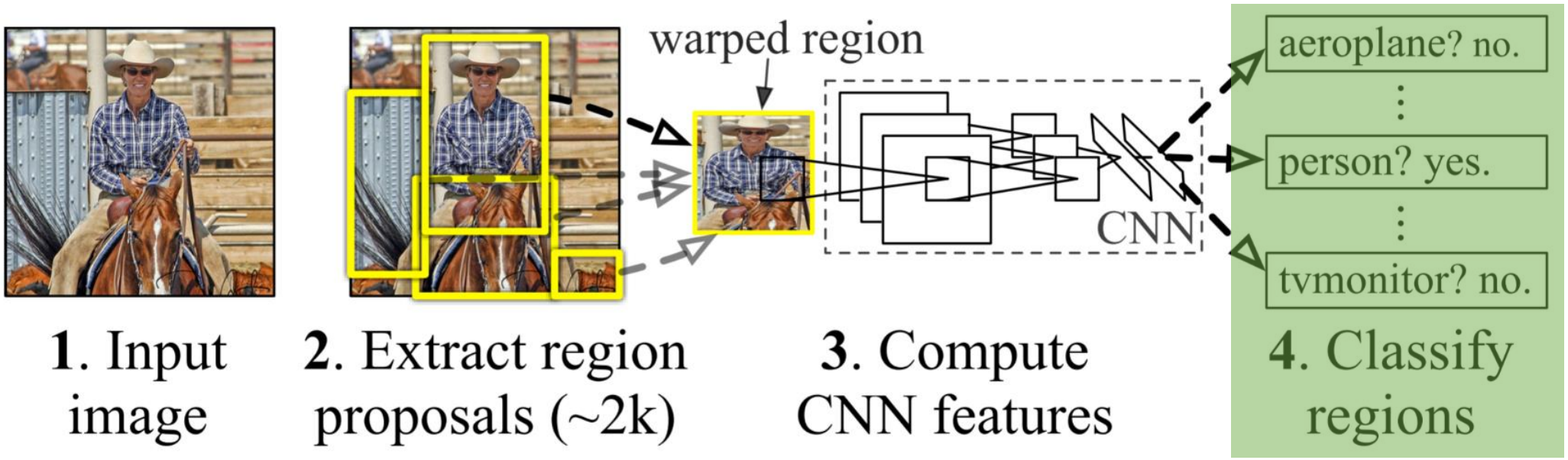
- (1) takes an input image
- (2) extracts around 2000 bottom-up regions using selective search
  - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013
- **(3) computes features for each region using a large convolutional neural network (CNN)**
  - AlexNet is used to compute the features **(227×227 pixels)**



Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**

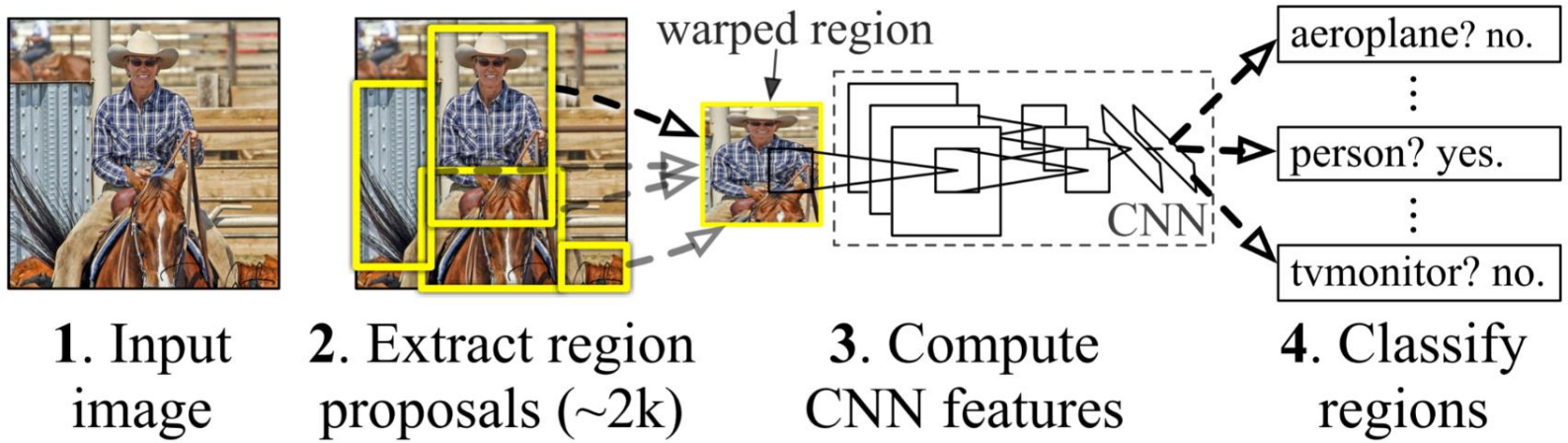
- (1) takes an input image
- (2) extracts around 2000 bottom-up regions using selective search
  - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013
- (3) computes features for each region using a large convolutional neural network (CNN)
  - AlexNet is used to compute the features (227×227 pixels)
- **(4) classifies each region using class-specific linear SVMs**



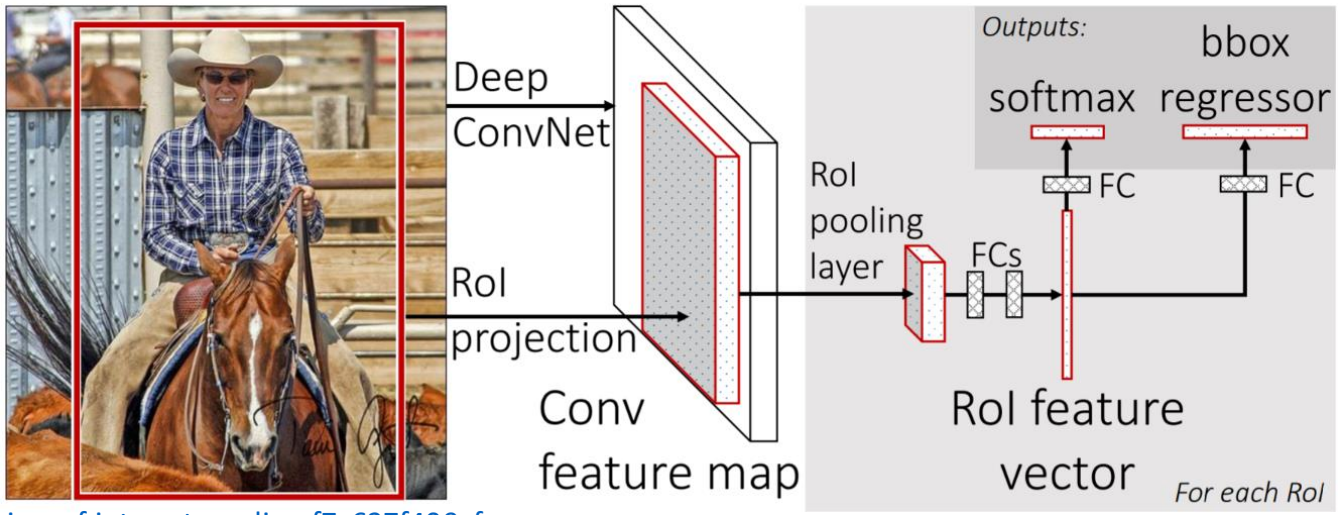
Region-Based CNNs (Fast R-CNNs)

- **R-CNN vs. Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape

R-CNN



Fast R-CNN

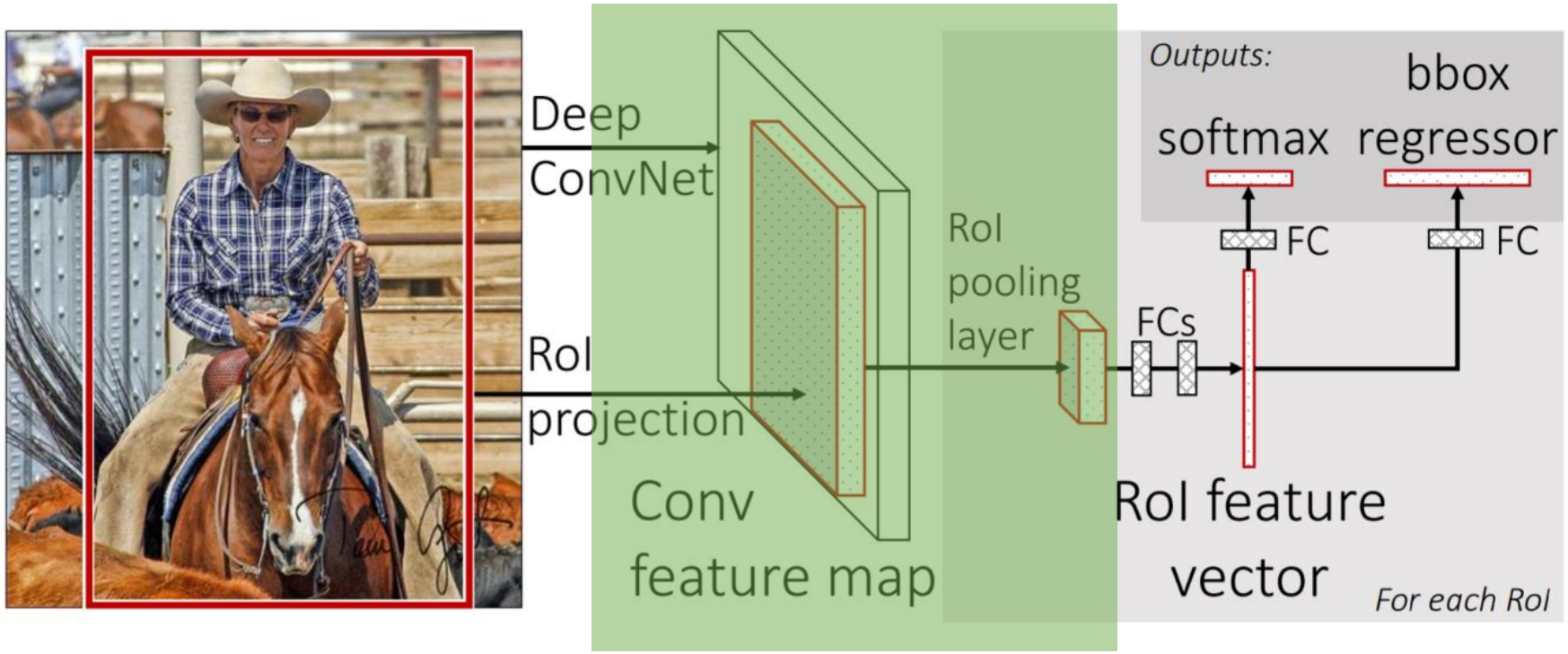


[1] https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af
[2] https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44

Region-Based CNNs (Fast R-CNNs)

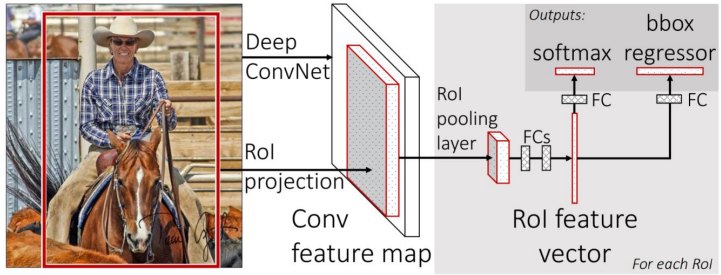
- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape

- **ROI Pooling (Region of interest pooling)** solves the problem
  - for every ROI (**proposal**) from the input, feature map which corresponds to that ROI is selected
  - transform this feature-map into a fixed dimension map



[1] https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af
[2] https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44

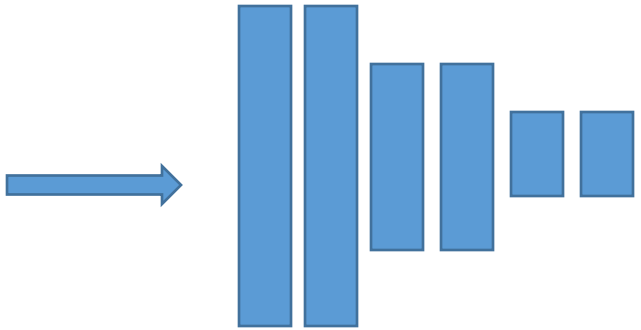
Region-Based CNNs (Fast R-CNNs)



- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape

- Example ([1, 2]):
  - input image size is **1056x640**
  - after several conv and pool operations the output feature map size is **reduced to 66x40**
  - **this feature map is used by ROI pooling layer**

1056x640

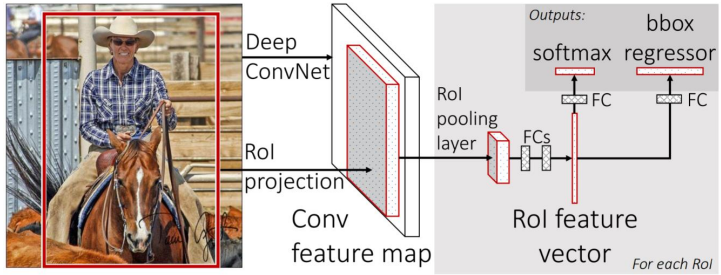


VGG-16

66x40 feature map

[1] https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af
[2] https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44

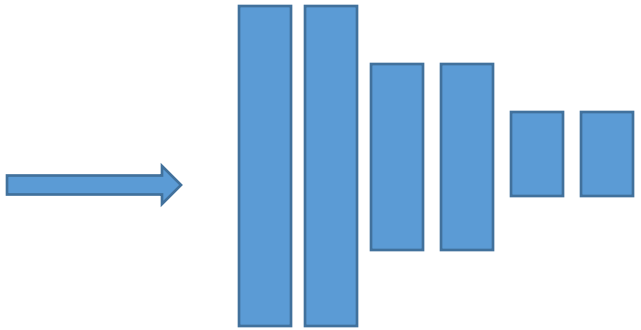
Region-Based CNNs (Fast R-CNNs)



- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape

- Example ([1, 2]):
  - input image size is **1056x640**
  - after several conv and pool operations the output feature map size is **reduced to 66x40**
  - **this feature map is used by ROI pooling layer**
  - **Get RoIs from the feature map?**

1056x640



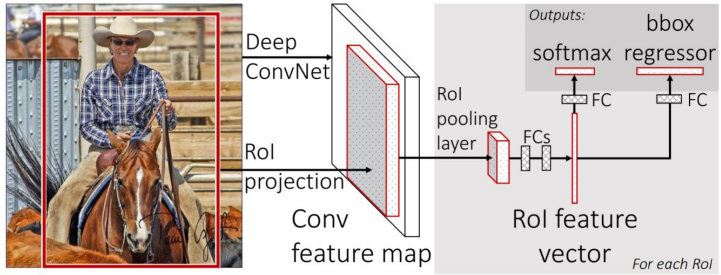
VGG-16

66x40 feature map

[1] https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af
[2] https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44

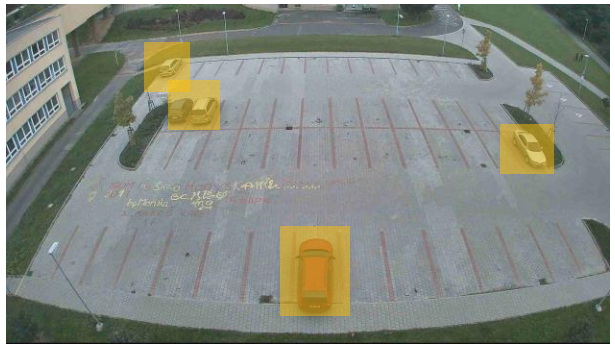
EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Region-Based CNNs (Fast R-CNNs)

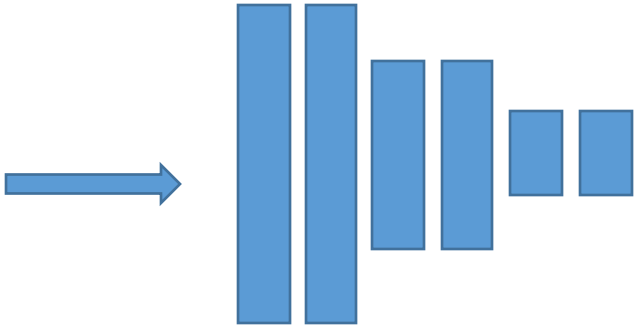


- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape

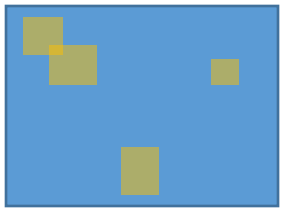
- Example ([1, 2]): input image size is **1056x640**
  - after several conv and pool operations the output feature map size is **reduced to 66x40**
  - **this feature map is used by ROI pooling layer**
  - **Get RoIs from the feature map?**
  - The extracted regions of interest (**proposal**) are generated based on input image size, so we need to rescale these regions to feature map size. In this particular case by 16 (1056/66=16 or 640/40=16).



1056x640



VGG-16

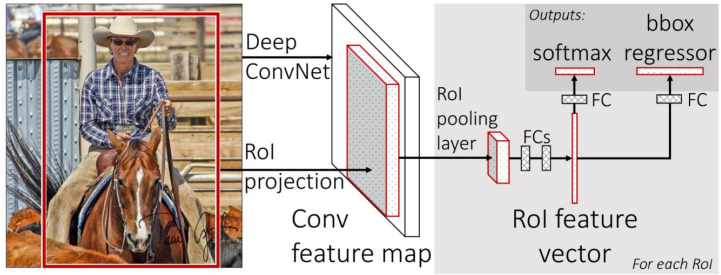


66x40 feature map

[1] https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af
[2] https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Region-Based CNNs (Fast R-CNNs)



- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape

- Example ([1, 2]):
  - input image size is **1056x640**
  - after several conv and pool operations the output feature map size is **reduced to 66x40**
  - **this feature map is used by ROI pooling layer**
  - **Get RoIs from the feature map?**
  - The extracted regions of interest (**proposal**) are generated based on input image size, so we need to rescale these regions to feature map size. In this particular case by 16 (1056/66=16 or 640/40=16).
  - **For every proposal in the input proposals, we take the corresponding feature map section and divide that section into W*H. After that take the maximum element of each block and copy to the output. So as the output we obtain fixed dimension feature map irrespective of the various sizes of the input proposals.**

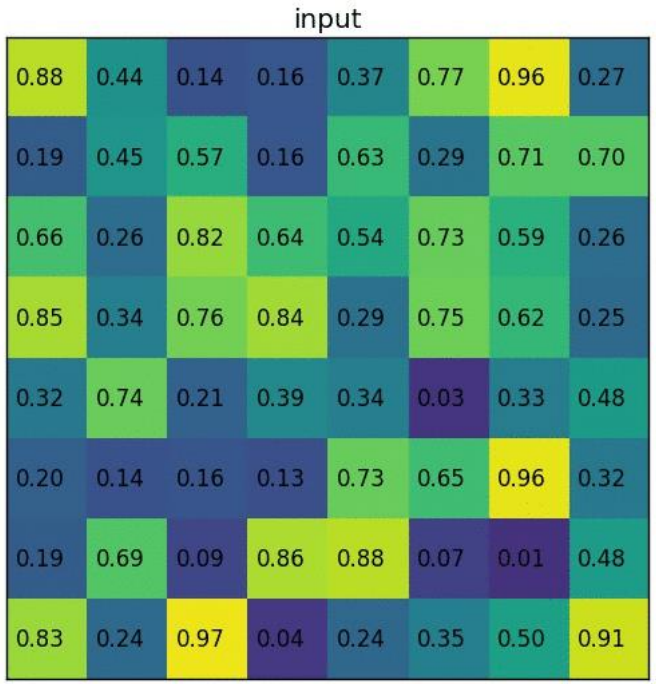
```
Scaled_Proposals = Proposals * spatial_scale
for every ROI in Scaled_Proposals:
    fmap_subset = feature_map[ROI] (Feature_map for that ROI)
    Divide fmap_subset into P_wxP_h blocks (ex: 6*6 blocks)
    Take the maximum element of each block and copy to output block
```

[1] https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af
[2] https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44

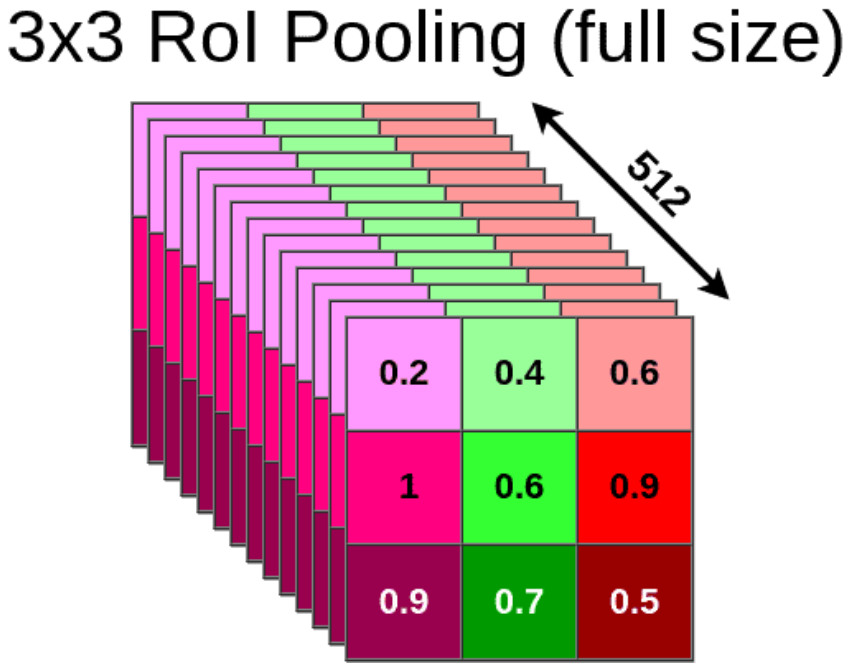
Region-Based CNNs (Fast R-CNNs)

- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape

- **ROI Pooling (Region of interest pooling)** solves the problem



source: https://miro.medium.com/max/840/1*5V5mycIRNu-mK-rPywL57w.gif



After the pooling process, (for example) the 3x3x512 matrixes can be used as input for FC layers for further processing. For each region we obtain fixed size of vector.

[1] https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af
[2] https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44

Region-Based CNNs (Faster R-CNNs)

- **Faster R-CNN – 2015**

- Selective search in R-CNN and Fast R-CNN is replaced by **Region Proposal Network**

- two modules:
  - 1. module is a deep fully convolutional network that proposes regions
  - 2. module is the Fast R-CNN detector that uses the proposed regions

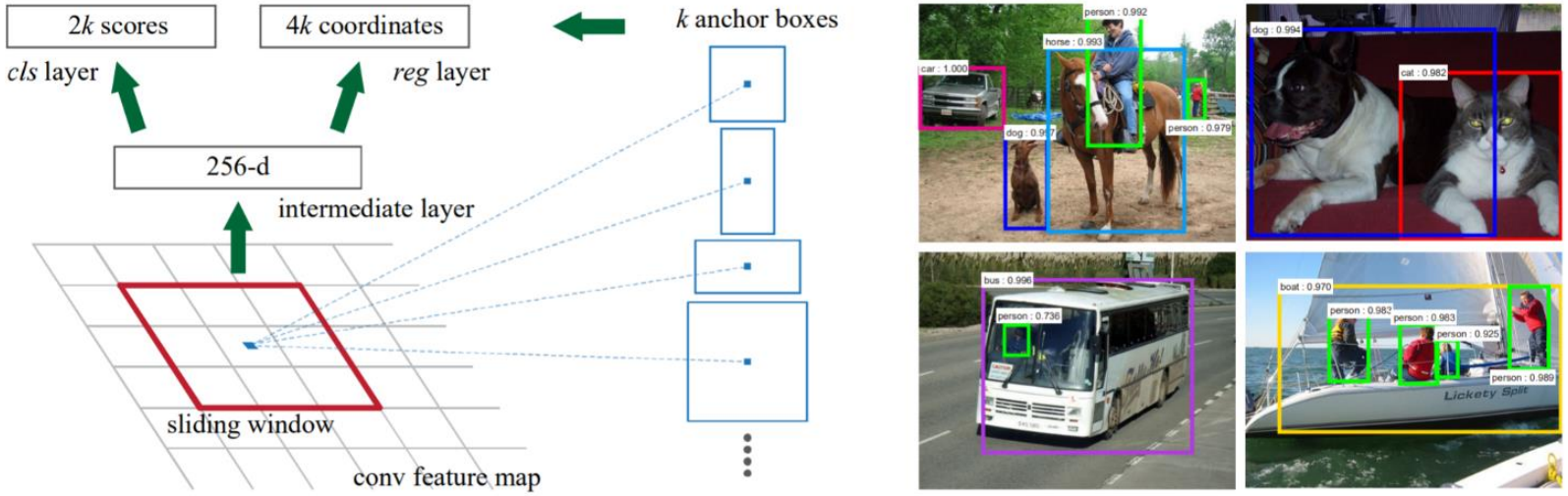
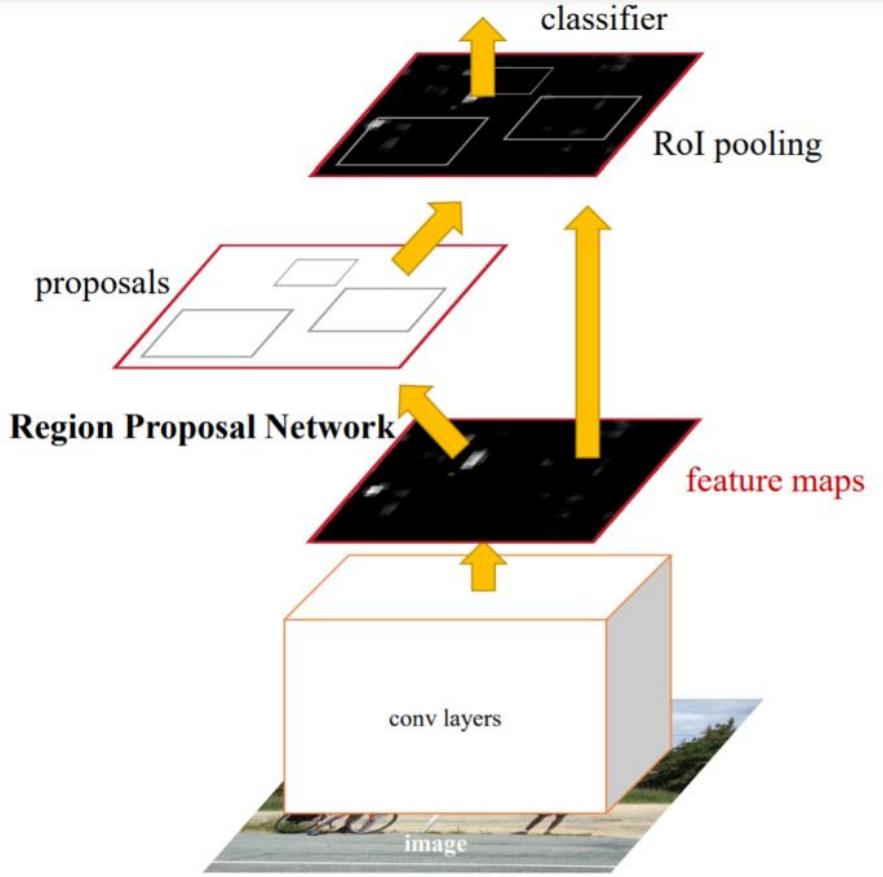


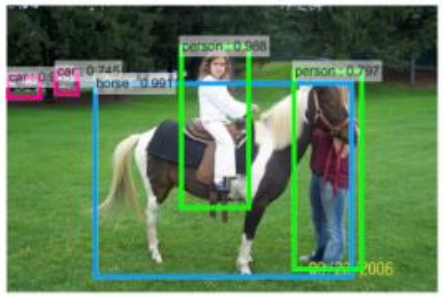
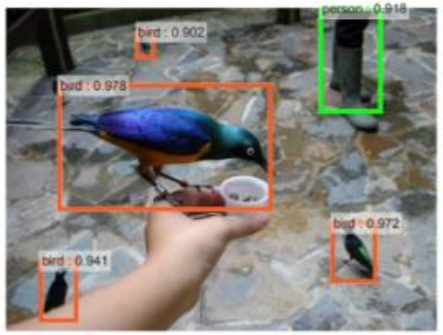
Figure 3: **Left**: Region Proposal Network (RPN). **Right**: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

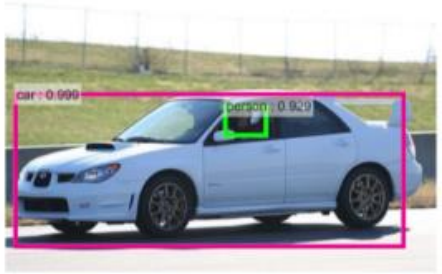
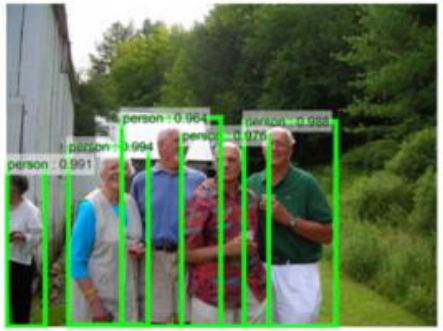
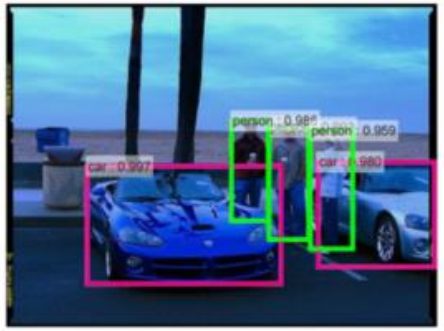


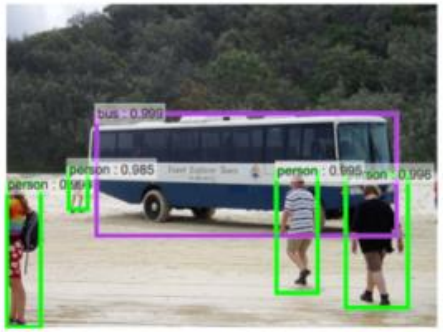
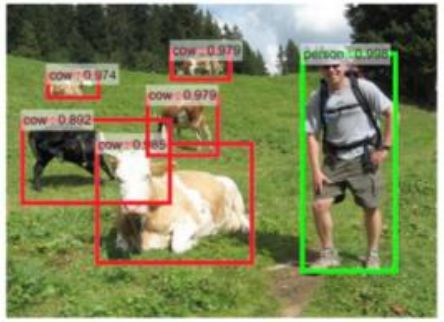
https://arxiv.org/abs/1506.01497

Region-Based CNNs (Faster R-CNNs)

- **Faster R-CNN – 2015**

https://arxiv.org/abs/1506.01497

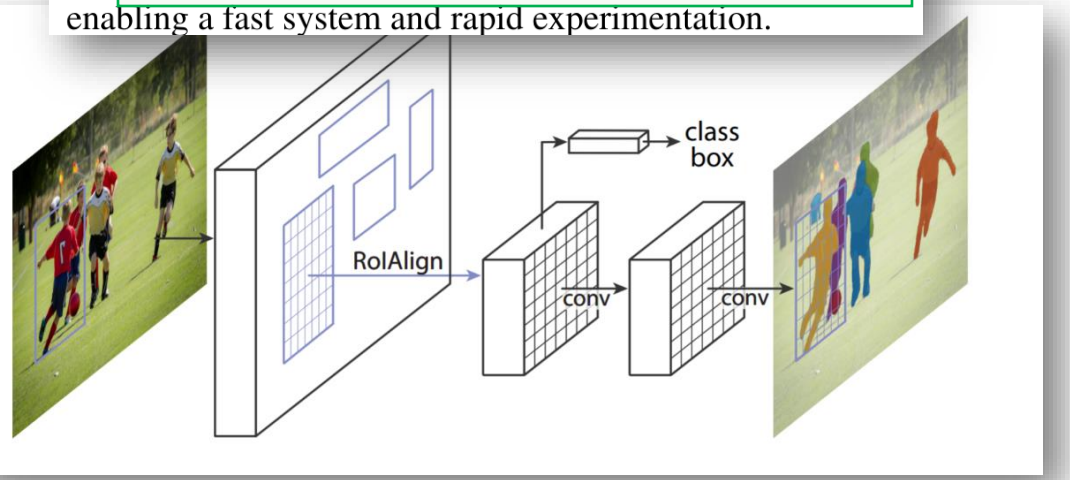
EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Region-Based CNNs (Mask R-CNNs)

Mask R-CNN

Kaiming He Georgia Gkioxari Piotr Dollár Ross Girshick
Facebook AI Research (FAIR)

Our method, called *Mask R-CNN*, extends Faster R-CNN [36] by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in *parallel* with the existing branch for classification and bounding box regression (Figure 1). The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. Mask R-CNN is simple to implement and train given the Faster R-CNN framework, which facilitates a wide range of flexible architecture designs. Additionally, the mask branch only adds a small computational overhead, enabling a fast system and rapid experimentation.



https://arxiv.org/abs/1506.01497
https://arxiv.org/abs/1703.06870

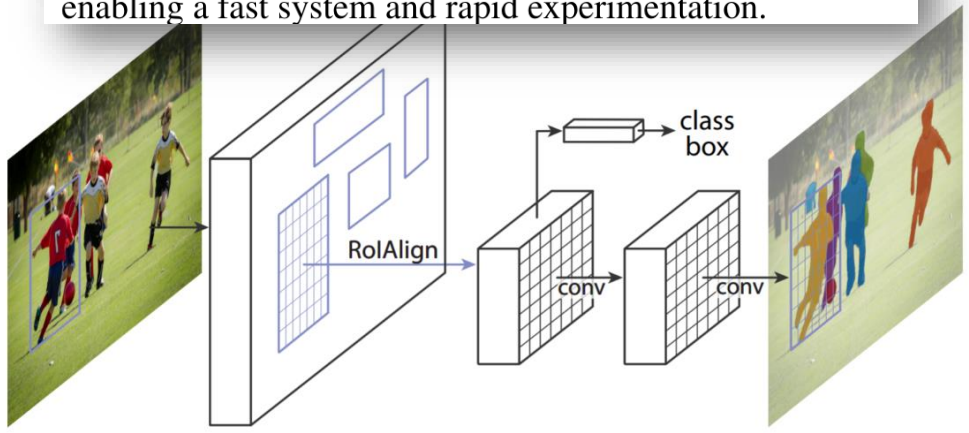
EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Mask R-CNNs

Mask R-CNN

Kaiming He Georgia Gkioxari Piotr Dollár Ross Girshick
Facebook AI Research (FAIR)

Our method, called *Mask R-CNN*, extends Faster R-CNN [36] by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in *parallel* with the existing branch for classification and bounding box regression (Figure 1). The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. Mask R-CNN is simple to implement and train given the Faster R-CNN framework, which facilitates a wide range of flexible architecture designs. Additionally, the mask branch only adds a small computational overhead, enabling a fast system and rapid experimentation.



We denote the *backbone* architecture using the nomenclature *network-depth-features*. We evaluate ResNet [19] and ResNeXt [45] networks of depth 50 or 101 layers. The original implementation of Faster R-CNN with ResNets [19] extracted features from the final convolutional layer of the 4-th stage, which we call C4. This backbone with ResNet-50, for example, is denoted by ResNet-50-C4. This is a common choice used in [19, 10, 21, 39].

We also explore another more effective backbone recently proposed by Lin *et al.* [27], called a Feature Pyramid Network (FPN). FPN uses a top-down architecture with lateral connections to build an in-network feature pyramid from a single-scale input. Faster R-CNN with an FPN backbone extracts RoI features from different levels of the feature pyramid according to their scale, but otherwise the rest of the approach is similar to vanilla ResNet. Using a ResNet-FPN backbone for feature extraction with Mask R-CNN gives excellent gains in both accuracy and speed. For further details on FPN, we refer readers to [27].

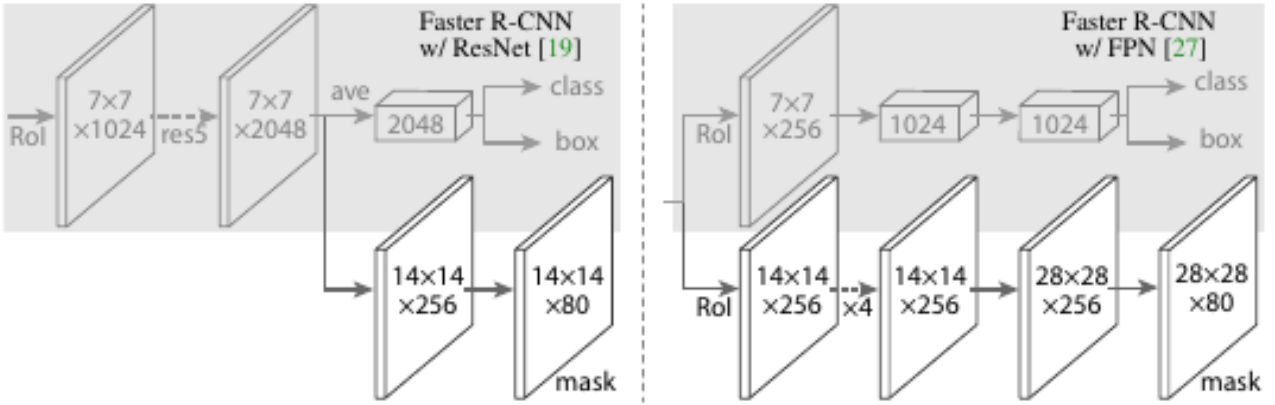


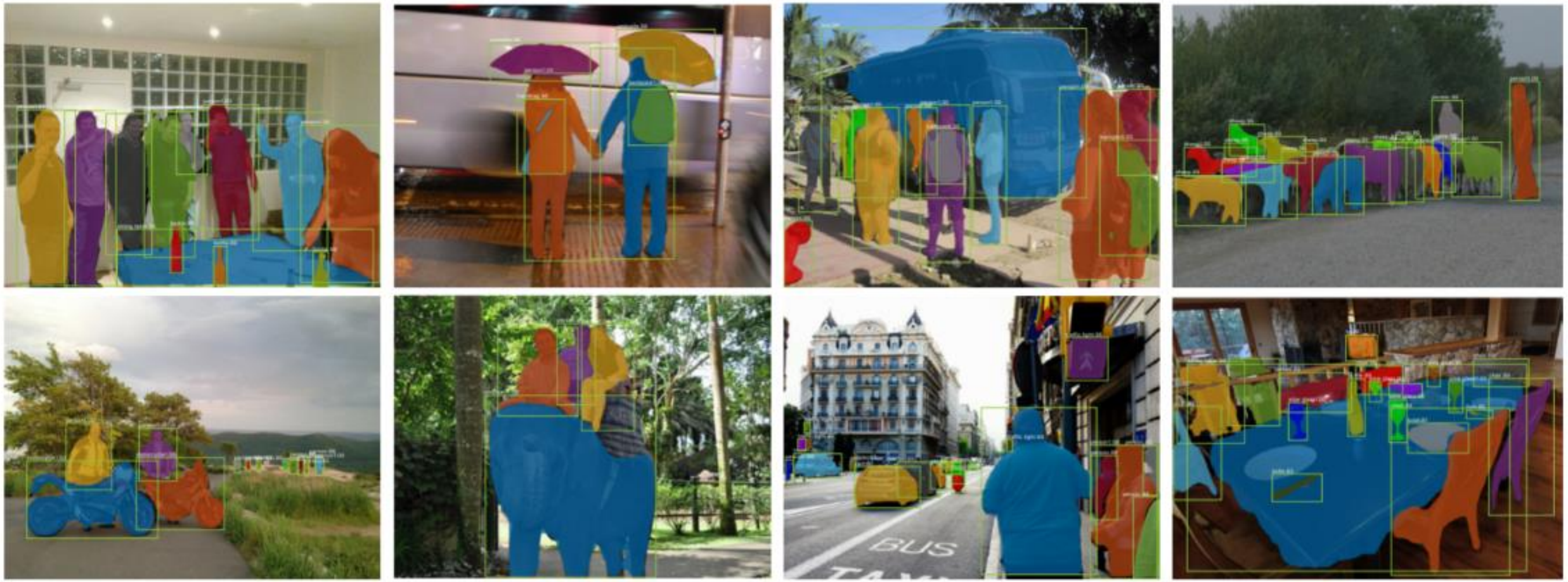
Figure 4. **Head Architecture**: We extend two existing Faster R-

https://arxiv.org/abs/1506.01497
https://arxiv.org/abs/1703.06870

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Region-Based CNNs (Mask R-CNNs)

- **Mask R-CNN – 2017**



https://arxiv.org/abs/1506.01497
https://arxiv.org/abs/1703.06870

Region-Based CNNs (Faster R-CNNs)

- **EXAMPLE Faster R-CNN – PyTorch**

https://pytorch.org/vision/main/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html

```
def main():

    cv2.namedWindow("detection", 0)
    print("main")

    test_images = [img for img in glob.glob("test_images/*.jpg")]
    test_images.sort()

    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.eval().to(device)

    transformRCNN = transforms.Compose([
        transforms.ToTensor(),
    ])
```

https://arxiv.org/abs/1506.01497
https://arxiv.org/abs/1703.06870

Region-Based CNNs (Faster R-CNNs)

- **EXAMPLE Faster R-CNN – PyTorch**

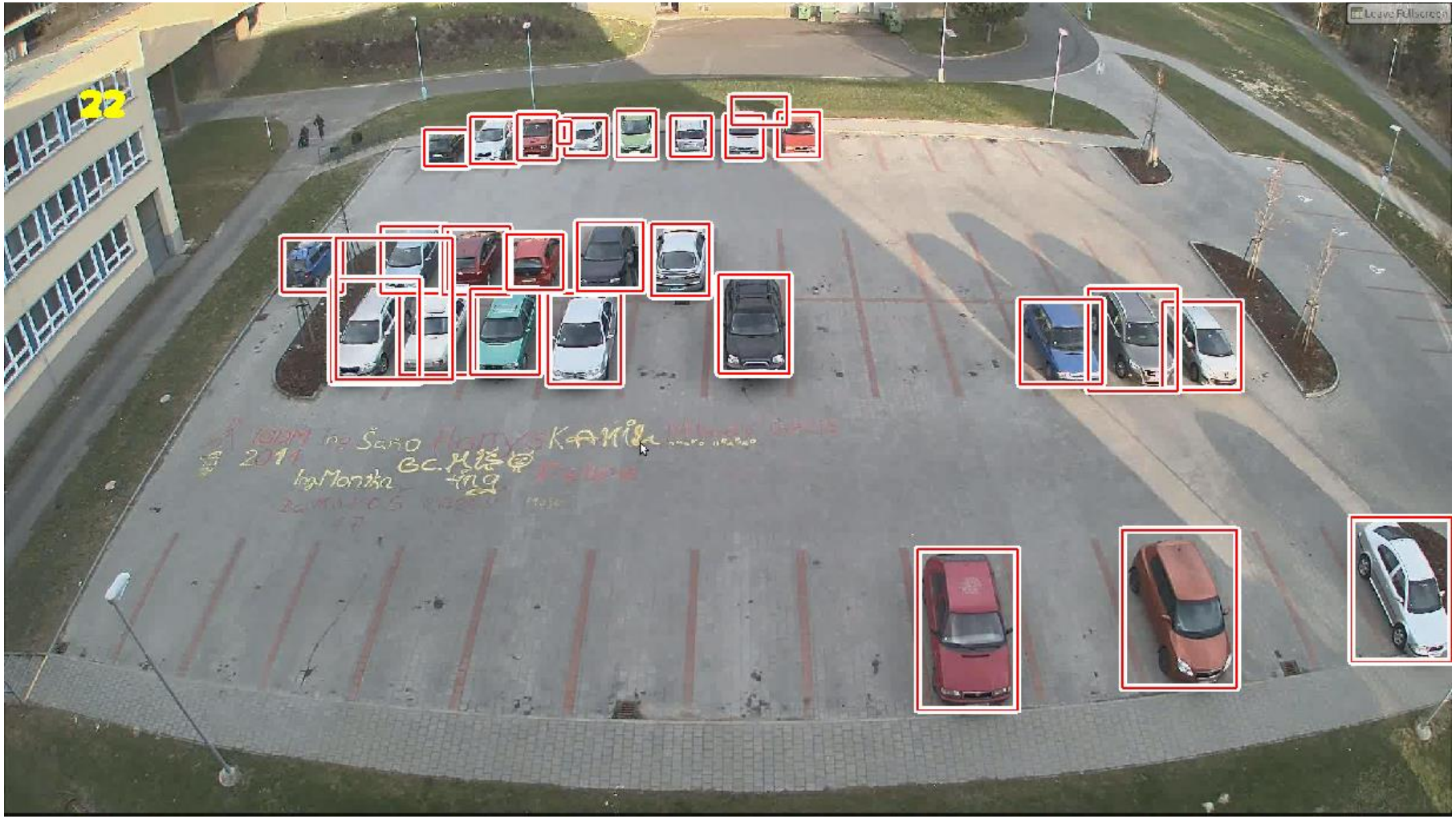
coco_names = ['__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N/A', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table', 'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']

```
for img in test_images:
    one_img = cv2.imread(img)
    one_img_paint = one_img.copy()

    one_img_rgb = cv2.cvtColor(one_img, cv2.COLOR_BGR2RGB)
    img_pil = Image.fromarray(one_img_rgb)
    imageRCNN = transformRCNN(img_pil).to(device)
    imageRCNN = imageRCNN.unsqueeze(0)
    outputsRCNN = model(imageRCNN)
    pred_classes = [coco_names[i] for i in outputsRCNN[0]['labels'].cpu().numpy()]
    pred_scores = outputsRCNN[0]['scores'].detach().cpu().numpy()
    pred_bboxes = outputsRCNN[0]['boxes'].detach().cpu().numpy()

    print(pred_scores)
    print(pred_classes)
```

https://arxiv.org/abs/1506.01497
https://arxiv.org/abs/1703.06870

Region-Based CNNs (Faster R-CNNs)

- **EXAMPLE Faster R-CNN – PyTorch**



https://arxiv.org/abs/1506.01497
https://arxiv.org/abs/1703.06870

YOLO

- **YOLO - You Only Look Once**

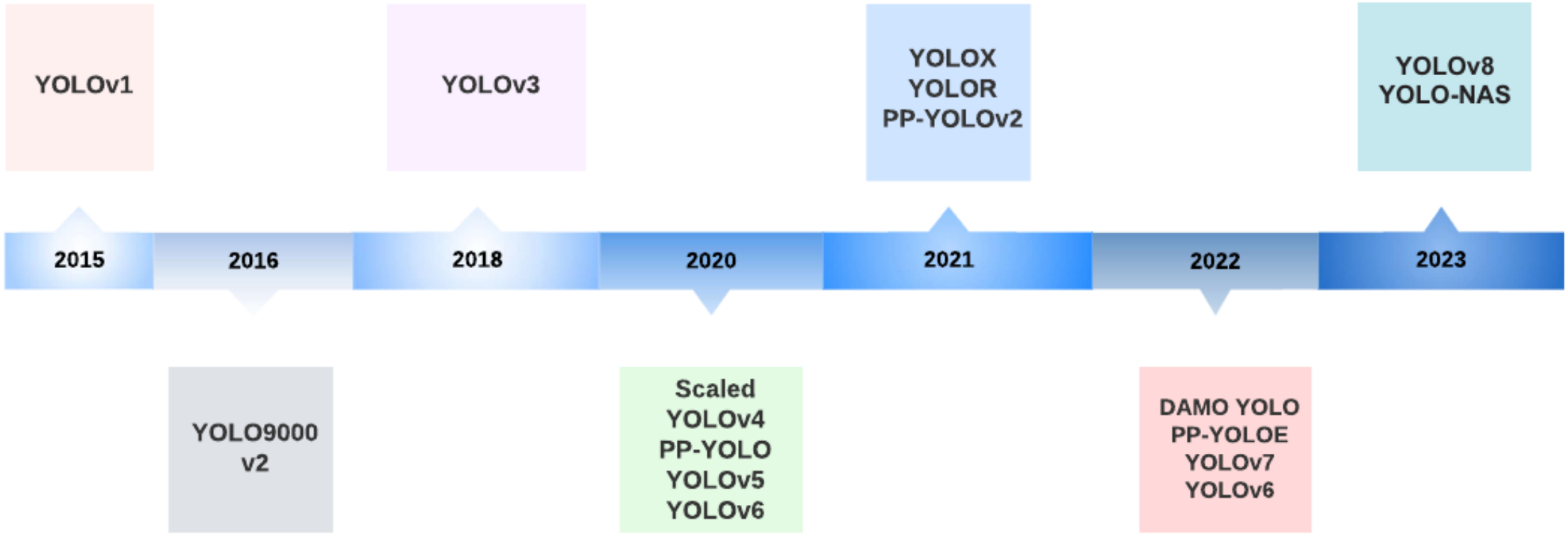


Figure 1: A timeline of YOLO versions.

Terven, J.R., & Córdova-Esparza, D. (2023). A Comprehensive Review of YOLO: From YOLOv1 and Beyond. https://arxiv.org/abs/2304.00501

https://arxiv.org/abs/1506.02640

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

YOLO

- **YOLOv1 - You Only Look Once**

You Only Look Once:
Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

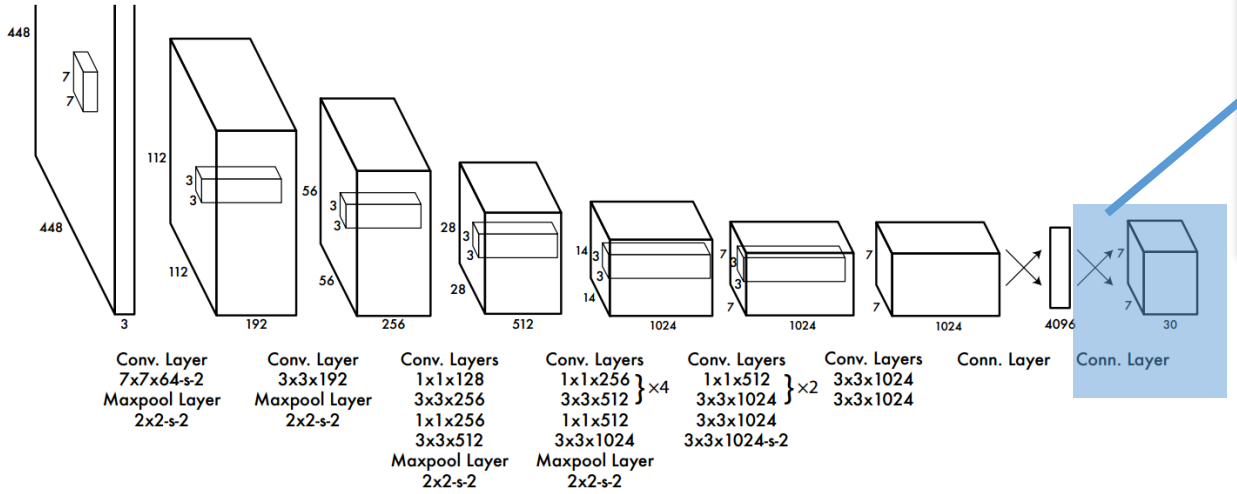


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

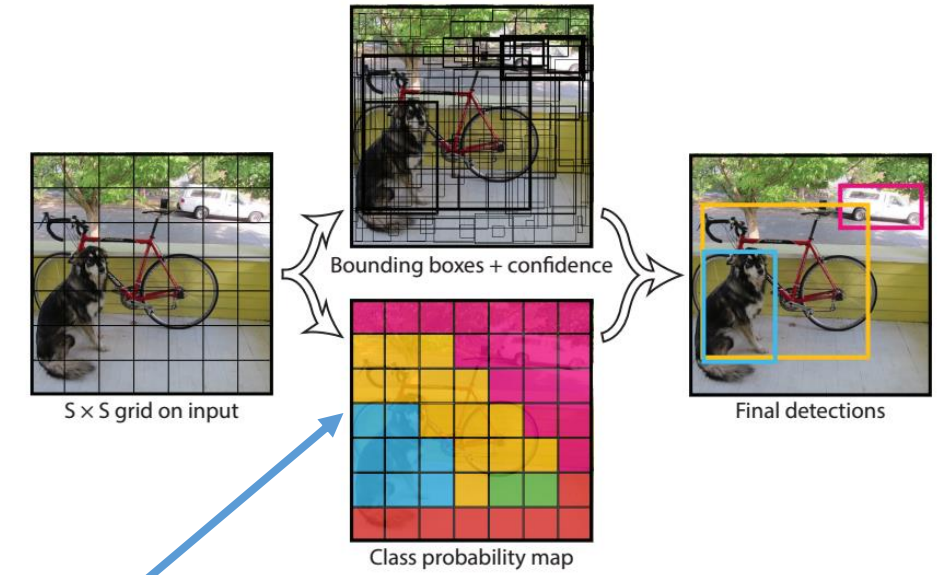


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

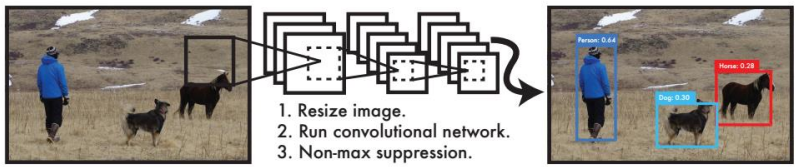
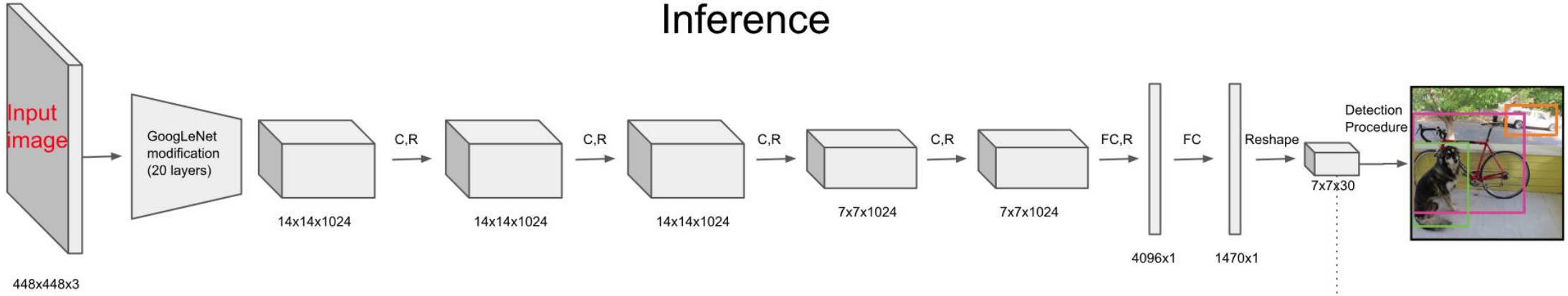


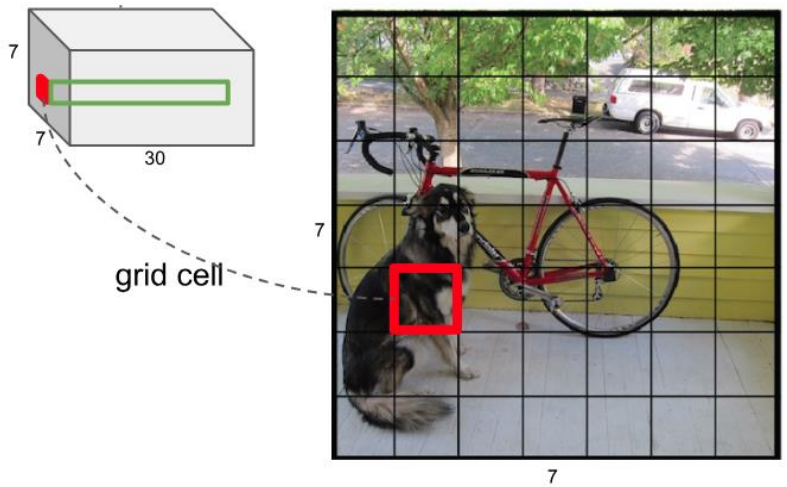
Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

Inference



Tensor values interpretation



YOLOv1 - You Only Look Once

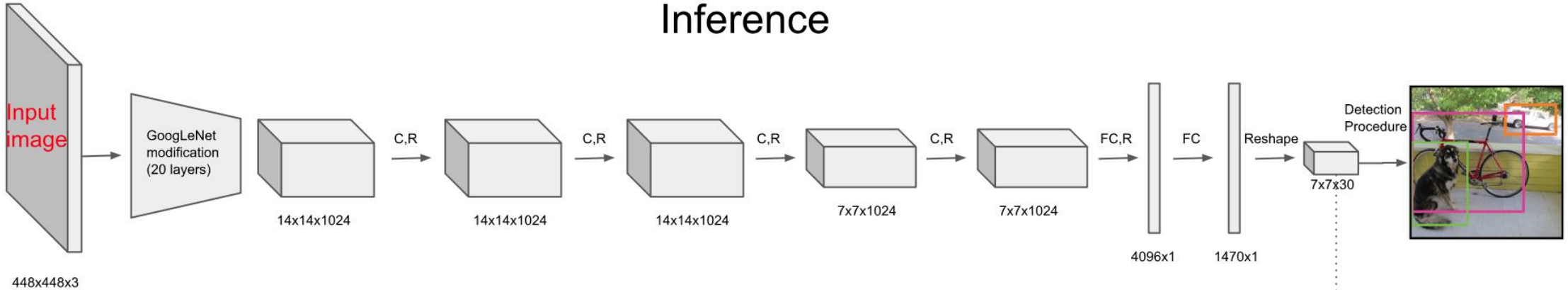
- **divide image into grid cells**

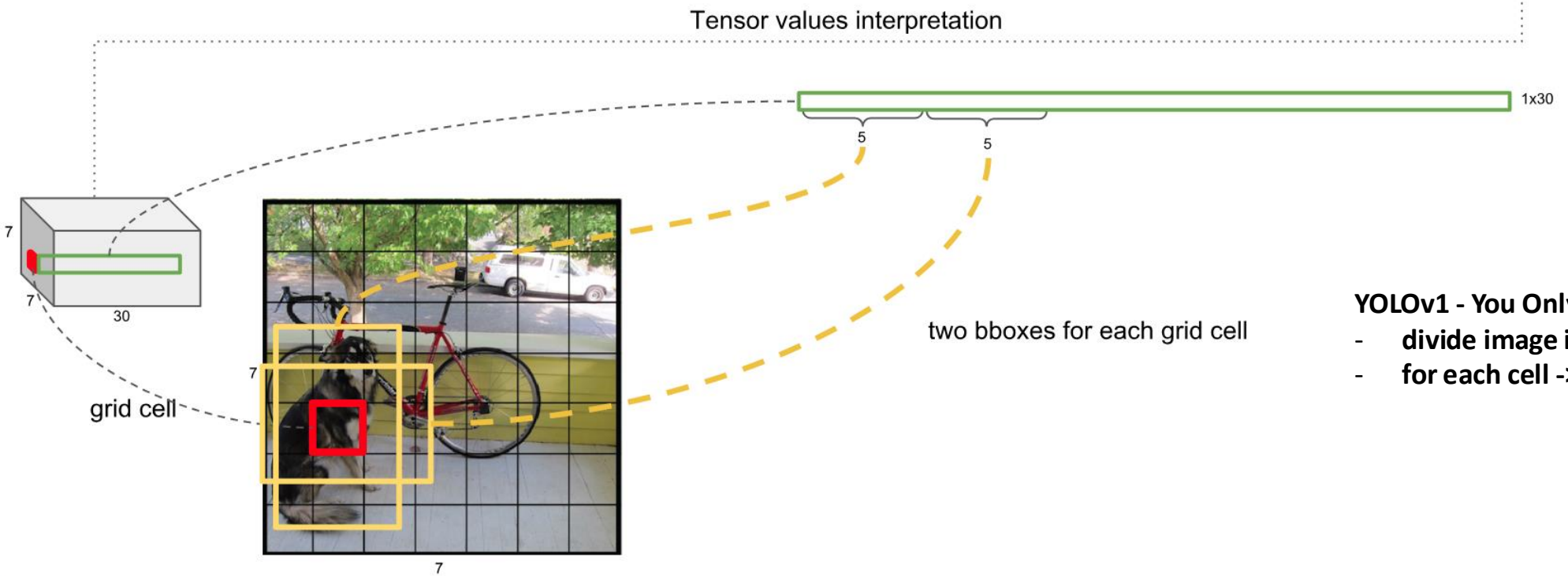
deepsystems.io

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

https://github.com/watersink/yolov1_tutorial
https://docs.google.com/presentation/d/1aeRvtKG21KHdD5lg6Hgyhx5rPq_ZOsGjG5rJ1HP7BbA/edit#slide=id.g137784ab86_4_4011

Inference





YOLOv1 - You Only Look Once

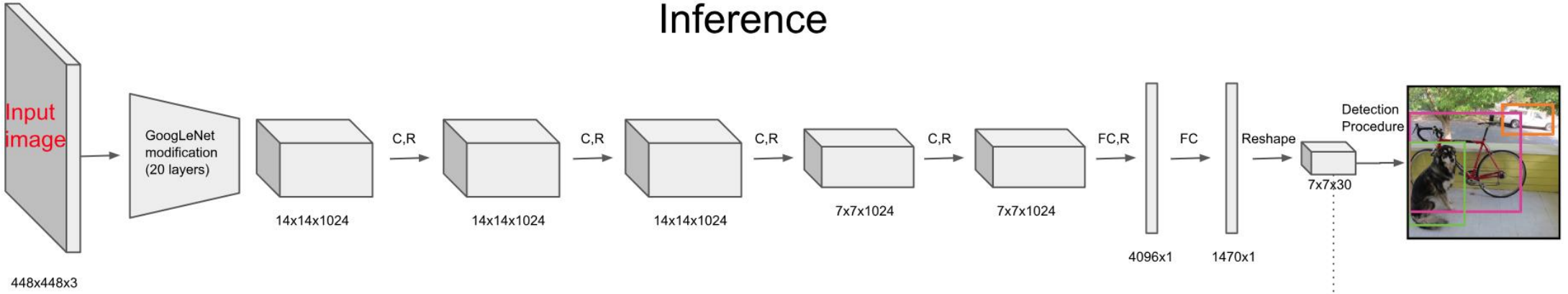
- **divide image into grid cells**
- **for each cell -> bounding box predictions**

deepsystems.io

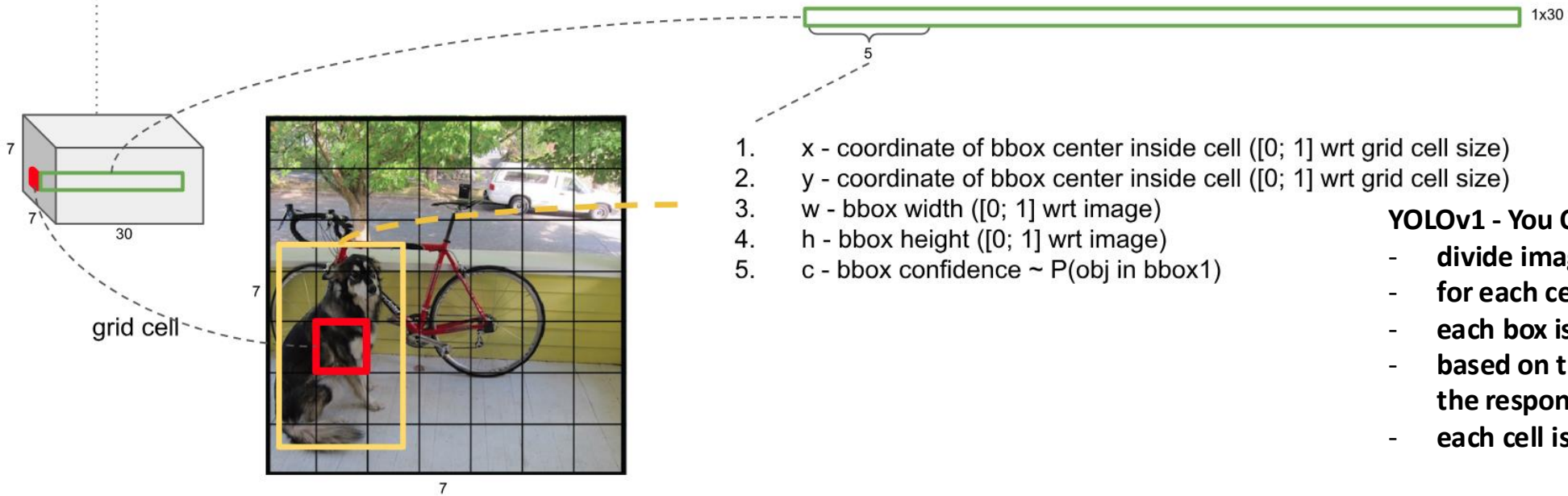
https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

https://github.com/watersink/yolov1_tutorial
https://docs.google.com/presentation/d/1aeRvtKG21KHdD5lg6Hgyhx5rPq_ZOsGjG5rJ1HP7BbA/edit#slide=id.g137784ab86_4_4011

Inference



Tensor values interpretation



1. x - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
2. y - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
3. w - bbox width ([0; 1] wrt image)
4. h - bbox height ([0; 1] wrt image)
5. c - bbox confidence ~ P(obj in bbox1)

YOLOv1 - You Only Look Once

- **divide image into grid cells**
- **for each cell -> bounding box predictions**
- **each box is described by x, y, w, h, Pc**
- **based on the middle point of each object, the responsible cell is selected**
- **each cell is responsible for one object**

deepsystems.io

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

https://github.com/watersink/yolov1_tutorial
https://docs.google.com/presentation/d/1aeRvtKG21KHdD5lg6Hgyhx5rPq_ZOsGjG5rJ1HP7BbA/edit#slide=id.g137784ab86_4_4011

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

YOLO

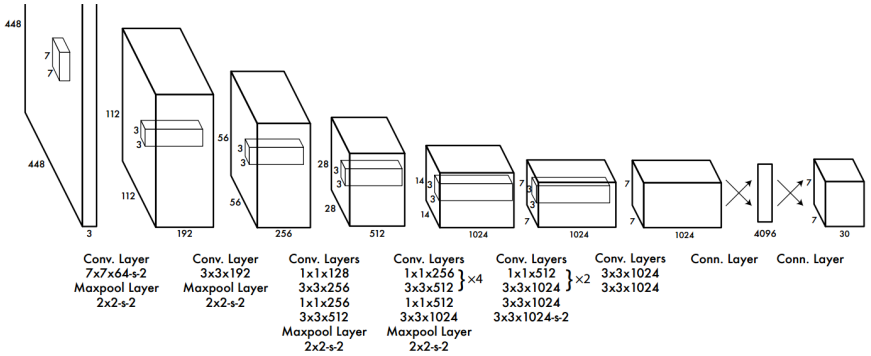
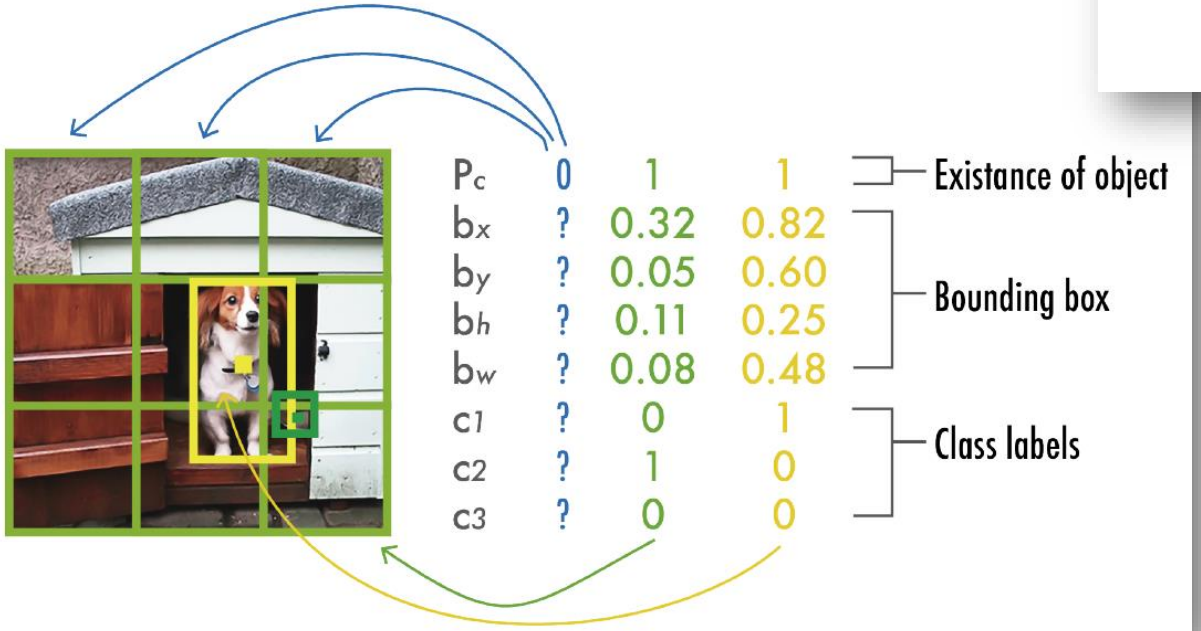


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.



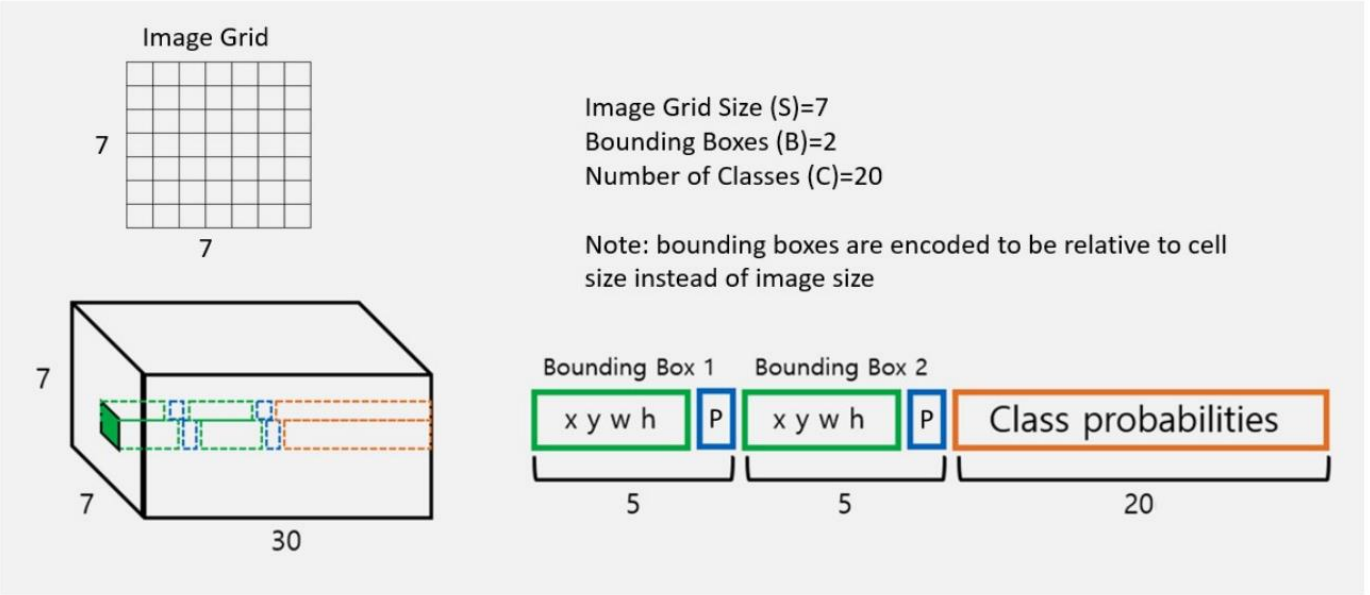


Fig. 3. YOLO Tensor Structure

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Table 1: Real-Time Systems on PASCAL VOC 2007. Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

YOLO

- **YOLOv1 – multi-part loss function**

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

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

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)$$

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

YOLO

- **YOLOv1 – multi-part loss function**

localization loss

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

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

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)$$

Sum of Square Error (SSE) of how close is the position (and size) of bounding box to the ground truth

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

YOLO

- **YOLOv1 – multi-part loss function**

localization loss

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

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

Sum of Square Error (SSE) of how close is the position (and size) of bounding box to the ground truth

confidence loss

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

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

SSE of how close is the confidence score to the ground truth

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)$$

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

YOLO

- **YOLOv1 – multi-part loss function**

localization loss

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

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

Sum of Square Error (SSE) of how close is the position (and size) of bounding box to the ground truth

confidence loss

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

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

SSE of how close is the confidence score to the ground truth

classification loss

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

SSE of how close is the predicted class probabilities to the ground truth

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

YOLO vs. Fast R-CNN

Figure 4 shows the breakdown of each error type averaged across all 20 classes.

YOLO struggles to localize objects correctly. Localization errors account for more of YOLO's errors than all other sources combined. Fast R-CNN makes much fewer localization errors but far more background errors. 13.6% of it's top detections are false positives that don't contain any objects. Fast R-CNN is almost 3x more likely to predict background detections than YOLO.

2.4. Limitations of YOLO

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds.

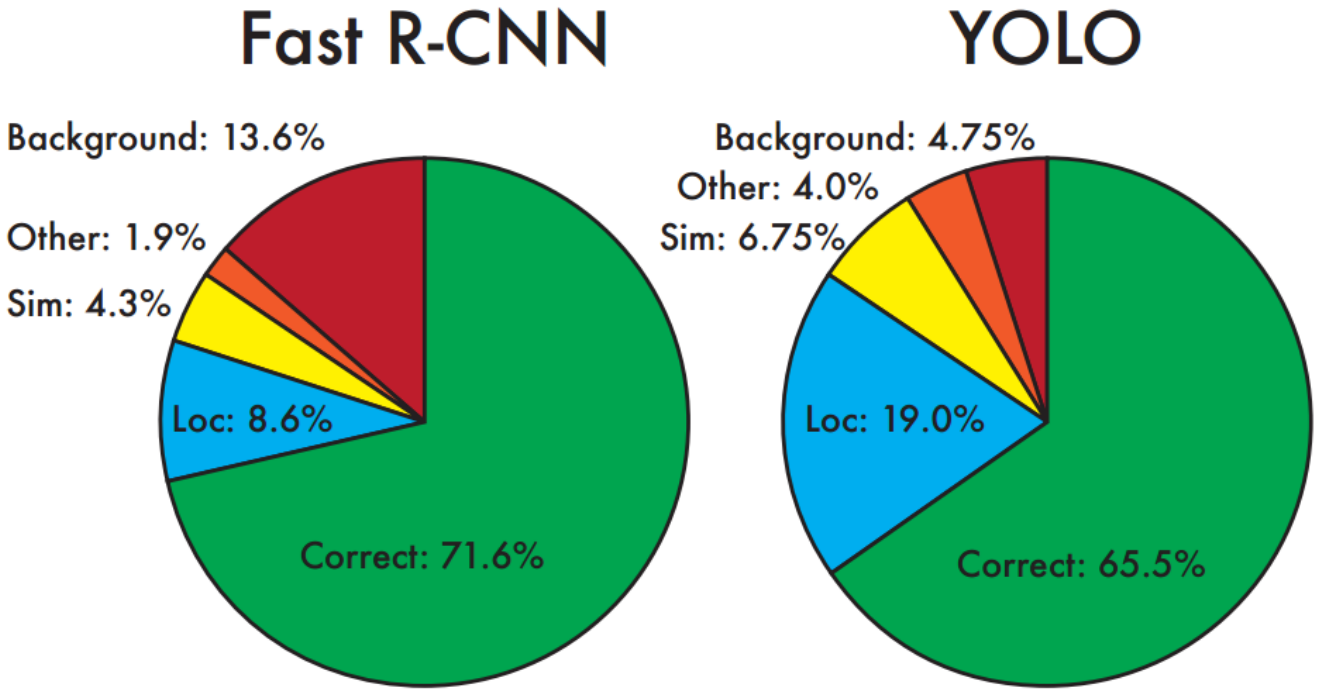


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

https://arxiv.org/abs/1506.02640

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

YOLO

https://youtu.be/NM6lrxy0bxs?si=v1aZzezBOguAgra7

https://arxiv.org/abs/1506.02640

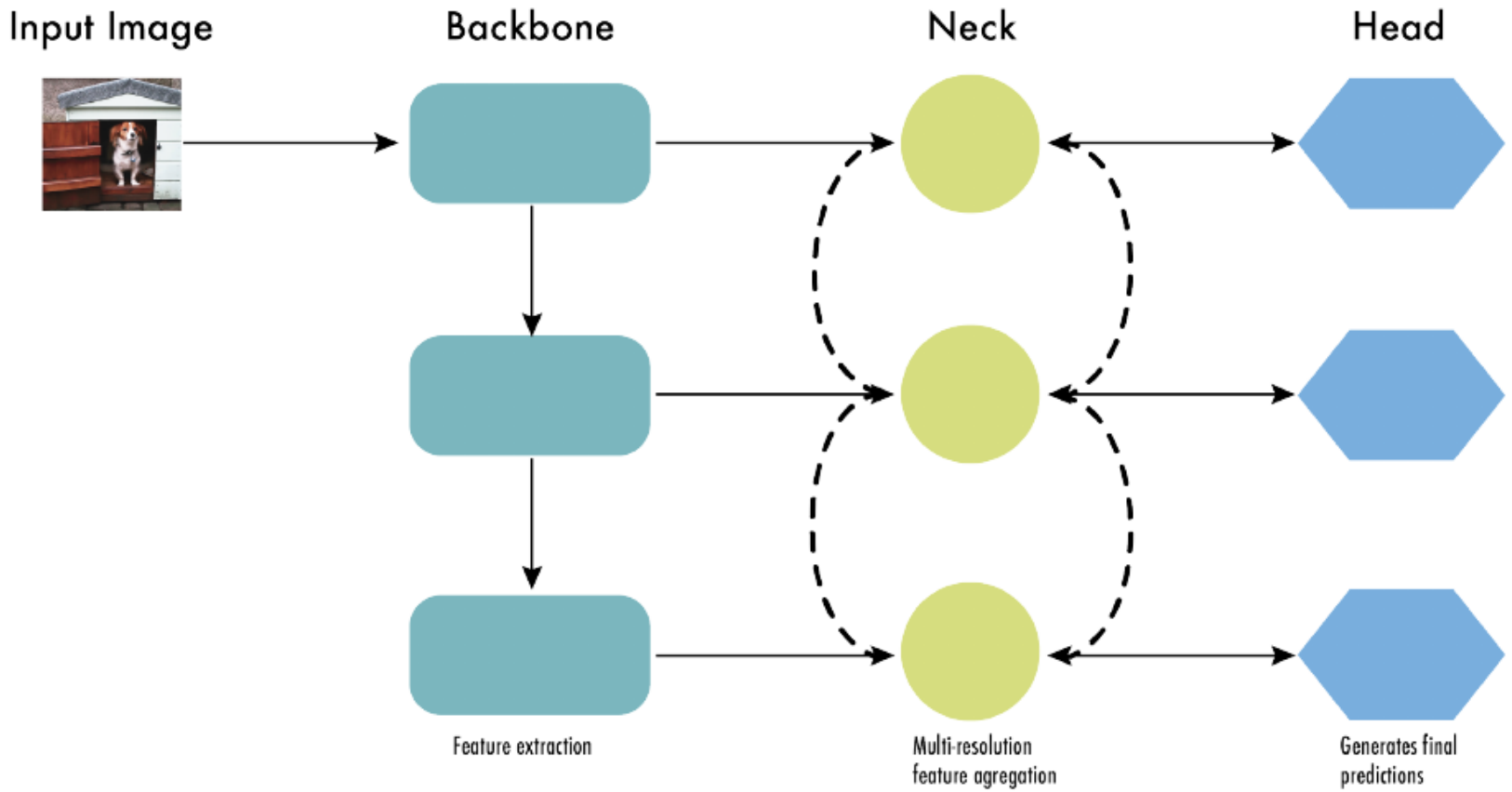


Figure 10: The architecture of modern object detectors can be described as the backbone, the neck, and the head. The backbone, usually a convolutional neural network (CNN), extracts vital features from the image at different scales. The neck refines these features, enhancing spatial and semantic information. Lastly, the head uses these refined features to make object detection predictions.

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

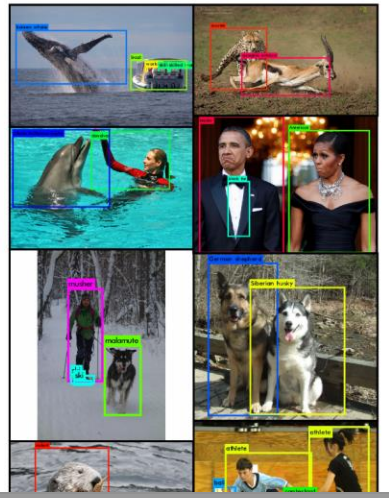
YOLOv2

YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.



The YOLO framework uses a custom network based on the Googlenet architecture [19]. This network is faster than VGG-16, only using 8.52 billion operations for a forward pass. However, it's accuracy is slightly worse than VGG-16. For single-crop, top-5 accuracy at 224×224, YOLO's custom model gets 88.0% ImageNet compared to 90.0% for VGG-16.

Darknet-19. We propose a new classification model to be used as the base of YOLOv2. Our model builds off of prior work on network design as well as common knowledge in the field. Similar to the VGG models we use mostly 3×3 filters and double the number of channels after every pooling step [17]. Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions [9]. We use batch normalization to stabilize training, speed up convergence, and regularize the model [7].

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [7]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size. Since our model downsamples by a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

https://arxiv.org/abs/1612.08242

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

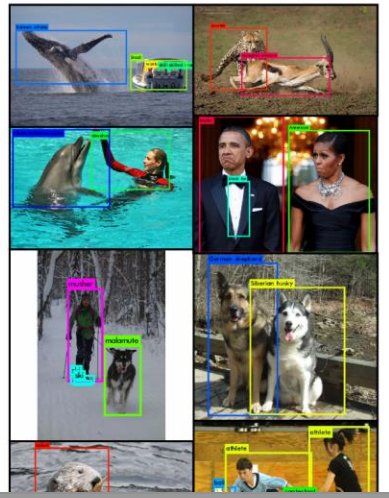
YOLOv2

YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.



The YOLO framework uses a custom network based on the Googlenet architecture [19]. This network is faster than VGG-16, only using 8.52 billion operations for a forward pass. However, it's accuracy is slightly worse than VGG-16. For single-crop, top-5 accuracy at 224×224, YOLO's custom model gets 88.0% ImageNet compared to 90.0% for VGG-16.

Darknet-19. We propose a new classification model to be used as the base of YOLOv2. Our model builds off of prior work on network design as well as common knowledge in the field. Similar to the VGG models we use mostly 3×3 filters and double the number of channels after every pooling step [17]. Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions [9]. We use batch normalization to stabilize training, speed up convergence, and regularize the model [7].

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [7]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size. Since our model downsamples by a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

https://arxiv.org/abs/1612.08242

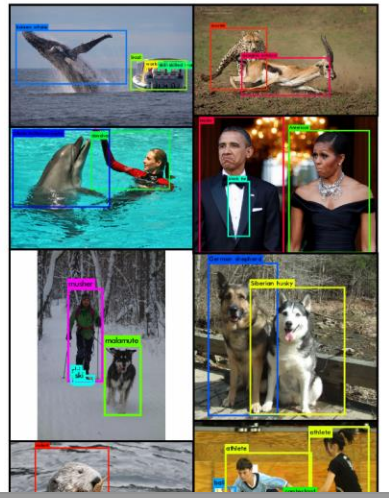
YOLOv2

YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.



The YOLO framework uses a custom network based on the Googlenet architecture [19]. This network is faster than VGG-16, only using 8.52 billion operations for a forward pass. However, it's accuracy is slightly worse than VGG-16. For single-crop, top-5 accuracy at 224×224, YOLO's custom model gets 88.0% ImageNet compared to 90.0% for VGG-16.

Darknet-19. We propose a new classification model to be used as the base of YOLOv2. Our model builds off of prior work on network design as well as common knowledge in the field. Similar to the VGG models we use mostly 3×3 filters and double the number of channels after every pooling step [17]. Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions [9]. We use batch normalization to stabilize training, speed up convergence, and regularize the model [7].

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [7]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size. Since our model downsamples by a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

https://arxiv.org/abs/1612.08242

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

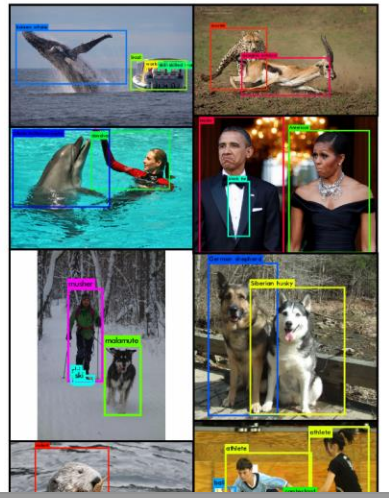
YOLOv2

YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.



The YOLO framework uses a custom network based on the Googlenet architecture [19]. This network is faster than VGG-16, only using 8.52 billion operations for a forward pass. However, it's accuracy is slightly worse than VGG-16. For single-crop, top-5 accuracy at 224×224, YOLO's custom model gets 88.0% ImageNet compared to 90.0% for VGG-16.

Darknet-19. We propose a new classification model to be used as the base of YOLOv2. Our model builds off of prior work on network design as well as common knowledge in the field. Similar to the VGG models we use mostly 3×3 filters and double the number of channels after every pooling step [17]. Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions [9]. We use batch normalization to stabilize training, speed up convergence, and regularize the model [7].

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [7]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size. Since our model downsamples by a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

https://arxiv.org/abs/1612.08242

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

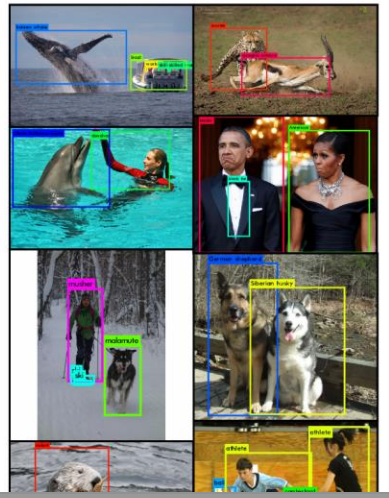
YOLOv2

YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.



The YOLO framework uses a custom network based on the Googlenet architecture [19]. This network is faster than VGG-16, only using 8.52 billion operations for a forward pass. However, it's accuracy is slightly worse than VGG-16. For single-crop, top-5 accuracy at 224×224, YOLO's custom model gets 88.0% ImageNet compared to 90.0% for VGG-16.

Darknet-19. We propose a new classification model to be used as the base of YOLOv2. Our model builds off of prior work on network design as well as common knowledge in the field. Similar to the VGG models we use mostly 3×3 filters and double the number of channels after every pooling step [17]. Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions [9]. We use batch normalization to stabilize training, speed up convergence, and regularize the model [7].

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [7]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size. Since our model downsamples by a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

https://arxiv.org/abs/1612.08242

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

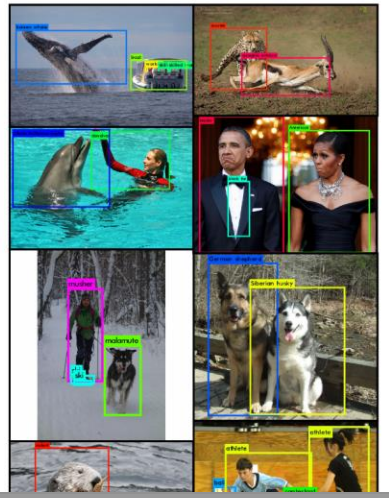
YOLOv2

YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.



The YOLO framework uses a custom network based on the Googlenet architecture [19]. This network is faster than VGG-16, only using 8.52 billion operations for a forward pass. However, it's accuracy is slightly worse than VGG-16. For single-crop, top-5 accuracy at 224×224, YOLO's custom model gets 88.0% ImageNet compared to 90.0% for VGG-16.

Darknet-19. We propose a new classification model to be used as the base of YOLOv2. Our model builds off of prior work on network design as well as common knowledge in the field. Similar to the VGG models we use mostly 3×3 filters and double the number of channels after every pooling step [17]. Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions [9]. We use batch normalization to stabilize training, speed up convergence, and regularize the model [7].

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [7]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size. Since our model downsamples by a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

https://arxiv.org/abs/1612.08242

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

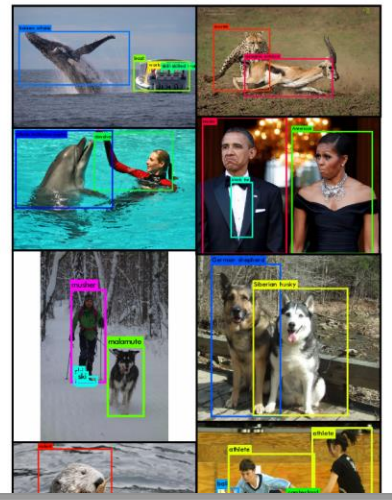
YOLOv2

YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.



The YOLO framework uses a custom network based on the Googlenet architecture [19]. This network is faster than VGG-16, only using 8.52 billion operations for a forward pass. However, it's accuracy is slightly worse than VGG-16. For single-crop, top-5 accuracy at 224×224, YOLO's custom model gets 88.0% ImageNet compared to 90.0% for VGG-16.

Darknet-19. We propose a new classification model to be used as the base of YOLOv2. Our model builds off of prior work on network design as well as common knowledge in the field. Similar to the VGG models we use mostly 3×3 filters and double the number of channels after every pooling step [17]. Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions [9]. We use batch normalization to stabilize training, speed up convergence, and regularize the model [7].

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [7]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size. Since our model downsamples by a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

- Batch normalization
- Darknet-19 backbone
- Anchor boxes
- Multiscale training
- Finner-grained features

https://arxiv.org/abs/1612.08242

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

YOLOv2

YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.

We remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes. First we eliminate one pooling layer to make the output of the network's convolutional layers higher resolution. We also shrink the network to operate on 416 input images instead of 448×448. We do this because we want an odd number of locations in our feature map so there is a single center cell. Objects, especially large objects, tend to occupy the center of the image so it's good to have a single location right at the center to predict these objects instead of four locations that are all nearby. YOLO's convolutional layers downsample the image by a factor of 32 so by using an input image of 416 we get an output feature map of 13×13.

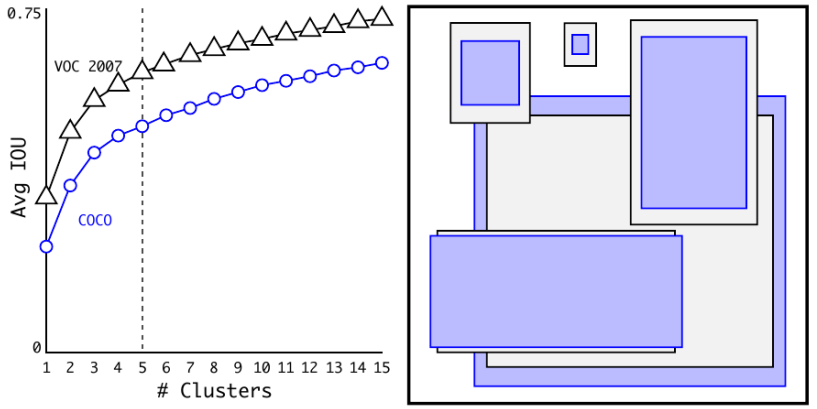


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. 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.

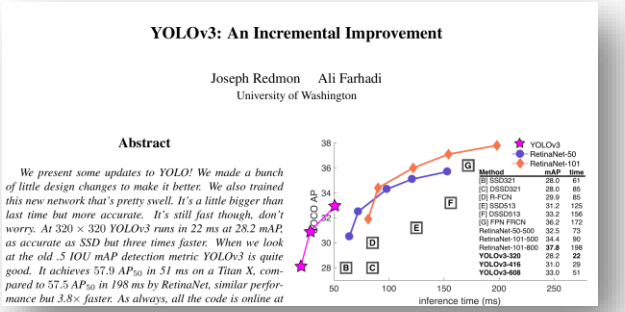
- Batch normalization
- Darknet-19 backbone
- Anchor boxes
- Multiscale training
- Finner-grained features

https://arxiv.org/abs/1612.08242

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

YOLOv3



YOLOv3: An Incremental Improvement

Joseph Redmon Ali Farhadi
University of Washington

Abstract

We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At 320 × 320 YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP_{50} in 51 ms on a Titan X, compared to 57.5 AP_{50} in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at

Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

2.4. Feature Extractor

We use a new network for performing feature extraction. Our new network is a hybrid approach between the network used in YOLOv2, Darknet-19, and that newfangled residual network stuff. Our network uses successive 3×3 and 1×1 convolutional layers but now has some shortcut connections as well and is significantly larger. It has 53 convolutional layers so we call it.... wait for it..... Darknet-53!

2.3. Predictions Across Scales

YOLOv3 predicts boxes at 3 different scales. Our system extracts features from those scales using a similar concept to feature pyramid networks [8]. From our base feature extractor we add several convolutional layers. The last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. In our experiments with COCO [10] we predict 3 boxes at each scale so the tensor is $N \times N \times [3 * (4 + 1 + 80)]$ for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.

https://arxiv.org/abs/1804.02767
https://arxiv.org/abs/2304.00501

YOLOv3

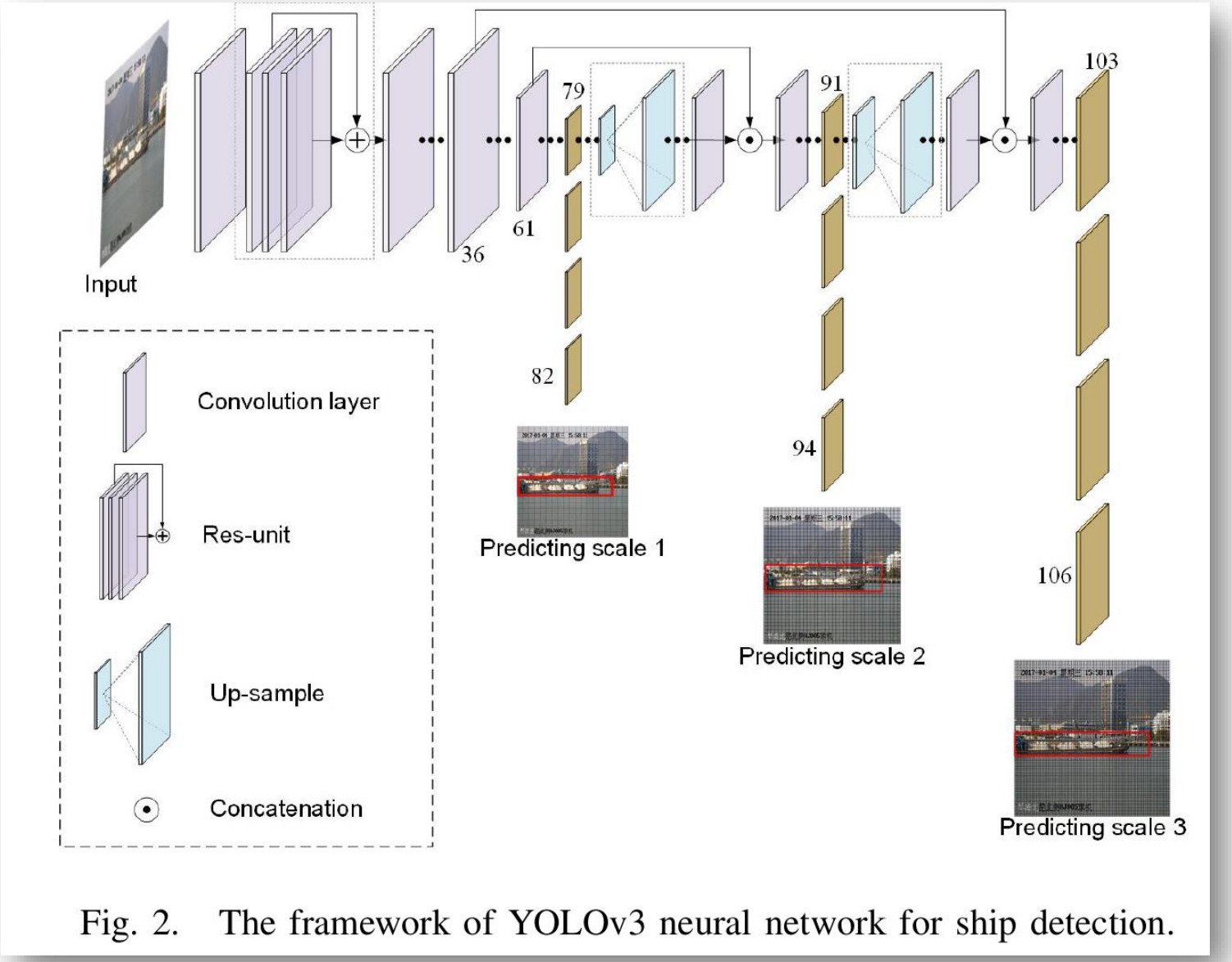


Fig. 2. The framework of YOLOv3 neural network for ship detection.

2.3. Predictions Across Scales

YOLOv3 predicts boxes at 3 different scales. Our system extracts features from those scales using a similar concept to feature pyramid networks [8]. From our base feature extractor we add several convolutional layers. The last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. In our experiments with COCO [10] we predict 3 boxes at each scale so the tensor is $N \times N \times [3 * (4 + 1 + 80)]$ for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.

	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
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

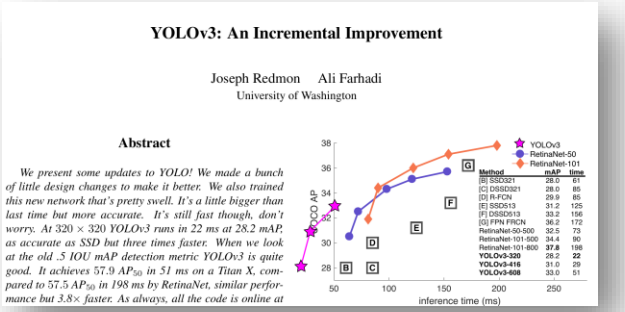
https://arxiv.org/abs/1804.02767
https://arxiv.org/abs/2304.00501

X. Nie, M. Yang and R. W. Liu, "Deep Neural Network-Based Robust Ship Detection Under Different Weather Conditions," *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Auckland, New Zealand, 2019, pp. 47-52, doi: 10.1109/ITSC.2019.8917475.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

YOLOv3



YOLOv3: An Incremental Improvement

Joseph Redmon Ali Farhadi
University of Washington

Abstract

We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At 320 × 320 YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP_{50} in 51 ms on a Titan X, compared to 57.5 AP_{50} in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at

2.4. Feature Extractor

We use a new network for performing feature extraction. Our new network is a hybrid approach between the network used in YOLOv2, Darknet-19, and that newfangled residual network stuff. Our network uses successive 3×3 and 1×1 convolutional layers but now has some shortcut connections as well and is significantly larger. It has 53 convolutional layers so we call it.... wait for it..... Darknet-53!

2.3. Predictions Across Scales

YOLOv3 predicts boxes at 3 different scales. Our system extracts features from those scales using a similar concept to feature pyramid networks [8]. From our base feature extractor we add several convolutional layers. The last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. In our experiments with COCO [10] we predict 3 boxes at each scale so the tensor is $N \times N \times [3 * (4 + 1 + 80)]$ for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.

- Darknet-53
- Residual blocks
- 3 different scales
- Feature pyramid net.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Table 3. I'm seriously just stealing all these tables from [9] they take soooo long to make from scratch. Ok, YOLOv3 is doing alright. Keep in mind that RetinaNet has like 3.8× longer to process an image. YOLOv3 is much better than SSD variants and comparable to state-of-the-art models on the AP_{50} metric.

	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
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

https://arxiv.org/abs/1804.02767
https://arxiv.org/abs/2304.00501

BOXES

- **YOLOv1**
  - 98 boxes
  - 7x7 cells, 2 boxes per cell
- **YOLOv2**
  - 845 boxes
  - 13x13 cells, 5 anchor boxes
- **YOLOv3**
  - 10647 boxes
  - 13x13, 26x26, 52x52 cells, 3 anchor boxes

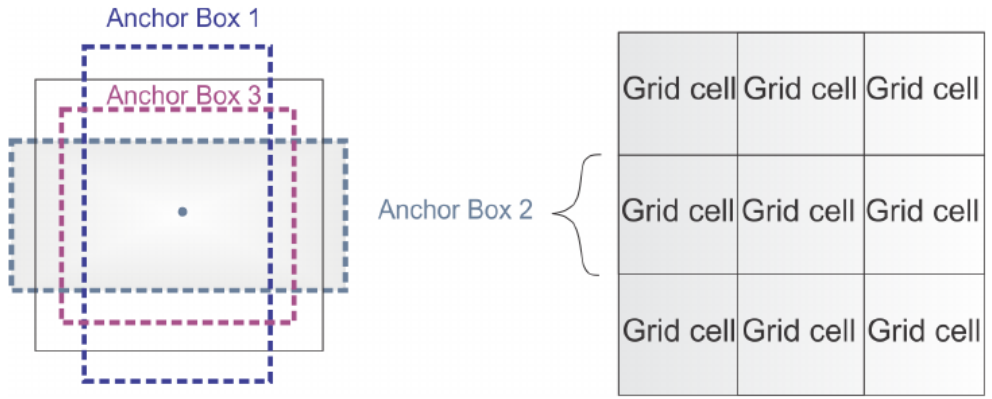


Figure 6: Anchor boxes. YOLOv2 defines multiple anchor boxes for each grid cell.

```yaml
# parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [116,90, 156,198, 373,326]  # P5/32
  - [30,61, 62,45, 59,119]  # P4/16
  - [10,13, 16,30, 33,23]  # P3/8

# YOLOv5 backbone
```

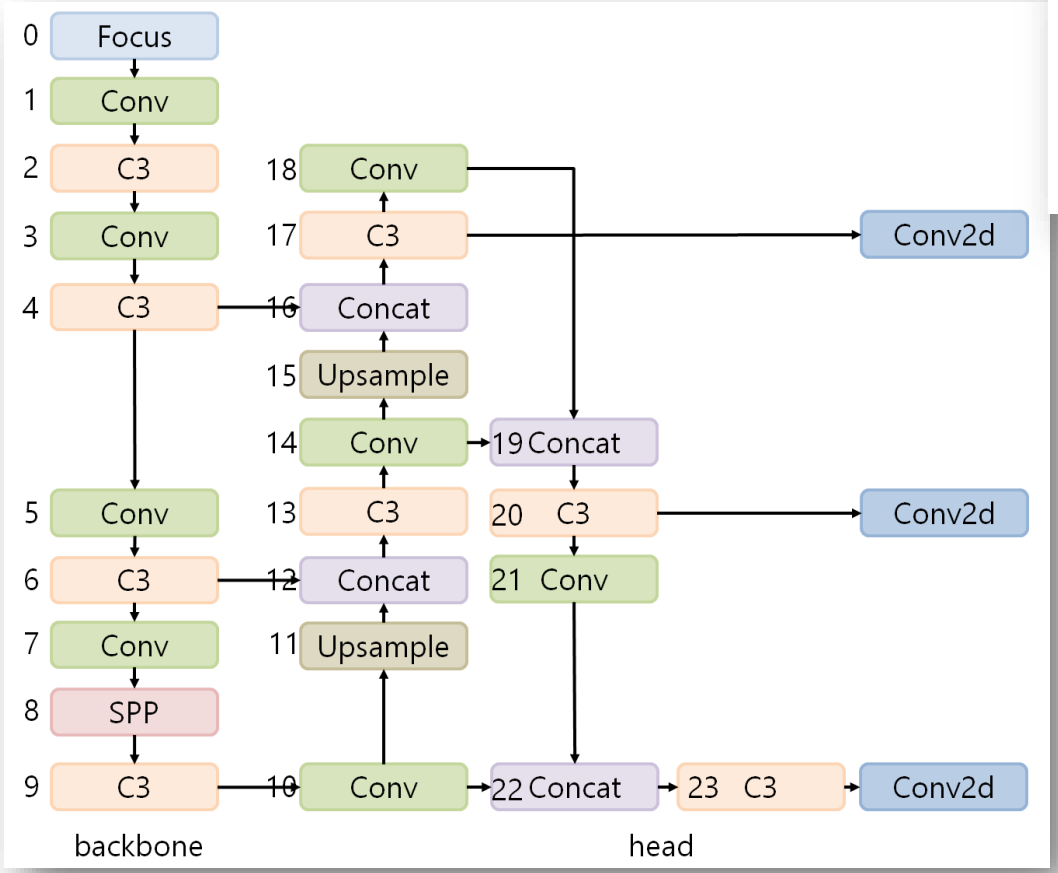
YOLOv5 anchor box configuration

https://blog.roboflow.com/what-is-an-anchor-box/
https://community.ultralytics.com/t/anchor-boxes-interpretation/146/5

https://arxiv.org/abs/1804.02767
https://arxiv.org/abs/2304.00501

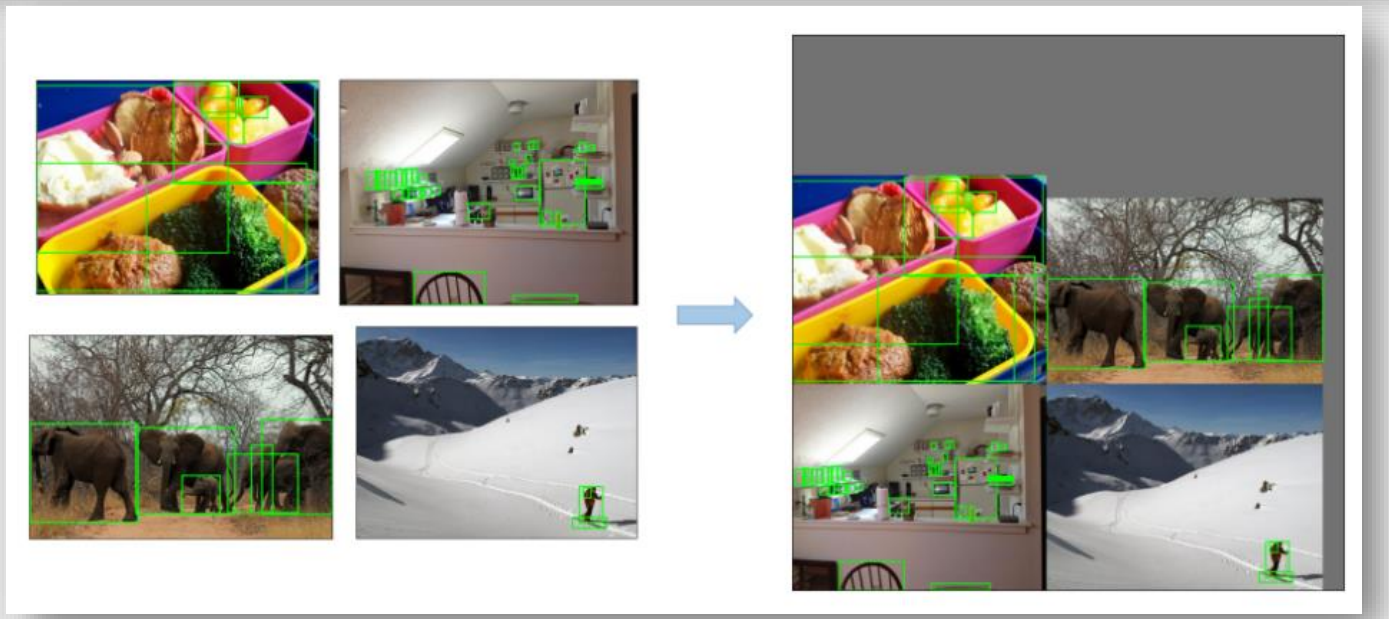
YOLOv5

https://github.com/ultralytics/yolov5/issues/280
https://docs.ultralytics.com/yolov5/tutorials/architecture_description/#1-model-structure
https://github.com/ultralytics/yolov5/issues/6998



YOLOv5 employs various data augmentation techniques to improve the model's ability to generalize and reduce overfitting. These techniques include:

- **Mosaic Augmentation**: An image processing technique that combines four training images into one in ways that encourage object detection models to better handle various object scales and translations.



https://github.com/ultralytics/yolov5/issues/280#issuecomment-1000948444

https://arxiv.org/abs/1804.02767
https://arxiv.org/abs/2304.00501

Table 4: Summary of YOLO architectures. The metric reported for YOLO and YOLOv2 were on VOC2007, while the rest are reported on COCO2017. The NAS-YOLO model reported has 16-bit precision.

Version	Date	Anchor	Framework	Backbone	AP (%)
YOLO	2015	No	Darknet	Darknet24	*63.4*
YOLOv2	2016	Yes	Darknet	Darknet24	*63.4*
YOLOv3	2018	Yes	Darknet	Darknet53	36.2
YOLOv4	2020	Yes	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Yes	Pytorch	YOLOv5CSPDarknet	55.8
PP-YOLO	2020	Yes	PaddlePaddle	ResNet50-vd	45.9
Scaled-YOLOv4	2021	Yes	Pytorch	CSPDarknet	56.0
PP-YOLOv2	2021	Yes	PaddlePaddle	ResNet101-vd	50.3
YOLOR	2021	Yes	Pytorch	CSPDarknet	55.4
YOLOX	2021	No	Pytorch	YOLOXCSPDarknet	51.2
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	54.7
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	YOLOv7Backbone	56.8
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	50.0
YOLOv8	2023	No	Pytorch	YOLOv8CSPDarknet	53.9
YOLO-NAS	2023	No	Pytorch	NAS	52.2

https://arxiv.org/abs/2304.00501
https://arxiv.org/abs/2305.17786
https://arxiv.org/abs/1506.02640

SSD: Single Shot MultiBox Detector

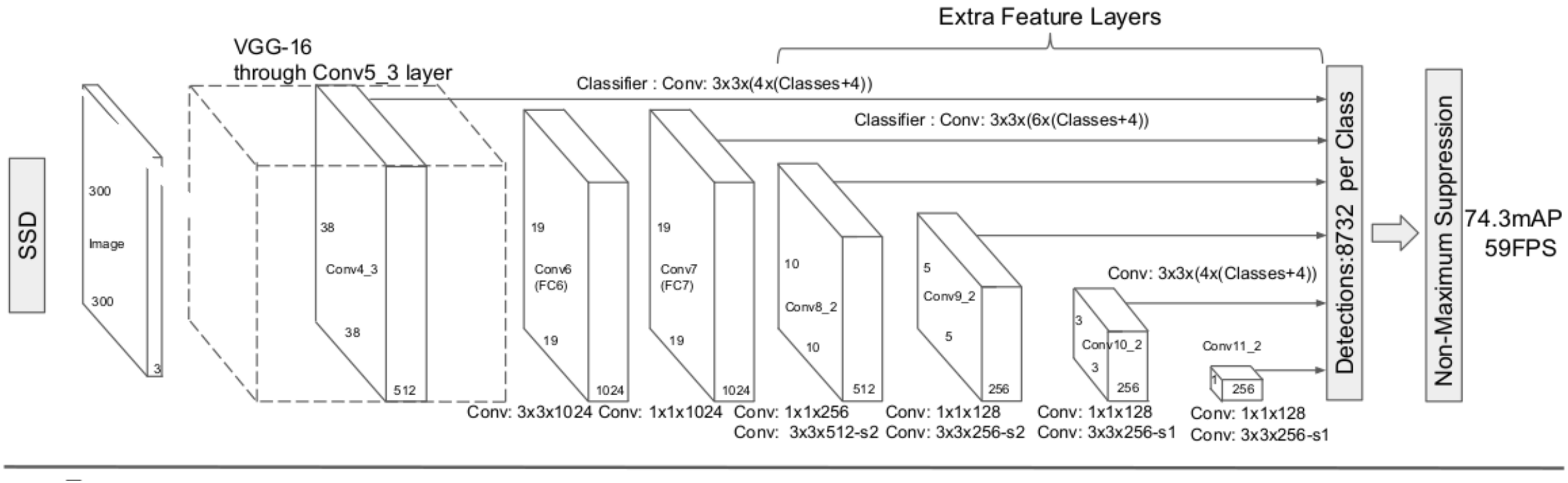
Wei Liu[1], Dragomir Anguelov[2], Dumitru Erhan[3], Christian Szegedy[3],
Scott Reed[4], Cheng-Yang Fu[1], Alexander C. Berg[1]

[1]UNC Chapel Hill [2]Zoox Inc. [3]Google Inc. [4]University of Michigan, Ann-Arbor
[1]wliu@cs.unc.edu, [2]drago@zoox.com, [3]{dumitru,szegedy}@google.com,
[4]reedscot@umich.edu, [1]{cyfu,aberg}@cs.unc.edu

- We introduce SSD, a single-shot detector for multiple categories that is faster than the previous state-of-the-art for single shot detectors (YOLO), and significantly more accurate, in fact as accurate as slower techniques that perform explicit region proposals and pooling (including Faster R-CNN).
- The core of SSD is predicting category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps.
- To achieve high detection accuracy we produce predictions of different scales from feature maps of different scales, and explicitly separate predictions by aspect ratio.
- These design features lead to simple end-to-end training and high accuracy, even on low resolution input images, further improving the speed vs accuracy trade-off.
- Experiments include timing and accuracy analysis on models with varying input size evaluated on PASCAL VOC, COCO, and ILSVRC and are compared to a range of recent state-of-the-art approaches.

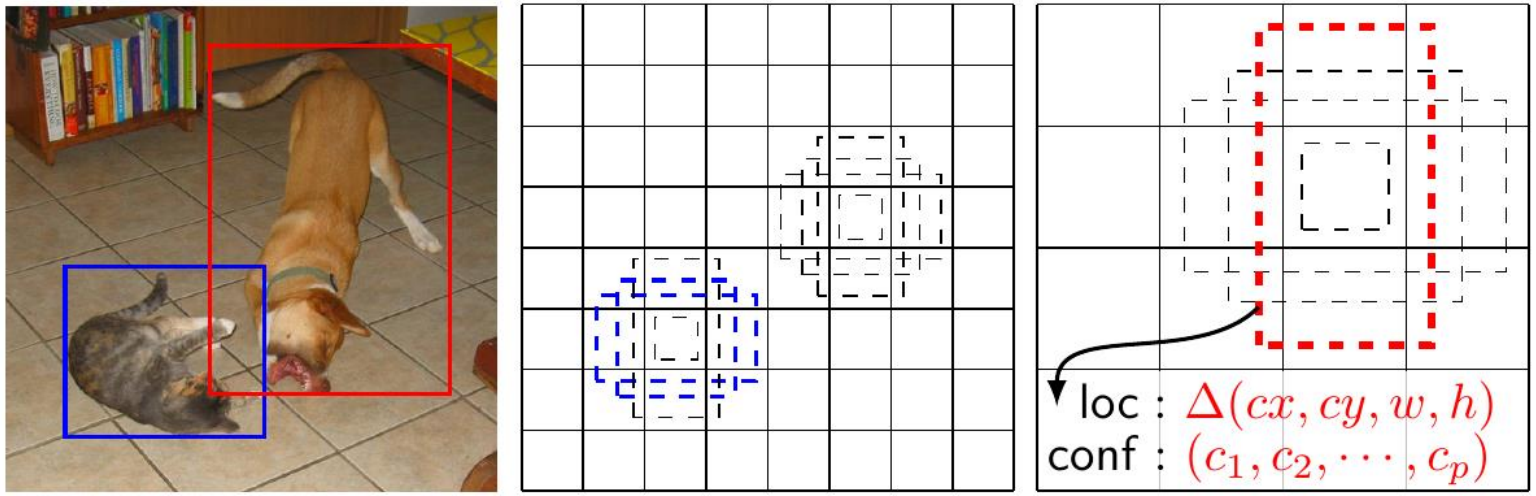
https://arxiv.org/abs/1512.02325

SSD



https://arxiv.org/abs/1512.02325

Anchor Boxes



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

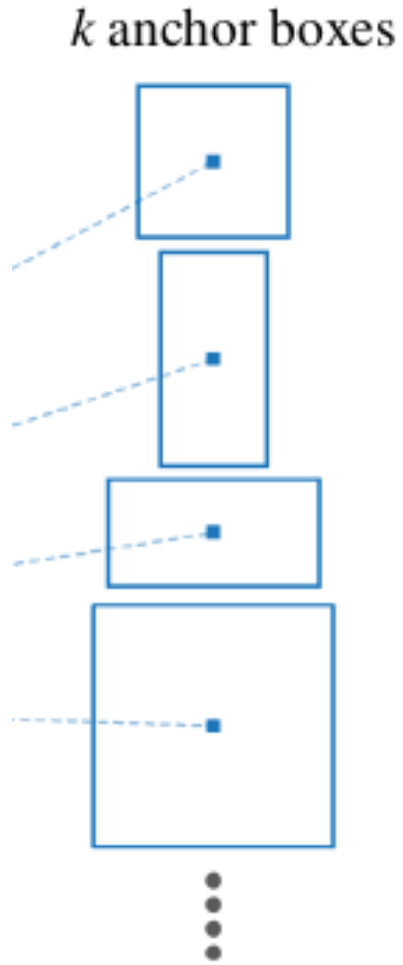


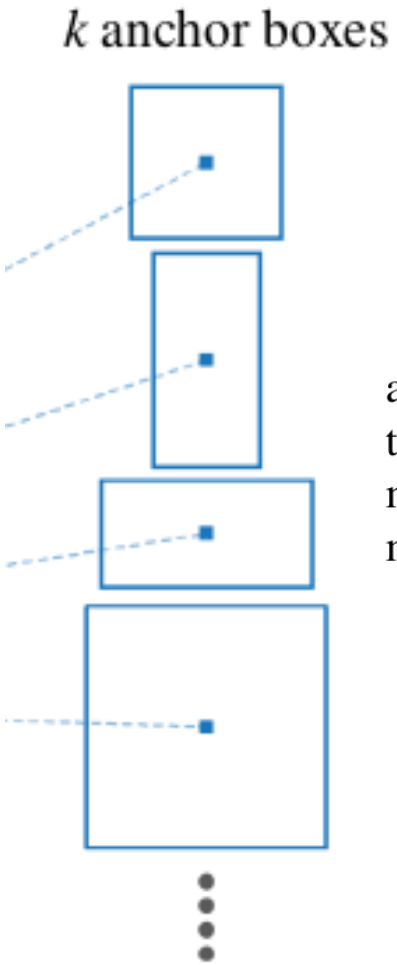
Fig. 1: **SSD framework.** (a) SSD only needs an input image and ground truth boxes for each object during training. In a convolutional fashion, we evaluate a small set (e.g. 4) of default boxes of different aspect ratios at each location in several feature maps with different scales (e.g. 8×8 and 4×4 in (b) and (c)). For each default box, we predict both the shape offsets and the confidences for all object categories ($(c_1, c_2, \cdots, c_p)$). At training time, we first match these default boxes to the ground truth boxes. For example, we have matched two default boxes with the cat and one with the dog, which are treated as positives and the rest as negatives. The model loss is a weighted sum between localization loss (e.g. Smooth L1 [6]) and confidence loss (e.g. Softmax).

https://arxiv.org/abs/1512.02325

https://arxiv.org/abs/1506.01497

Anchor Boxes

Face
Pedestrian
Car
...
..
..

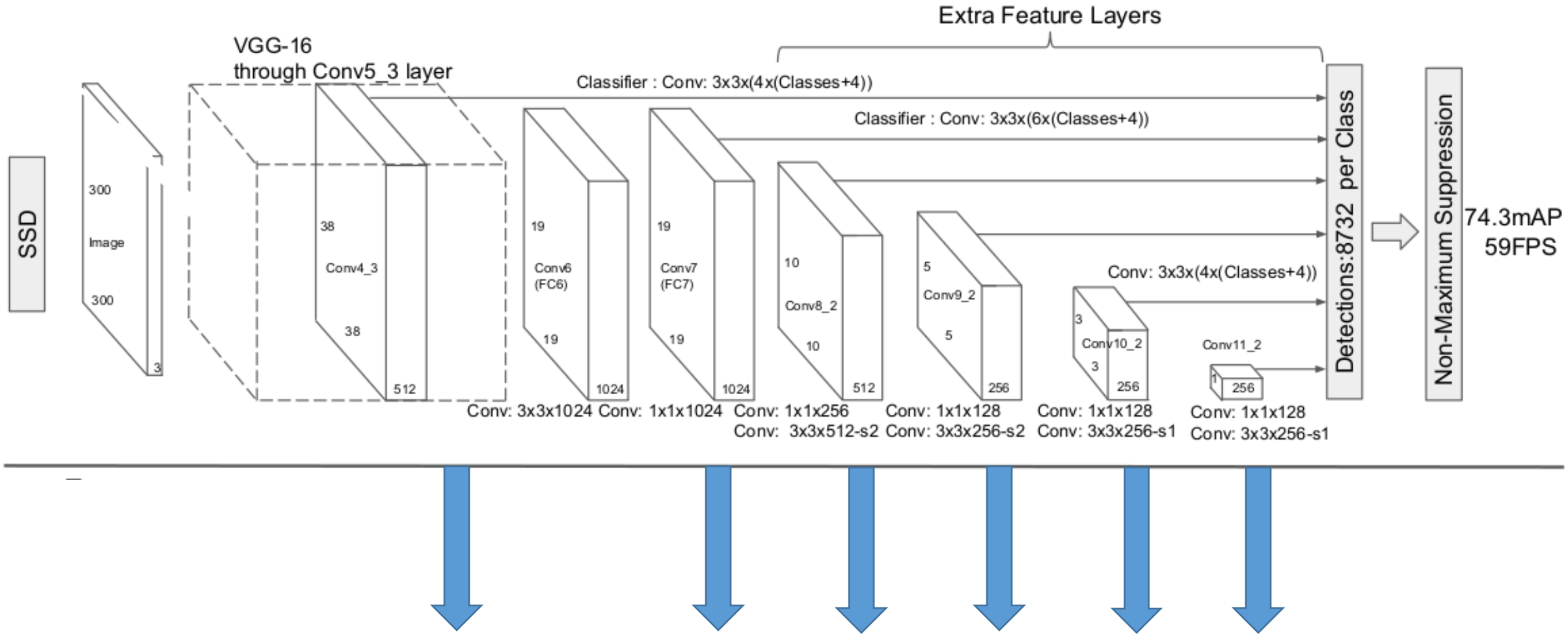


an illustration of default boxes, please refer to Fig. 1. Our default boxes are similar to the *anchor boxes* used in Faster R-CNN [2], however we apply them to several feature maps of different resolutions. Allowing different default box shapes in several feature maps let us efficiently discretize the space of possible output box shapes.

Sizes can be obtained from dataset

https://arxiv.org/abs/1506.01497
https://arxiv.org/abs/1512.02325

SSD



cnn layers make predictions, each with different anchor boxes

https://arxiv.org/abs/1512.02325

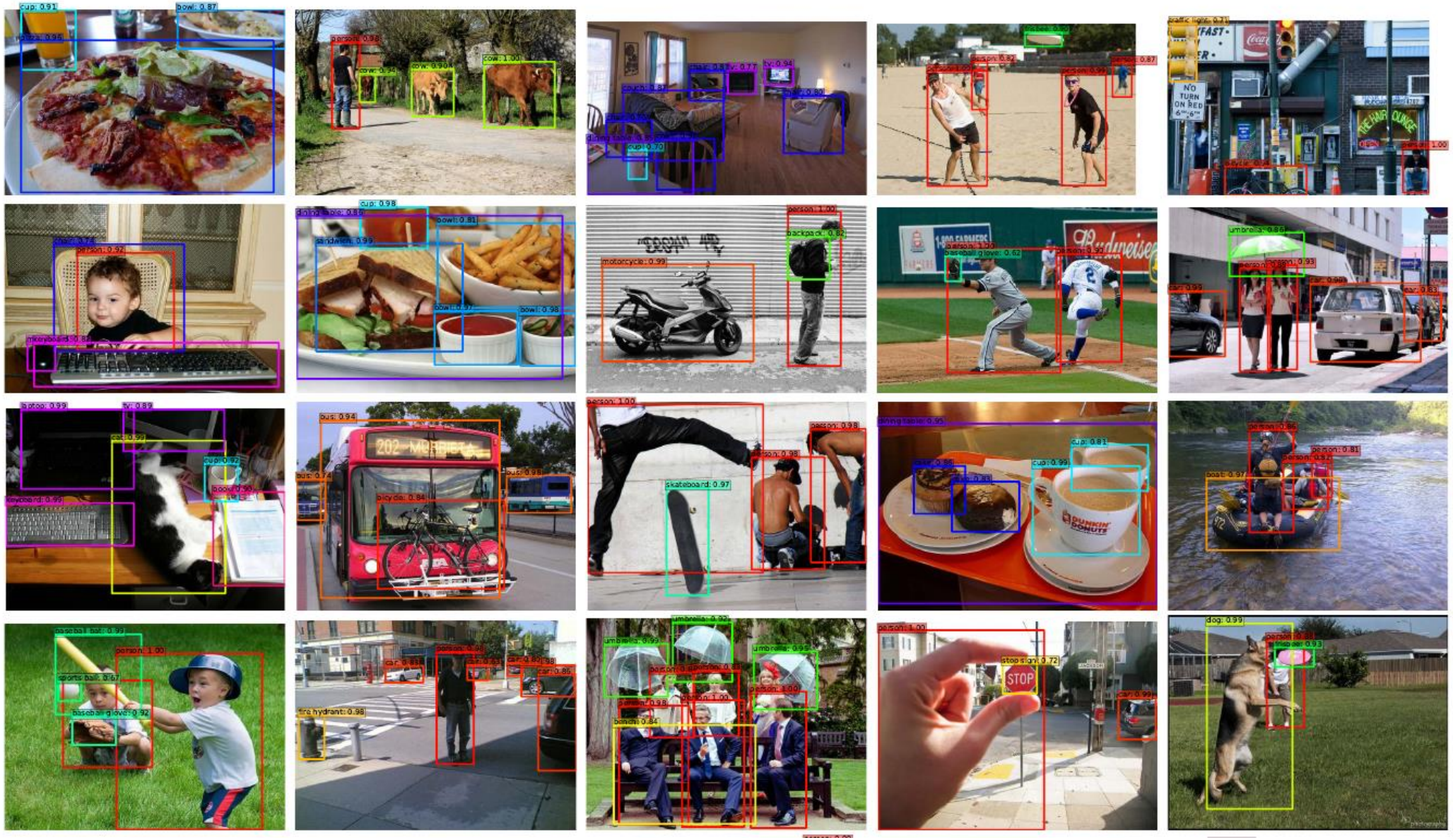
SSD

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Table 7: **Results on Pascal VOC2007 `test`.** SSD300 is the only real-time detection method that can achieve above 70% mAP. By using a larger input image, SSD512 outperforms all methods on accuracy while maintaining a close to real-time speed.

https://arxiv.org/abs/1512.02325

SSD



https://arxiv.org/abs/1512.02325

PyTorch Example

Docs > Models and pre-trained weights

Here is an example of how to use the pre-trained object detection models:

```
from torchvision.io.image import read_image
from torchvision.models.detection import fasterrcnn_resnet50_fpn_v2,
FasterRCNN_ResNet50_FPN_V2_Weights
from torchvision.utils import draw_bounding_boxes
from torchvision.transforms.functional import to_pil_image

img = read_image("test/assets/encode_jpeg/grace_hopper_517x606.jpg")

# Step 1: Initialize model with the best available weights
weights = FasterRCNN_ResNet50_FPN_V2_Weights.DEFAULT
model = fasterrcnn_resnet50_fpn_v2(weights=weights, box_score_thresh=0.9)
model.eval()

# Step 2: Initialize the inference transforms
preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms
batch = [preprocess(img)]

# Step 4: Use the model and visualize the prediction
prediction = model(batch)[0]
labels = [weights.meta["categories"][i] for i in prediction["labels"]]
box = draw_bounding_boxes(img, boxes=prediction["boxes"],
                          labels=labels,
                          colors="red",
                          width=4, font_size=30)
im = to_pil_image(box.detach())
im.show()
```

The classes of the pre-trained model outputs can be found at `weights.meta["categories"]`. For details on how to plot the bounding boxes of the models, you may refer to Instance segmentation models.

Table of all available Object detection weights

Box MAPs are reported on COCO val2017:

Weight	Box MAP	Params	Recipe
FCOS_ResNet50_FPN_Weights.COCO_V1	39.2	32.3M	link
FasterRCNN_MobileNet_V3_Large_320_FPN_Weights.COCO_V1	22.8	19.4M	link
FasterRCNN_MobileNet_V3_Large_FPN_Weights.COCO_V1	32.8	19.4M	link
FasterRCNN_ResNet50_FPN_V2_Weights.COCO_V1	46.7	43.7M	link
FasterRCNN_ResNet50_FPN_Weights.COCO_V1	37	41.8M	link
RetinaNet_ResNet50_FPN_V2_Weights.COCO_V1	41.5	38.2M	link
RetinaNet_ResNet50_FPN_Weights.COCO_V1	36.4	34.0M	link
SSD300_VGG16_Weights.COCO_V1	25.1	35.6M	link
SSDLite320_MobileNet_V3_Large_Weights.COCO_V1	21.3	3.4M	link

https://pytorch.org/vision/stable/models.html#object-detection-instance-segmentation-and-person-keypoint-detection

FPN/RETINA NET

Feature Pyramid Networks for Object Detection

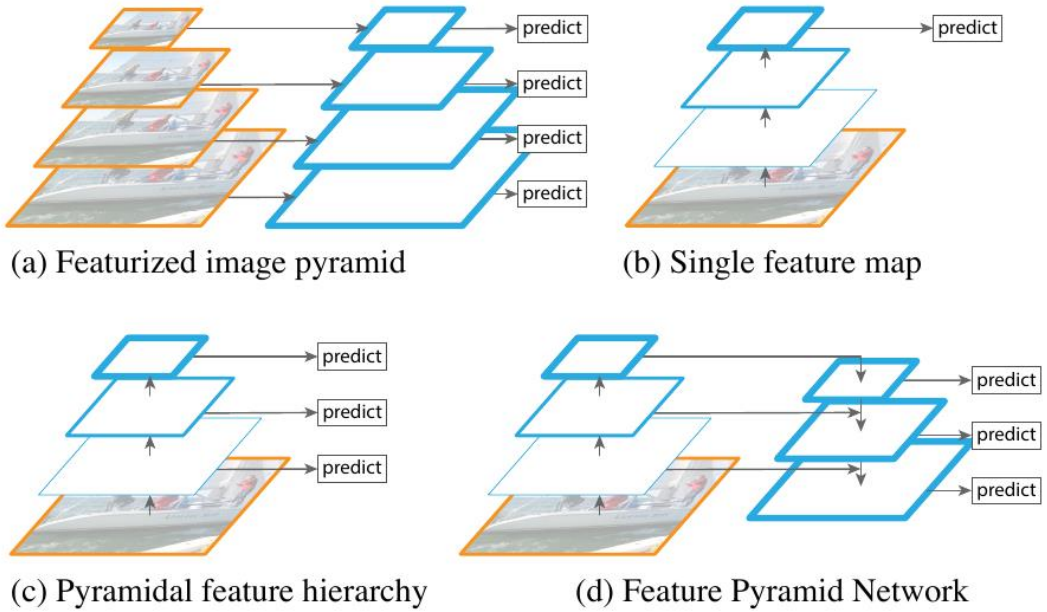
Tsung-Yi Lin[1,2], Piotr Dollár[1], Ross Girshick[1],
Kaiming He[1], Bharath Hariharan[1], and Serge Belongie[2]

[1]Facebook AI Research (FAIR)
[2]Cornell University and Cornell Tech

The goal of this paper is to naturally leverage the pyramidal shape of a ConvNet's feature hierarchy while creating a feature pyramid that has strong semantics at all scales. To achieve this goal, we rely on an architecture that combines low-resolution, semantically strong features with high-resolution, semantically weak features via a top-down pathway and lateral connections (Fig. 1(d)). The result is a feature pyramid that has rich semantics at all levels and is built quickly from a single input image scale. In other words, we show how to create in-network feature pyramids that can be used to replace featurized image pyramids without sacrificing representational power, speed, or memory.

4. Applications

Our method is a generic solution for building feature pyramids inside deep ConvNets. In the following we adopt our method in RPN [29] for bounding box proposal generation and in Fast R-CNN [11] for object detection. To demonstrate the simplicity and effectiveness of our method, we make minimal modifications to the original systems of [29, 11] when adapting them to our feature pyramid.



(a) Featurized image pyramid (b) Single feature map

(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Figure 1. (a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow. (b) Recent detection systems have opted to use only single scale features for faster detection. (c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid. (d) Our proposed Feature Pyramid Network (FPN) is fast like (b) and (c), but more accurate. In this figure, feature maps are indicate by blue outlines and thicker outlines denote semantically stronger features.

https://arxiv.org/abs/1708.02002
https://arxiv.org/abs/1612.03144

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

FPN/RETINA NET

Focal Loss for Dense Object Detection

Tsung-Yi Lin Priya Goyal Ross Girshick Kaiming He Piotr Dollár
Facebook AI Research (FAIR)

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2

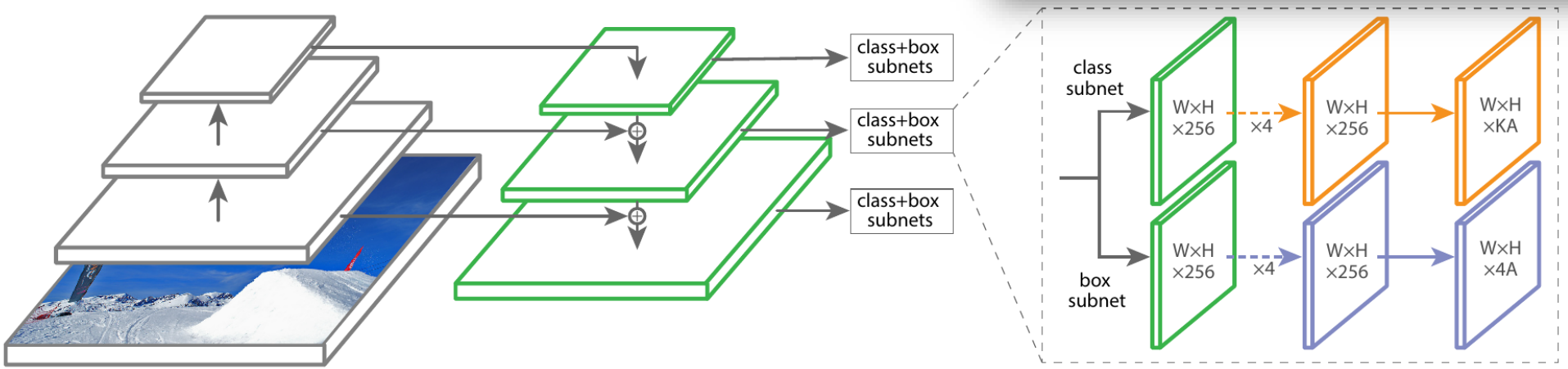


Figure 3. The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). The network design is intentionally simple, which enables this work to focus on a novel focal loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN [20] while running at faster speeds.

https://arxiv.org/abs/1708.02002
https://arxiv.org/abs/1612.03144