https://github.com/kaopanboonyuen/deep_learning_for_vision

Deep Learning for Computer Vision:
The Abridged Guide

Kao Panboonyuen

teerapong.pa@chula.ac.th

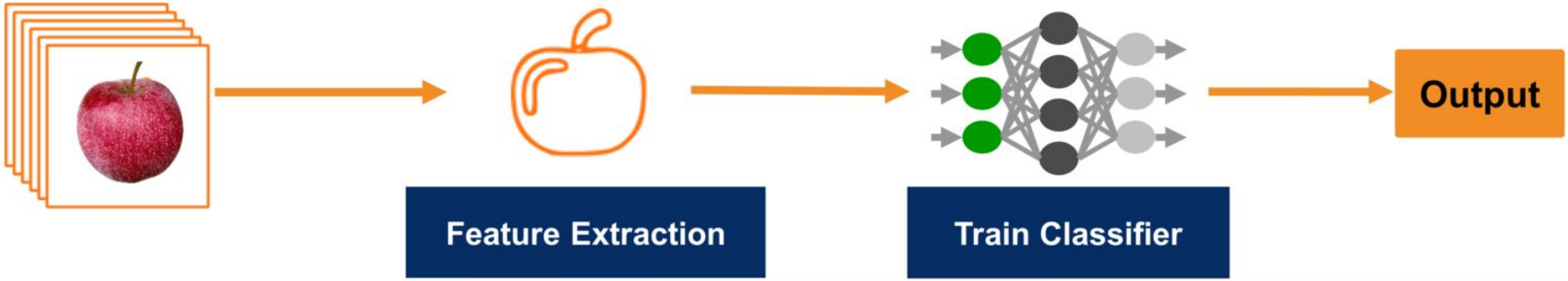
Outlines

- Introduction to Computer Vision
- Code and Demo: Object Detection
- Code and Demo: Instance Segmentation

Reference

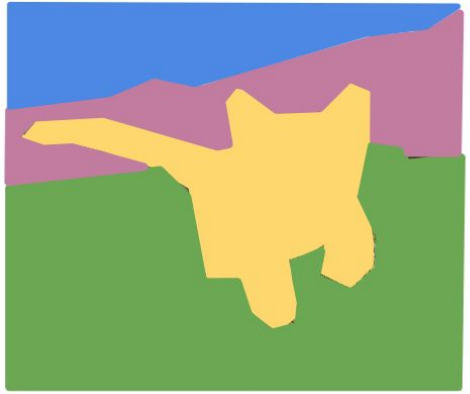
- https://blog.research.google/
- https://medium.com/kaggle-blog
- https://pnut2357.github.io/cv_intro/
- https://sif-dlv.github.io/
- https://www.run.ai/guides/deep-learning-for-computer-vision
- https://www.oreilly.com/library/view/practical-machine-learning/9781098102357/ch04.html
- https://docs.ultralytics.com/

Classic Machine Learning



Deep Learning

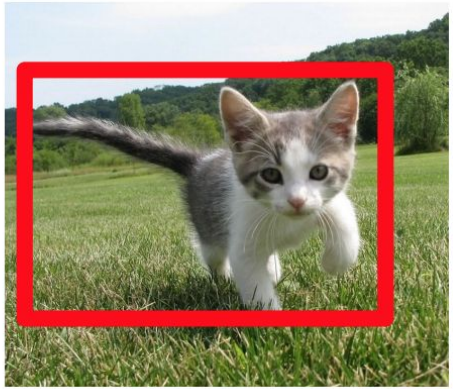
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

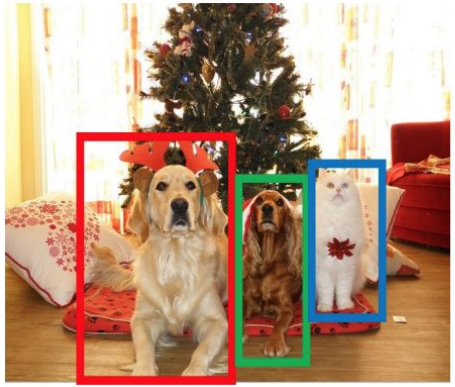
Classification + Localization



CAT

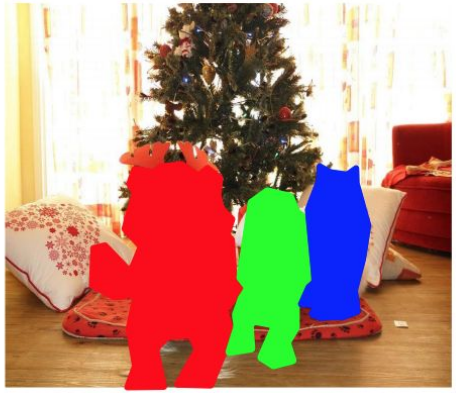
Single Object

Object Detection



DOG, DOG, CAT

Instance Segmentation



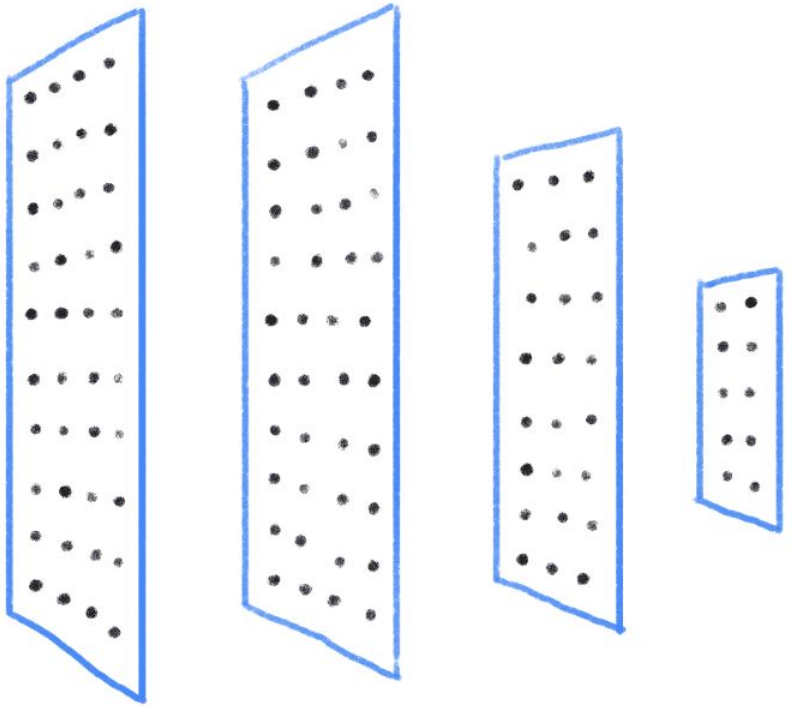
DOG, DOG, CAT

Multiple Object

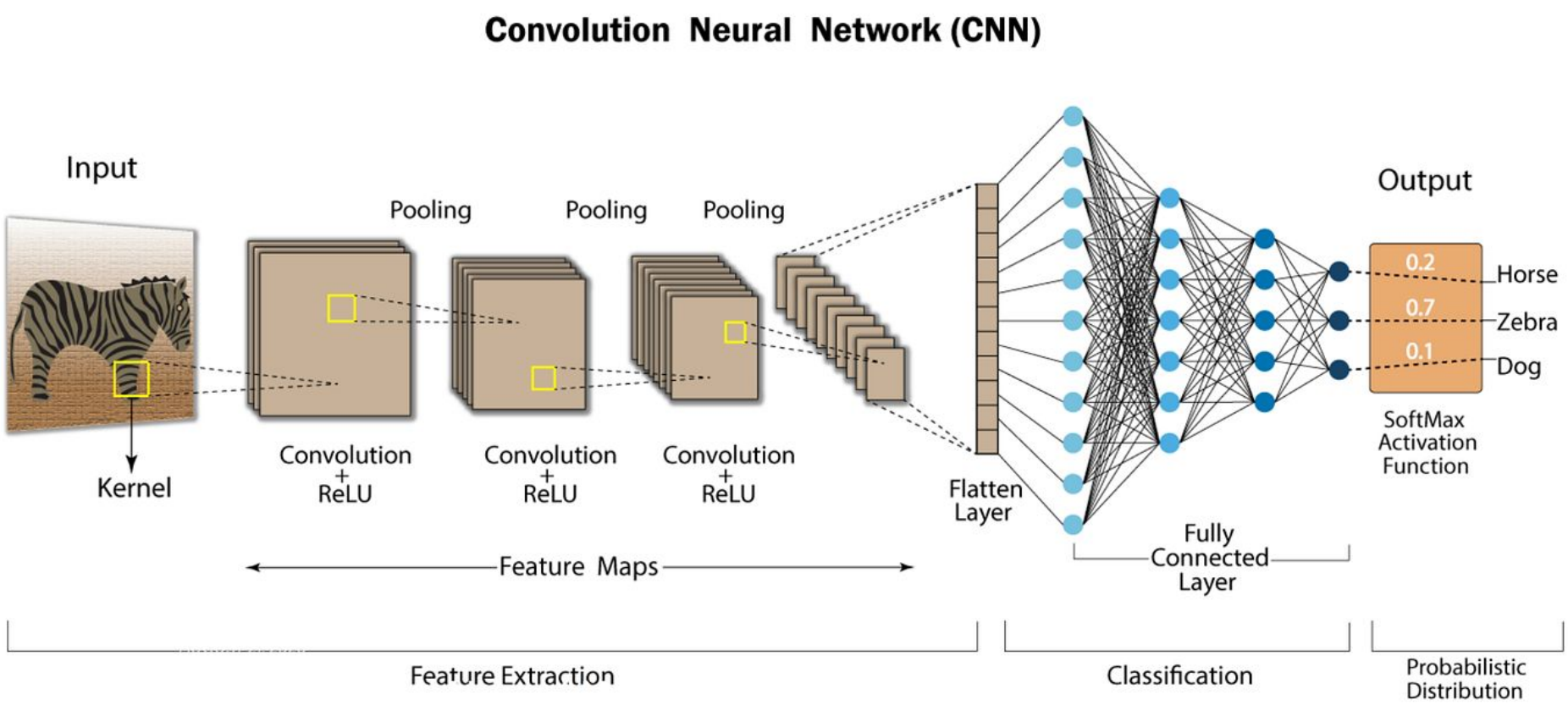
CAT

(LABELED PHOTOS)

DOG

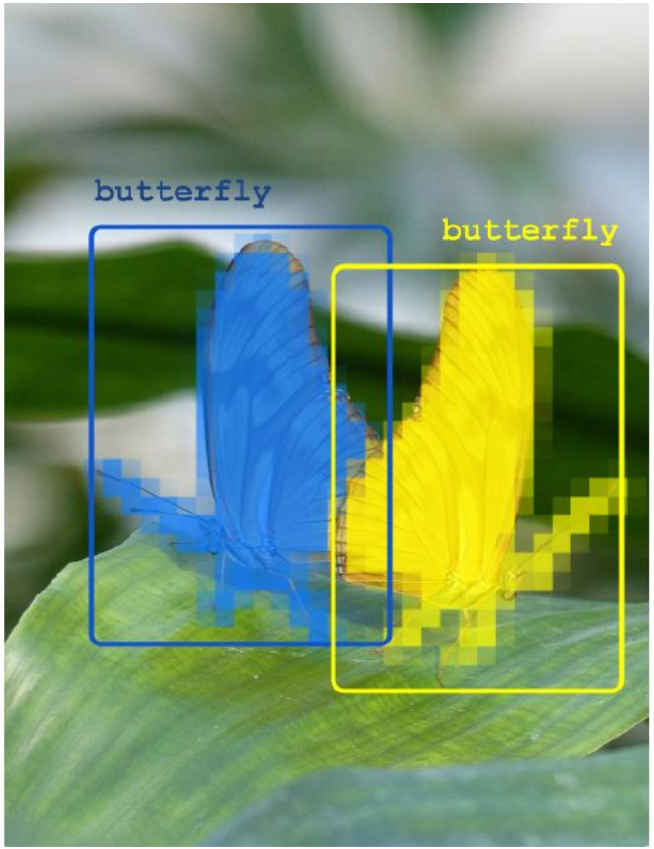


OUTPUT

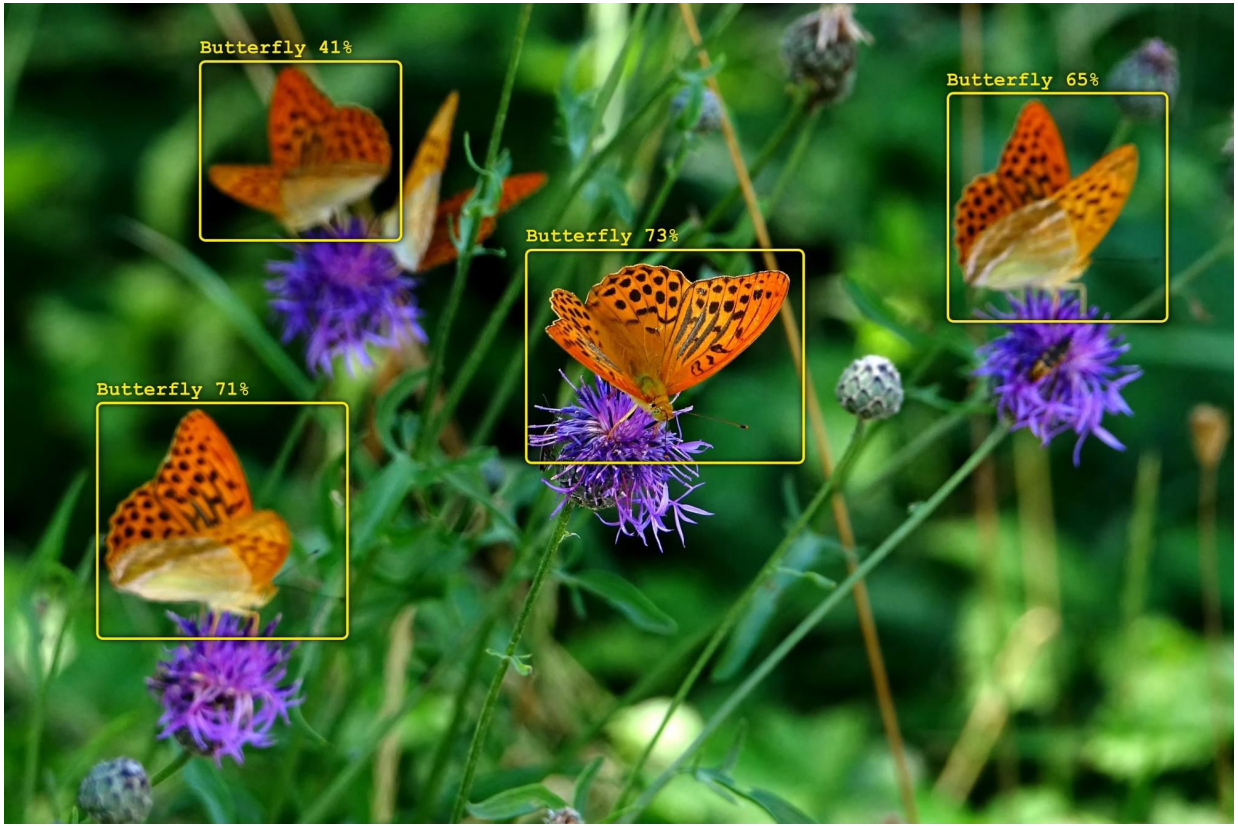

Convolution Neural Network (CNN)
Input
Pooling
Pooling
Pooling
Output
0.2
Horse
0.7
Zebra
0.1
Dog
Kernel
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Flatten Layer
SoftMax Activation Function
Feature Maps
Fully Connected Layer
Feature Extraction
Classification
Probabilistic Distribution

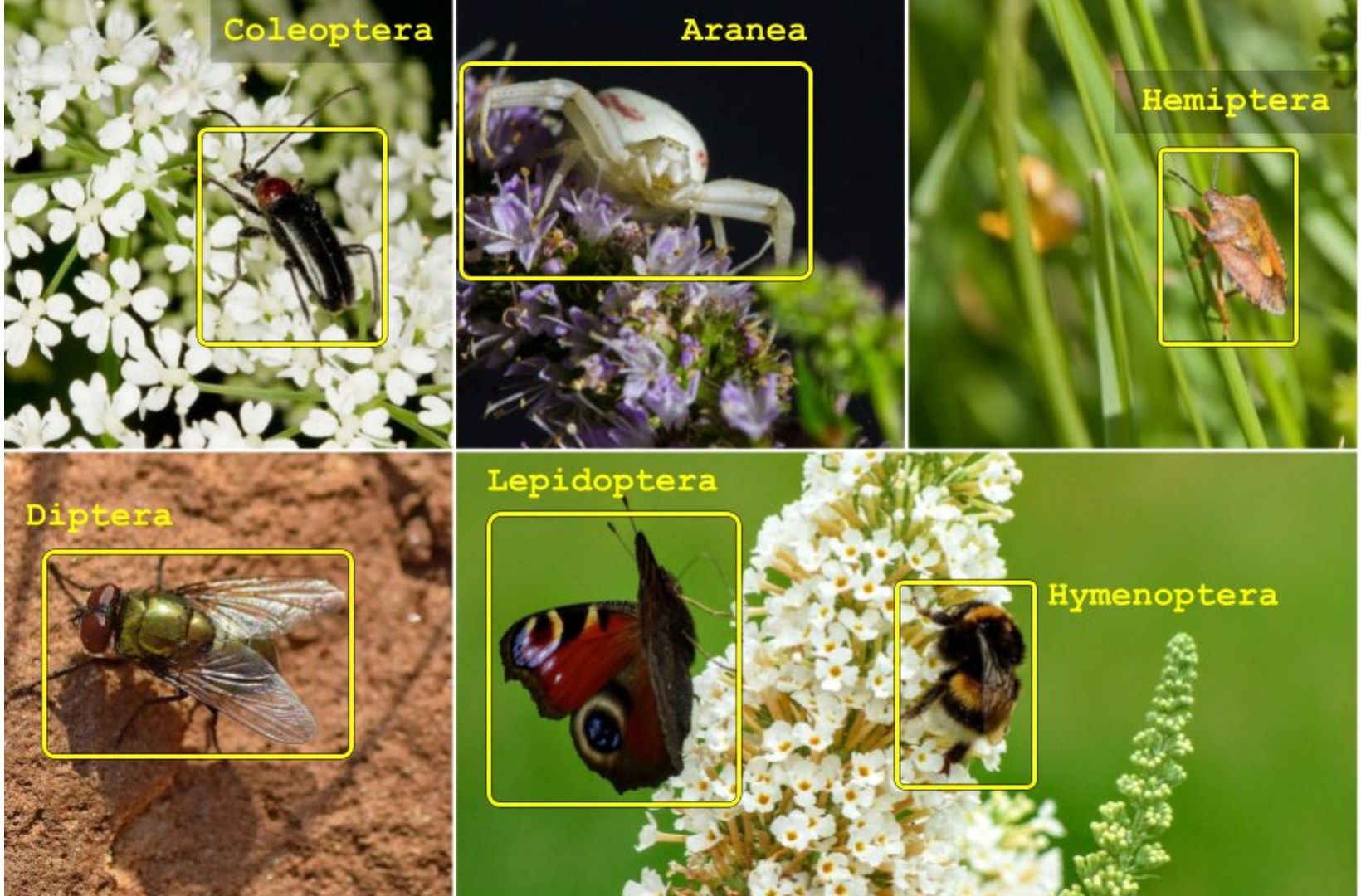
Object Detection


Butterfly 73%


butterfly
butterfly

Object Detection

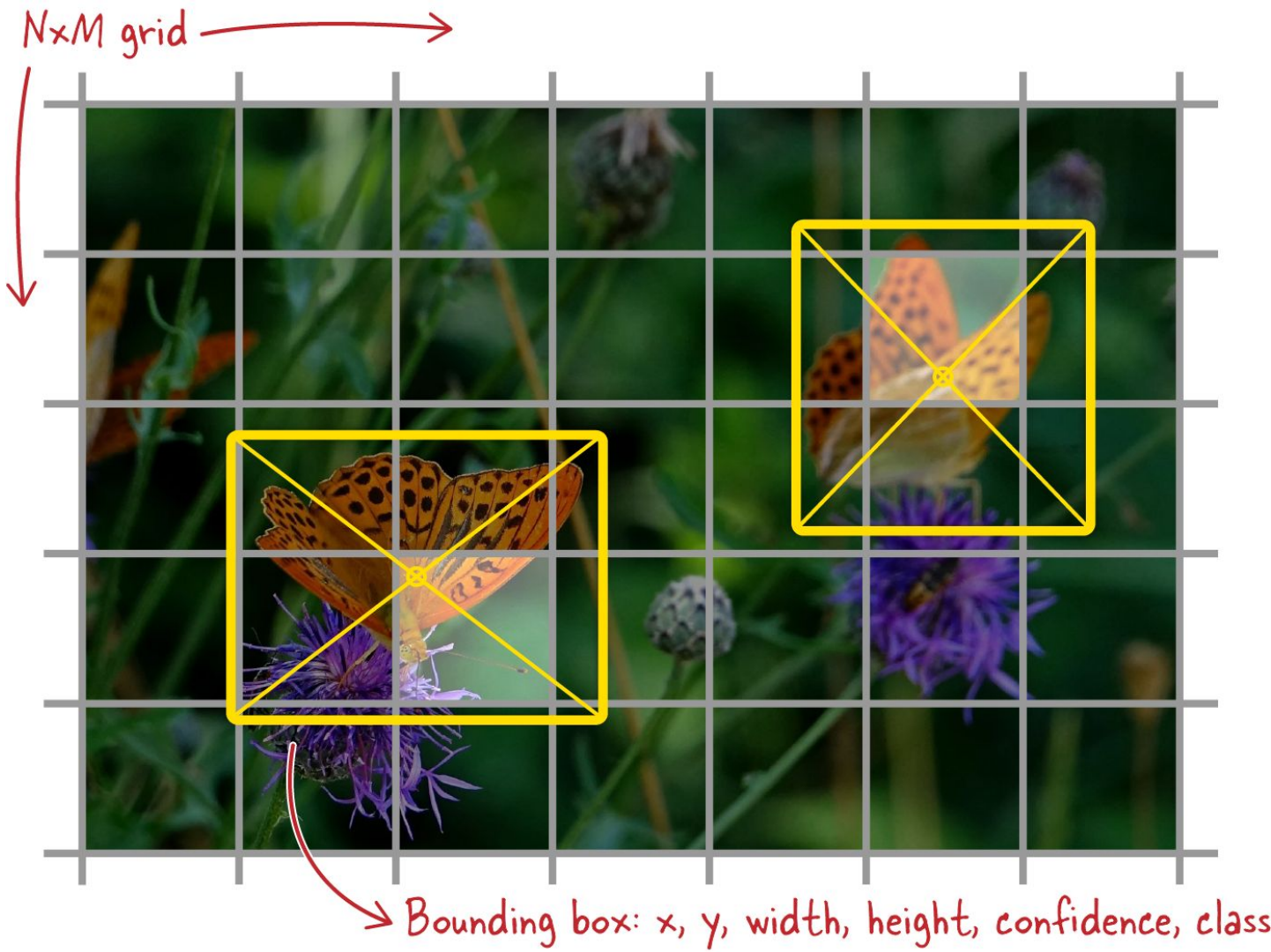



Coleoptera
Aranea
Hemiptera
Diptera
Lepidoptera
Hymenoptera

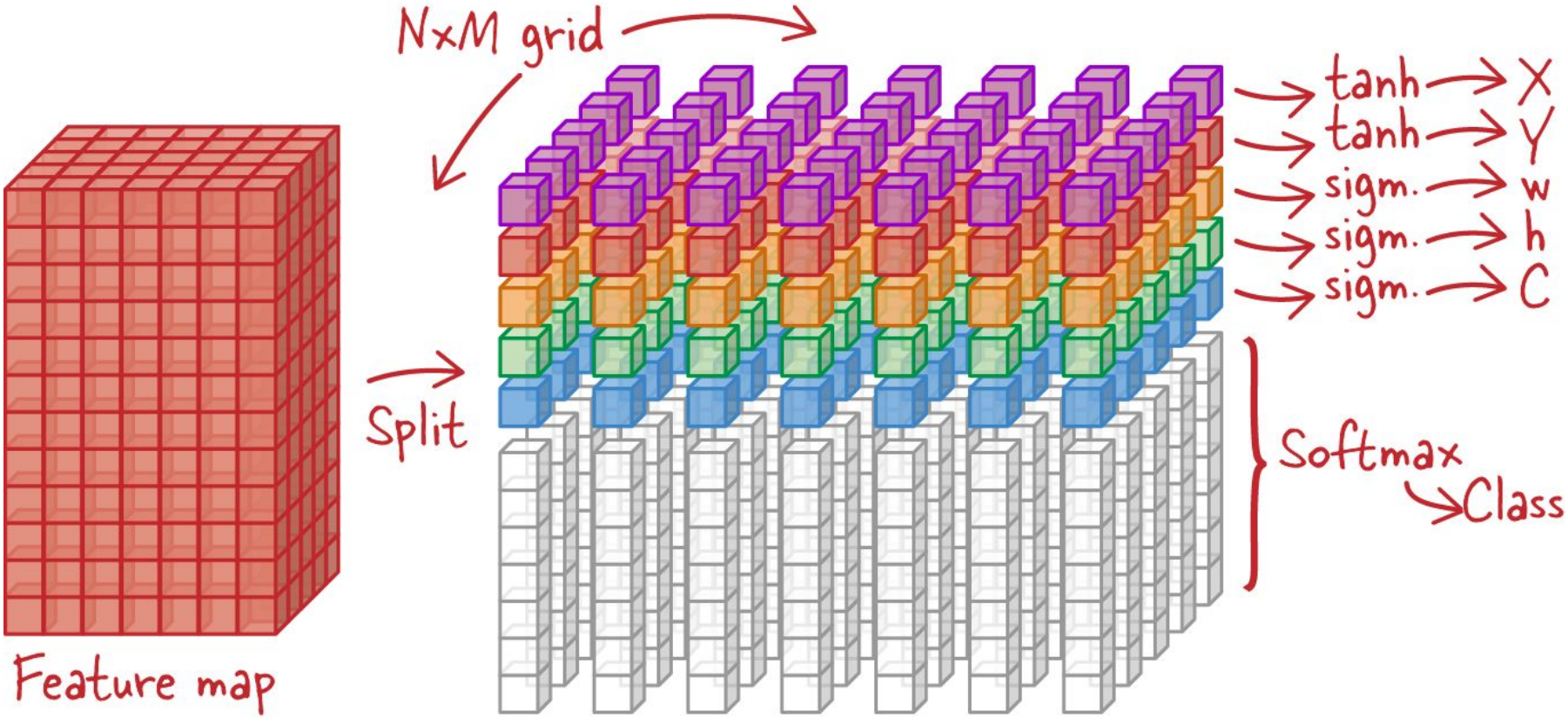
YOLO

YOLO (you only look once) is the simplest object detection architecture. It is not the most accurate, but it’s one of the fastest when it comes to prediction times.

YOLO grid

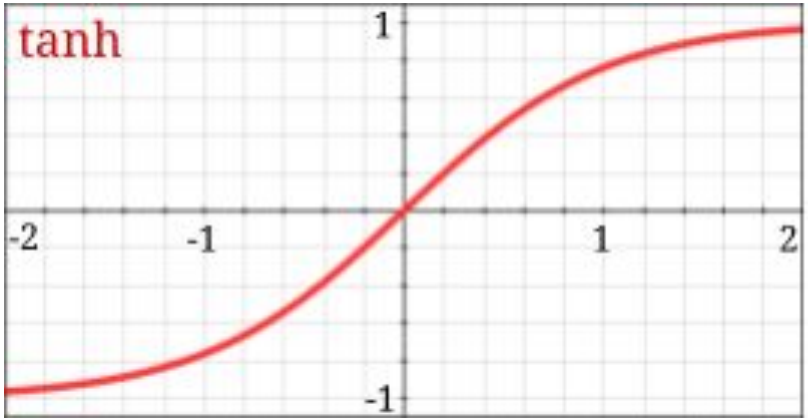


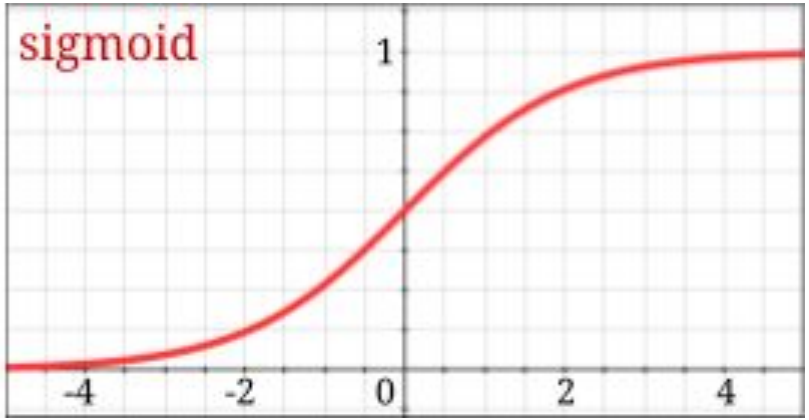
Object detection head



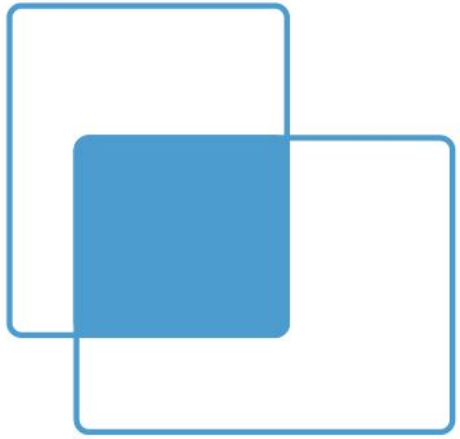
Activation functions

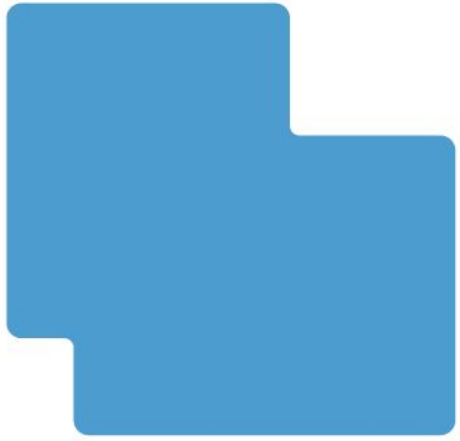
The tanh and sigmoid activation functions. Tanh outputs values in the [–1, 1] range, while the sigmoid function outputs them in the [0, 1] range.



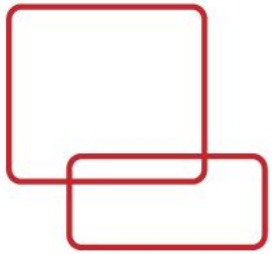


$$IOU = \frac{\text{Area of intersection}}{\text{Area of union}}$$

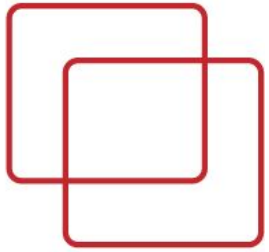




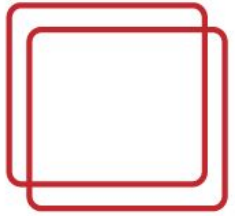
Examples



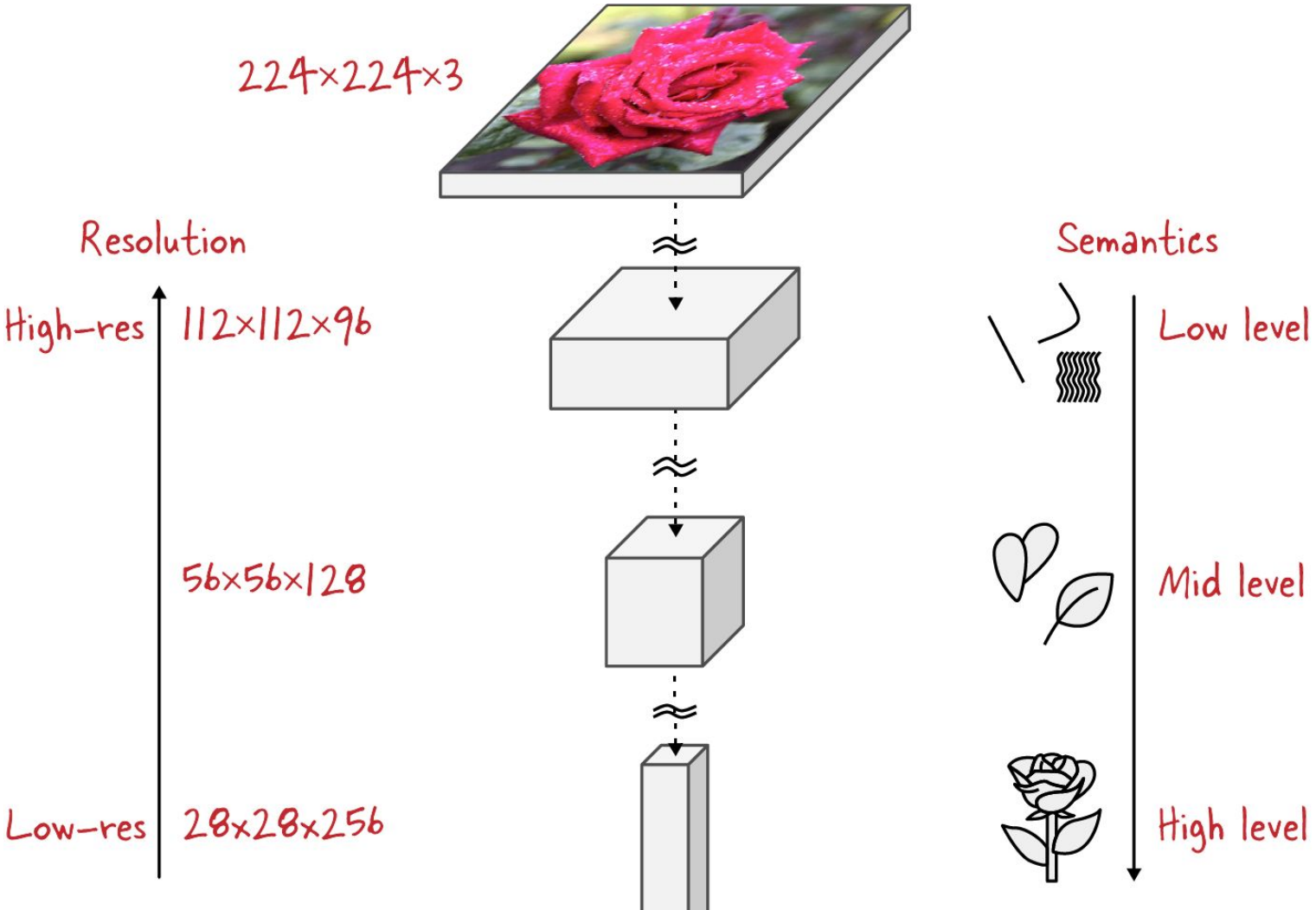
IOU = 0.1

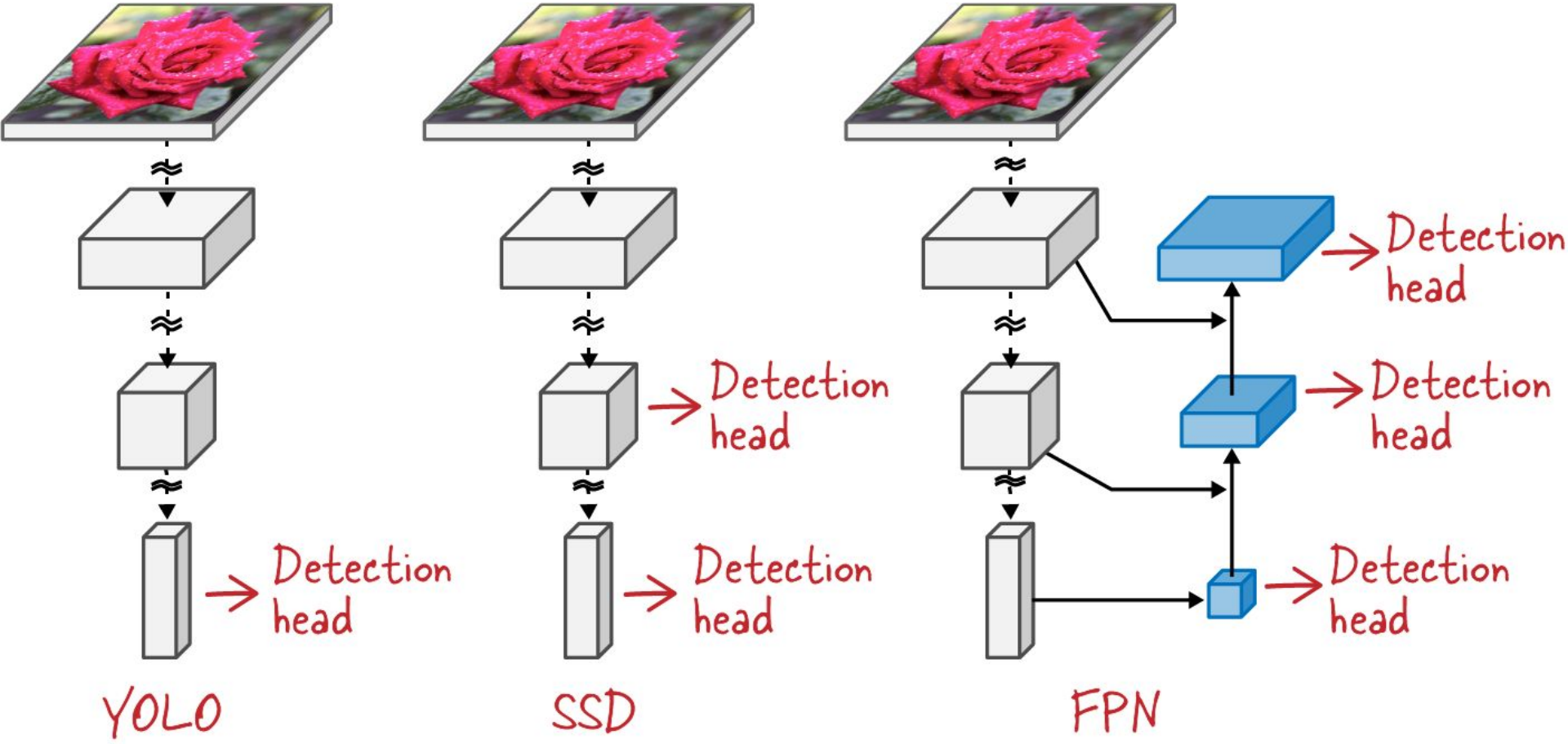


IOU = 0.3



IOU = 0.6


224×224×3
Resolution
High-res
112×112×96
56×56×128
Low-res
28×28×256
Semantics
Low level
Mid level
High level


Detection head
Detection head
Detection head
Detection head
Detection head
Detection head
YOLO
SSD
FPN

640×960×3

80×120×512

40×60×1024

20×30×2048

256 ch.

conv 1x1

conv 3x3

256 ch.

Detection head

upsample x2

256 ch.

conv 1x1

conv 3x3

256 ch.

Detection head

upsample x2

256 ch.

conv 1x1

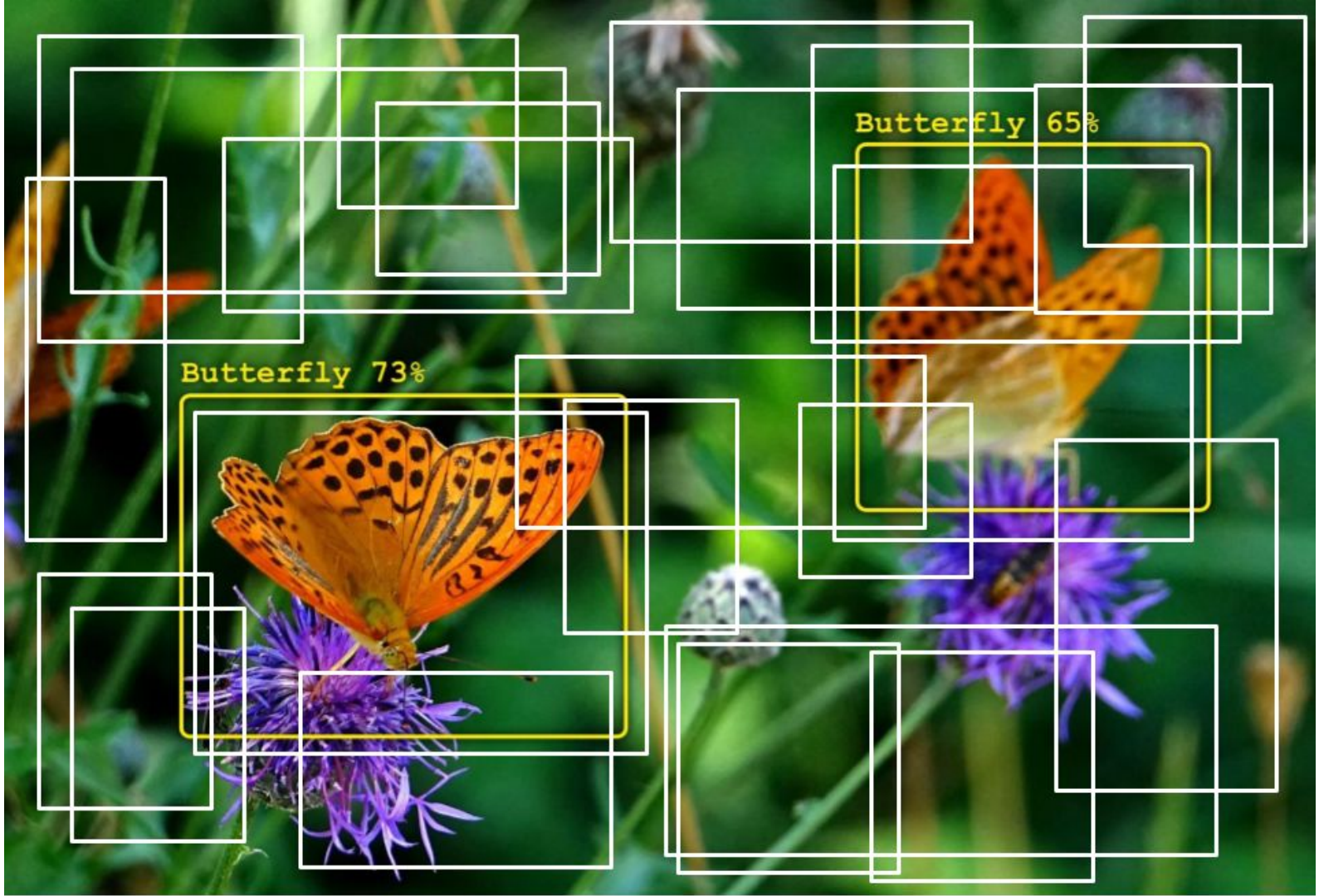
conv 3x3

256 ch.

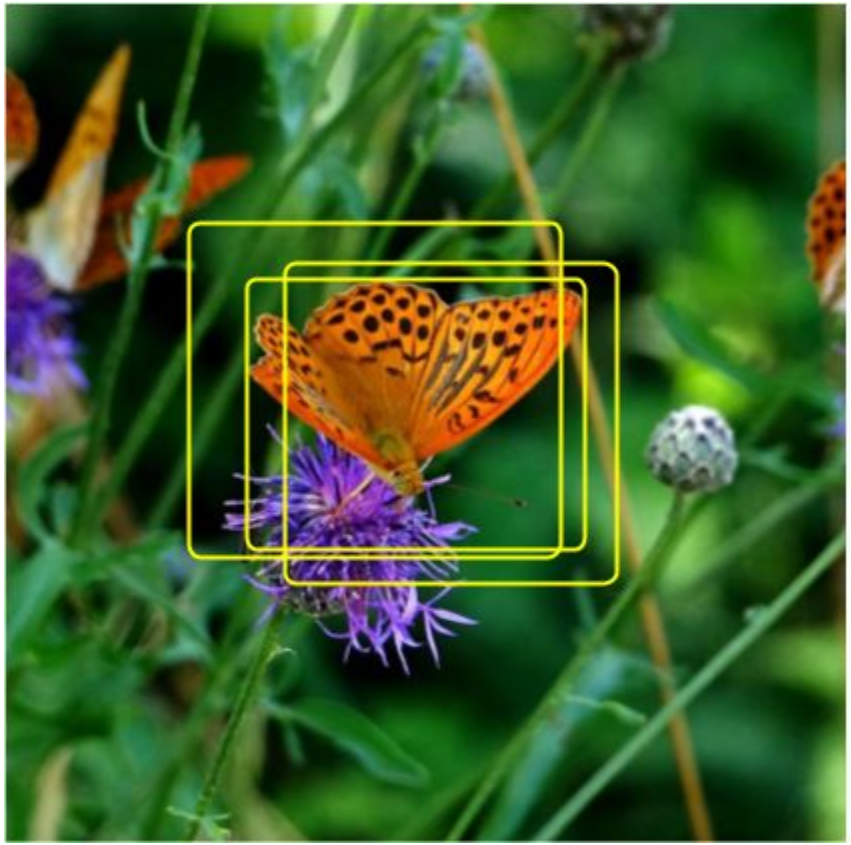
Detection head

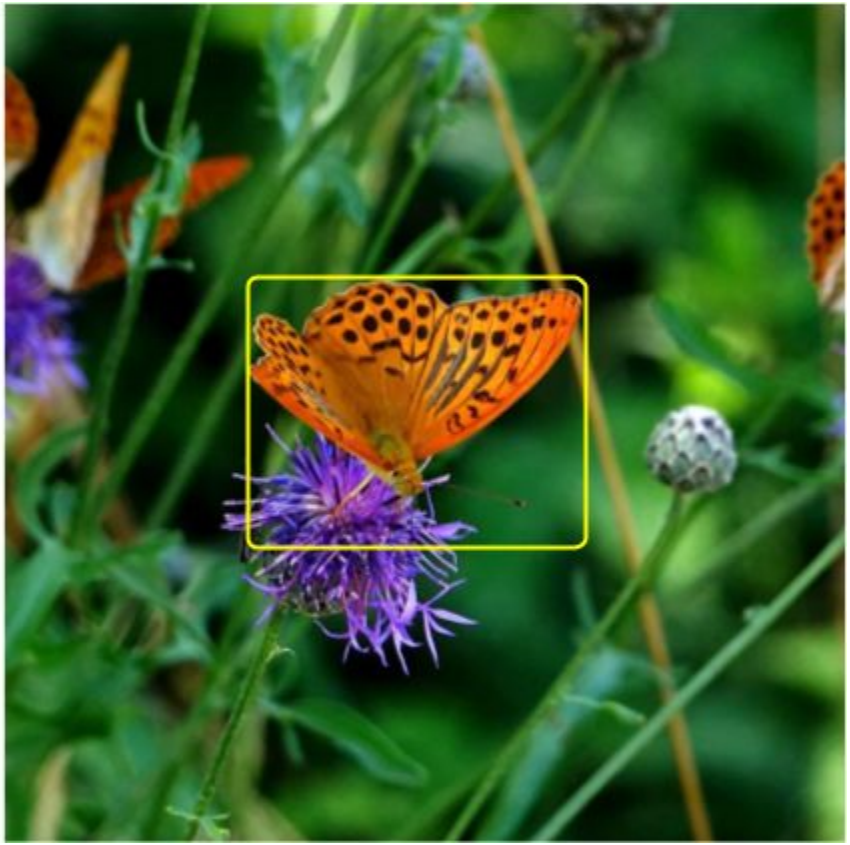
FPN

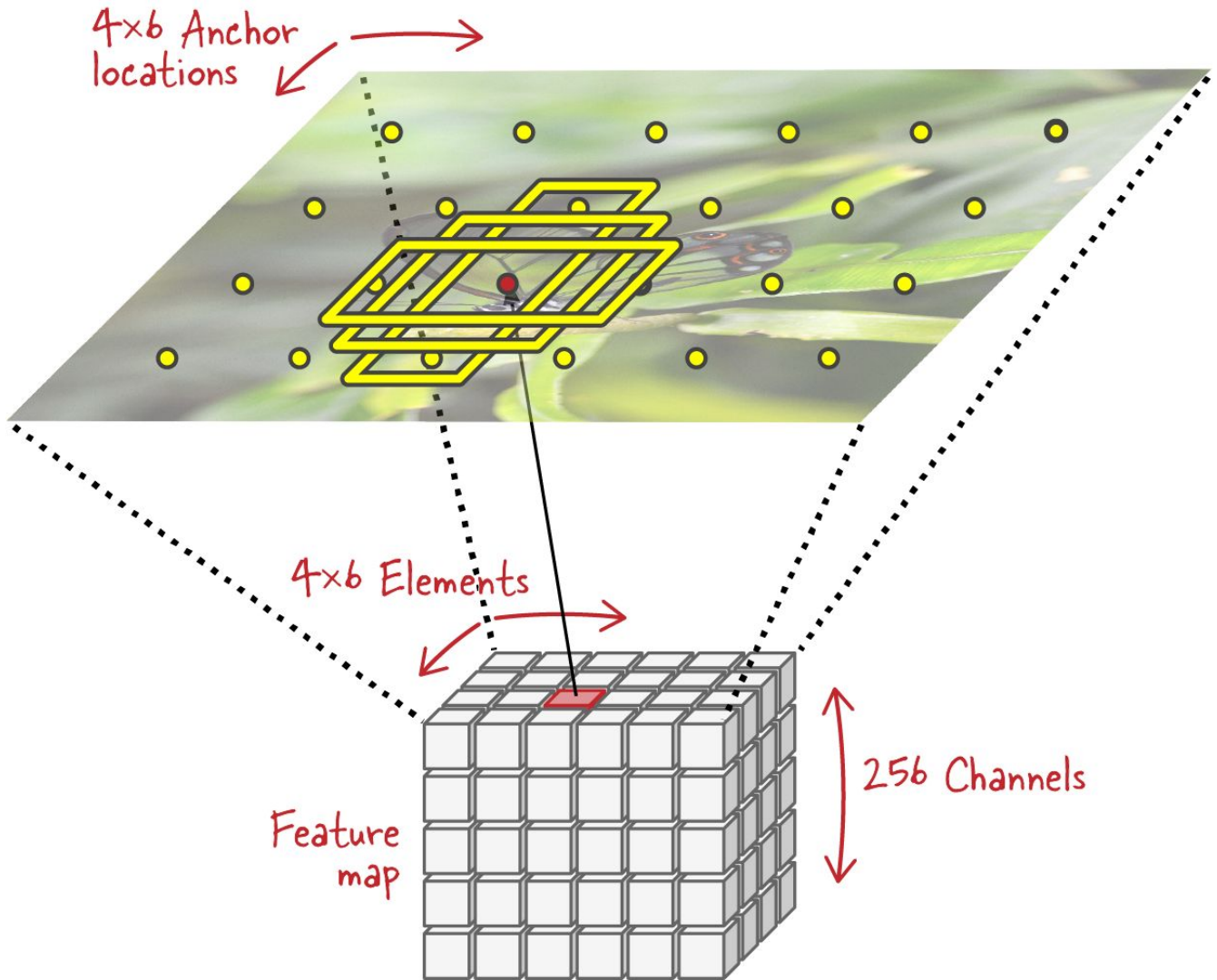
Anchor

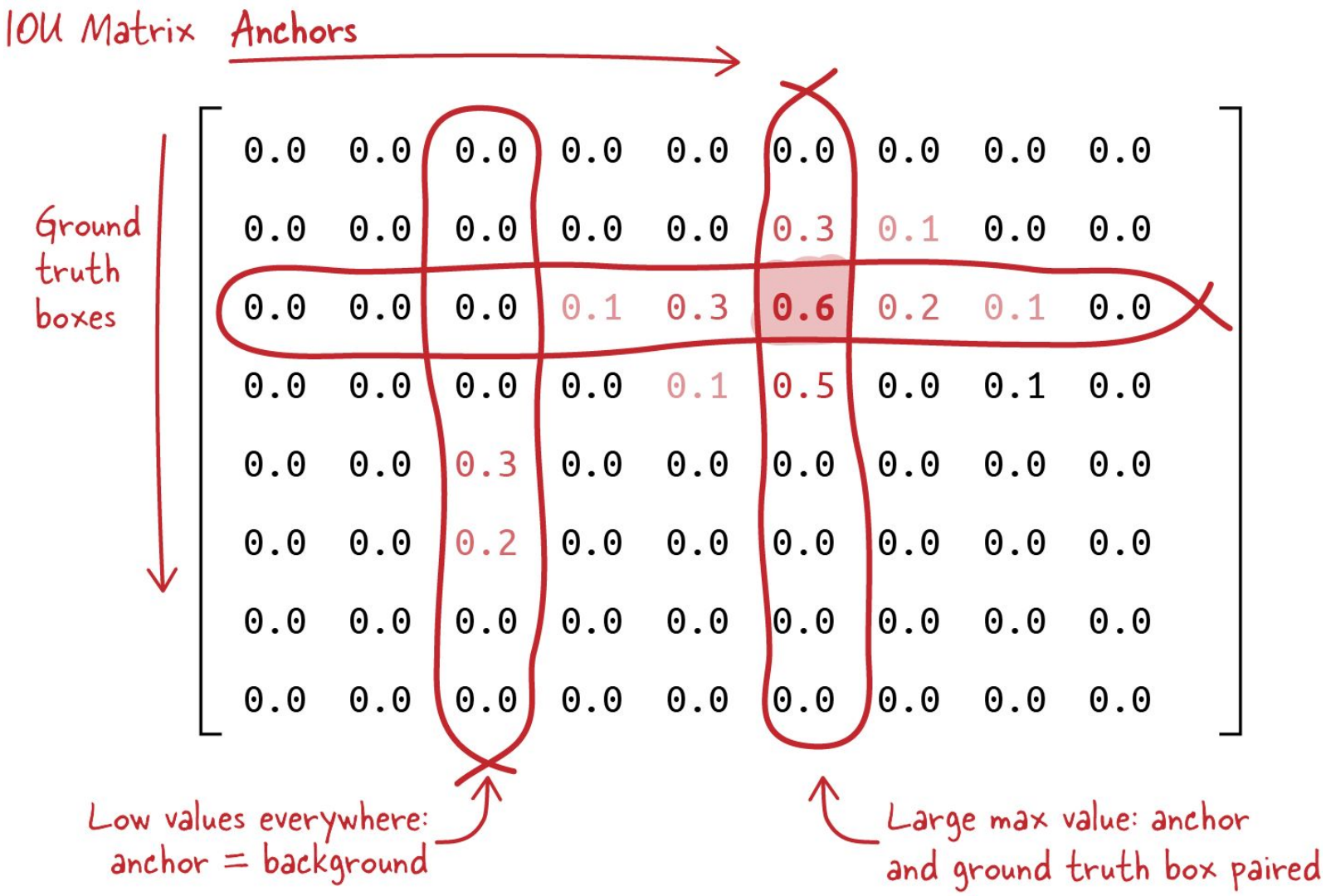


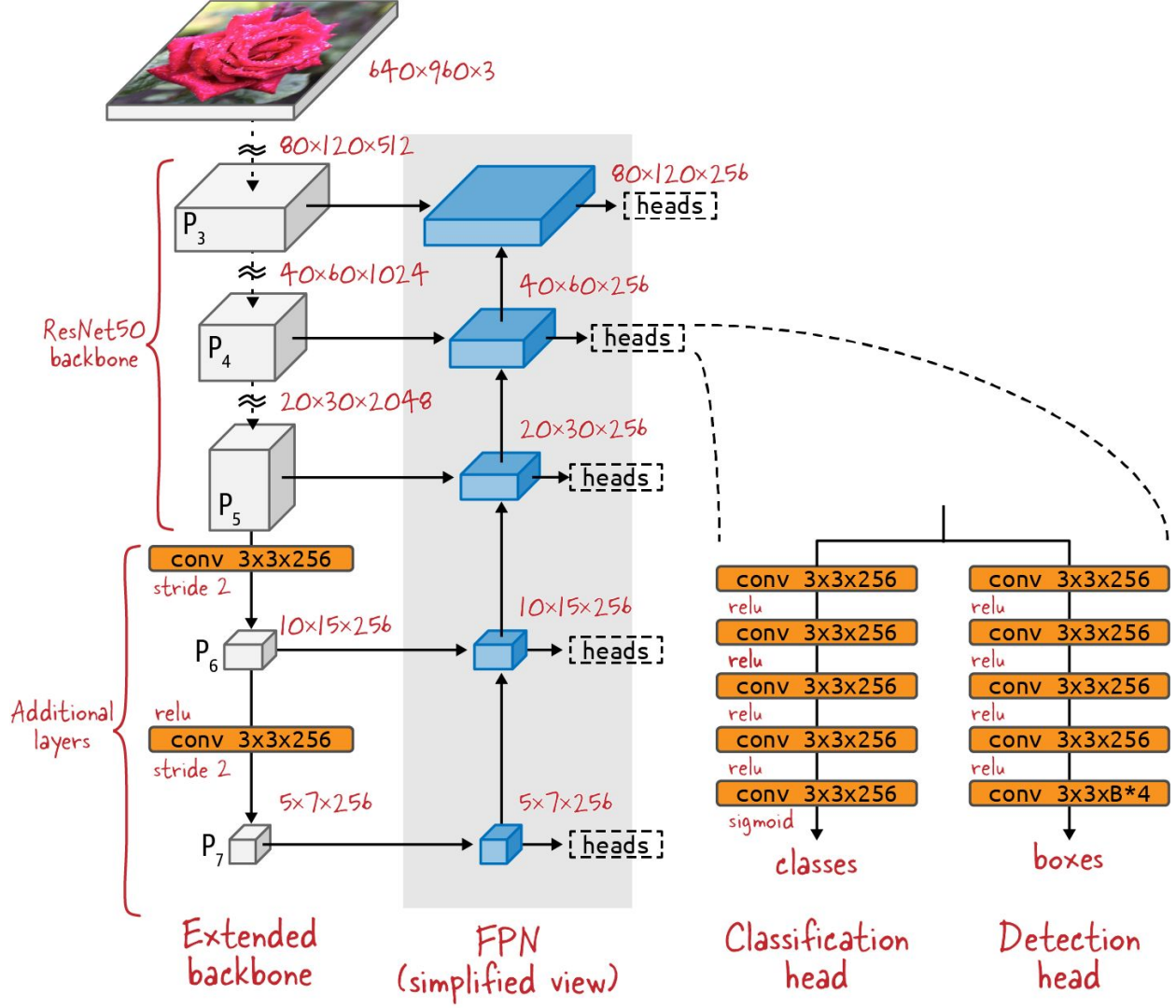
Non-maximum suppression (NMS)






4×6 Anchor locations
4×6 Elements
Feature map
256 Channels

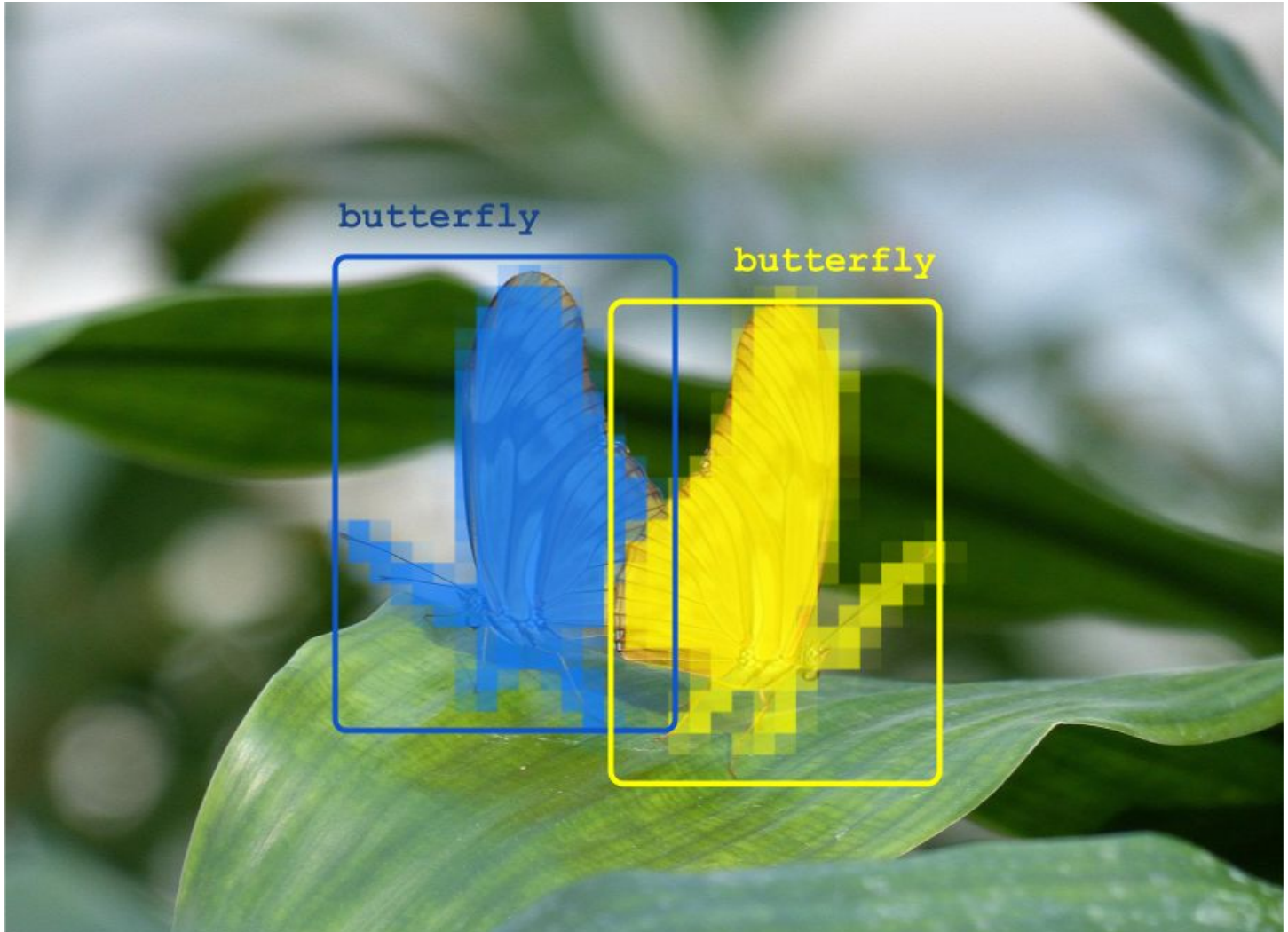

IOU Matrix
Anchors
Ground truth boxes
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.3 0.1 0.0 0.0
0.0 0.0 0.0 0.1 0.3 0.6 0.2 0.1 0.0
0.0 0.0 0.0 0.0 0.1 0.5 0.0 0.1 0.0
0.0 0.0 0.3 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.2 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Low values everywhere: anchor = background
Large max value: anchor and ground truth box paired

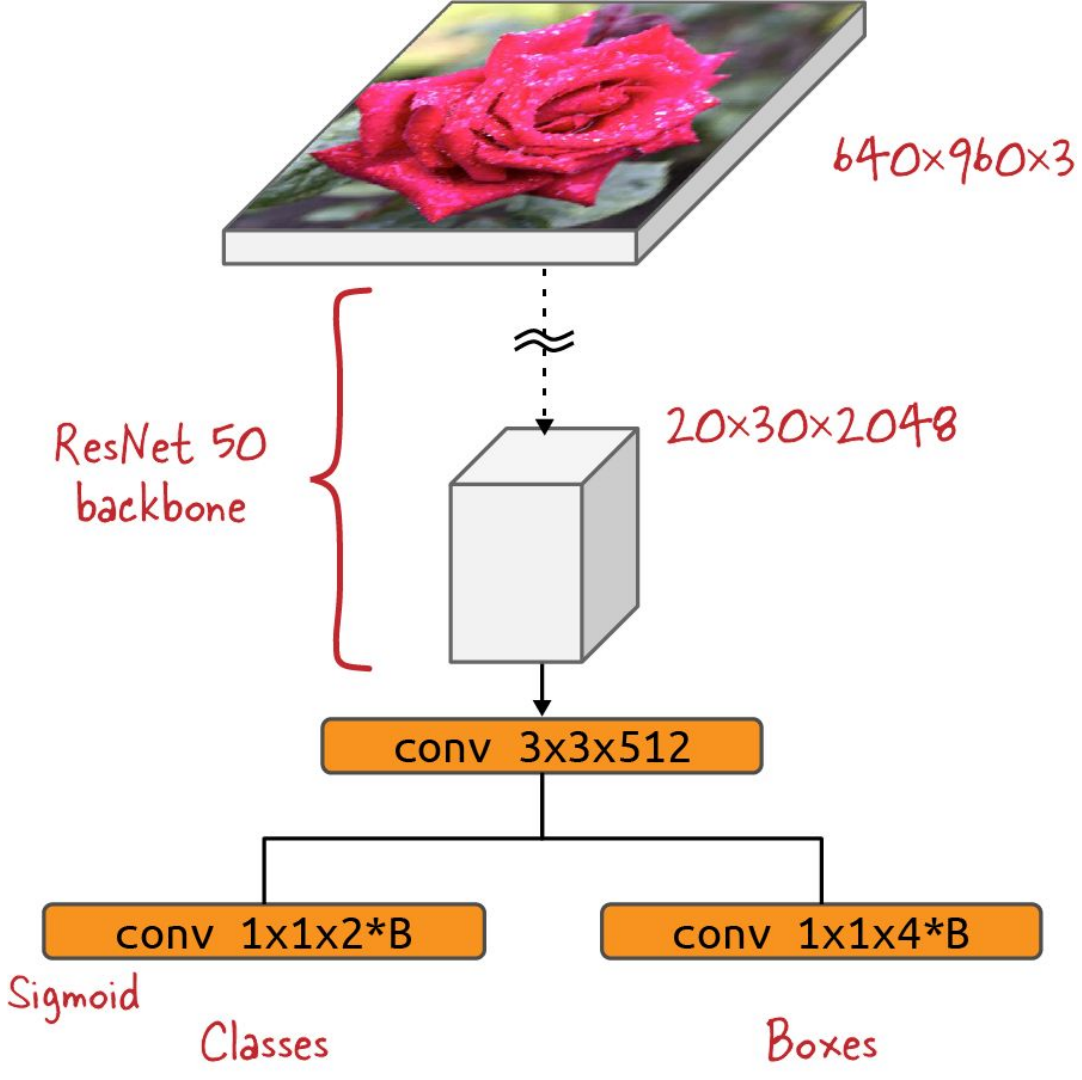

640×960×3
80×120×512
P3
80×120×256
heads
40×60×1024
P4
40×60×256
heads
ResNet50 backbone
20×30×2048
P5
20×30×256
heads
conv 3x3x256
stride 2
10×15×256
P6
10×15×256
heads
Additional layers
relu
conv 3x3x256
stride 2
5×7×256
P7
5×7×256
heads
conv 3x3x256
relu
conv 3x3x256
relu
conv 3x3x256
relu
conv 3x3x256
relu
conv 3x3x256
sigmoid
classes
conv 3x3x256
relu
conv 3x3x256
relu
conv 3x3x256
relu
conv 3x3x256
relu
conv 3x3xB*4
boxes
Extended backbone
FPN (simplified view)
Classification head
Detection head


IOU=0.42
butterfly 78%
butterfly 85%


IOU=0.42
butterfly 55%
butterfly 85%

Mask R-CNN and Instance Segmentation


butterfly
butterfly


640×960×3
ResNet 50
backbone
20×30×2048
conv 3x3x512
conv 1x1x2*B
conv 1x1x4*B
Sigmoid
Classes
Boxes

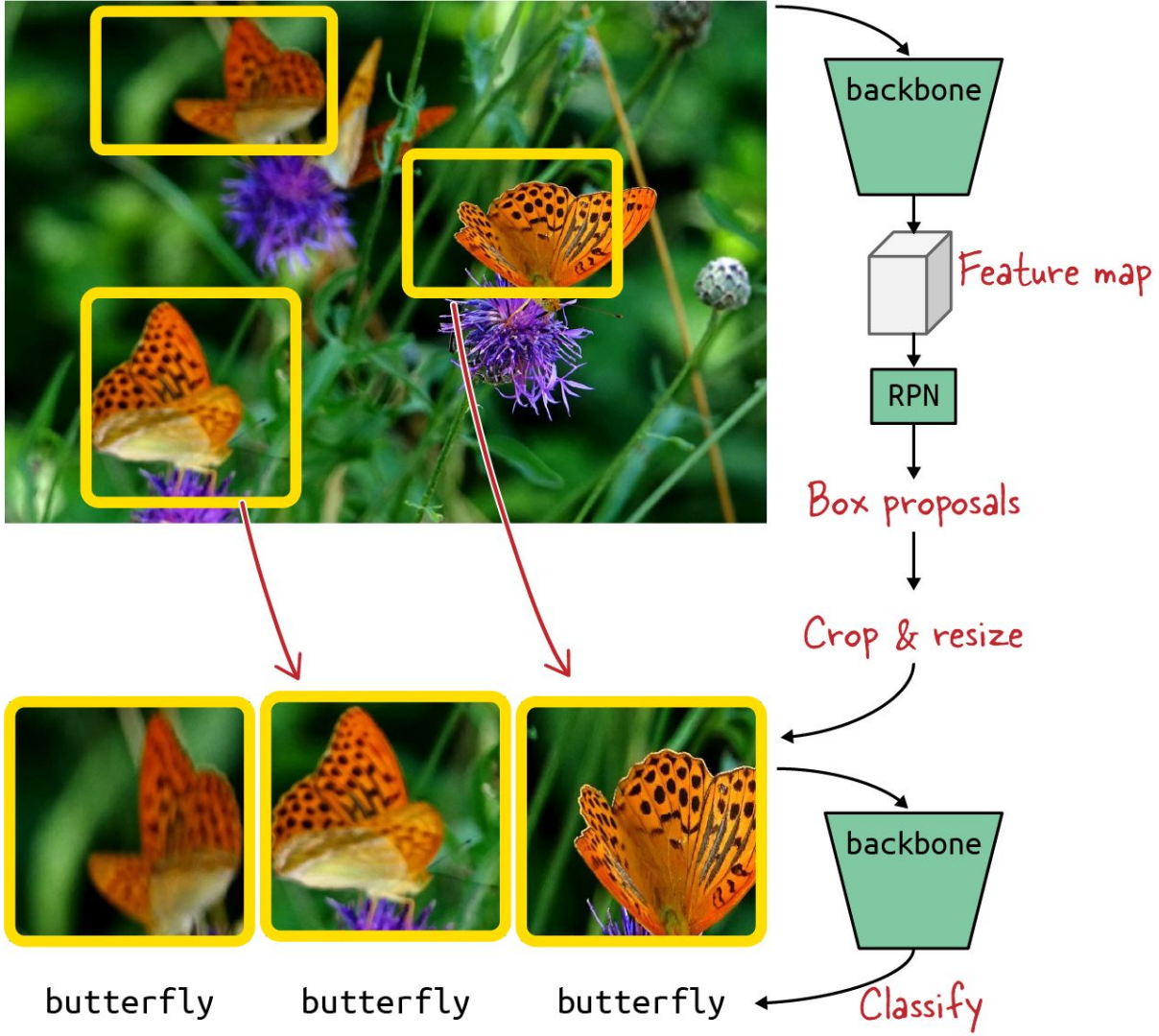
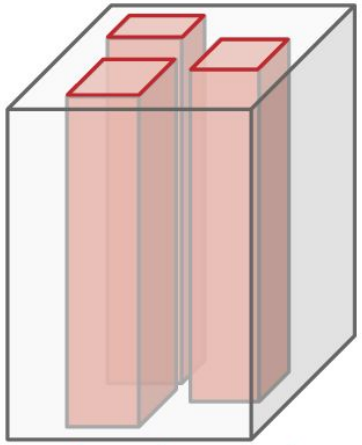

backbone
Feature map
RPN
Box proposals
Crop & resize
backbone
Classify
butterfly
butterfly
butterfly

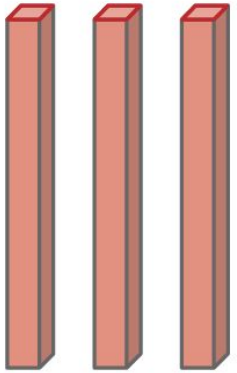
Image with ROIs

Feature map



40x60x256

Extracted feature maps



7x7x256

ROI map & align

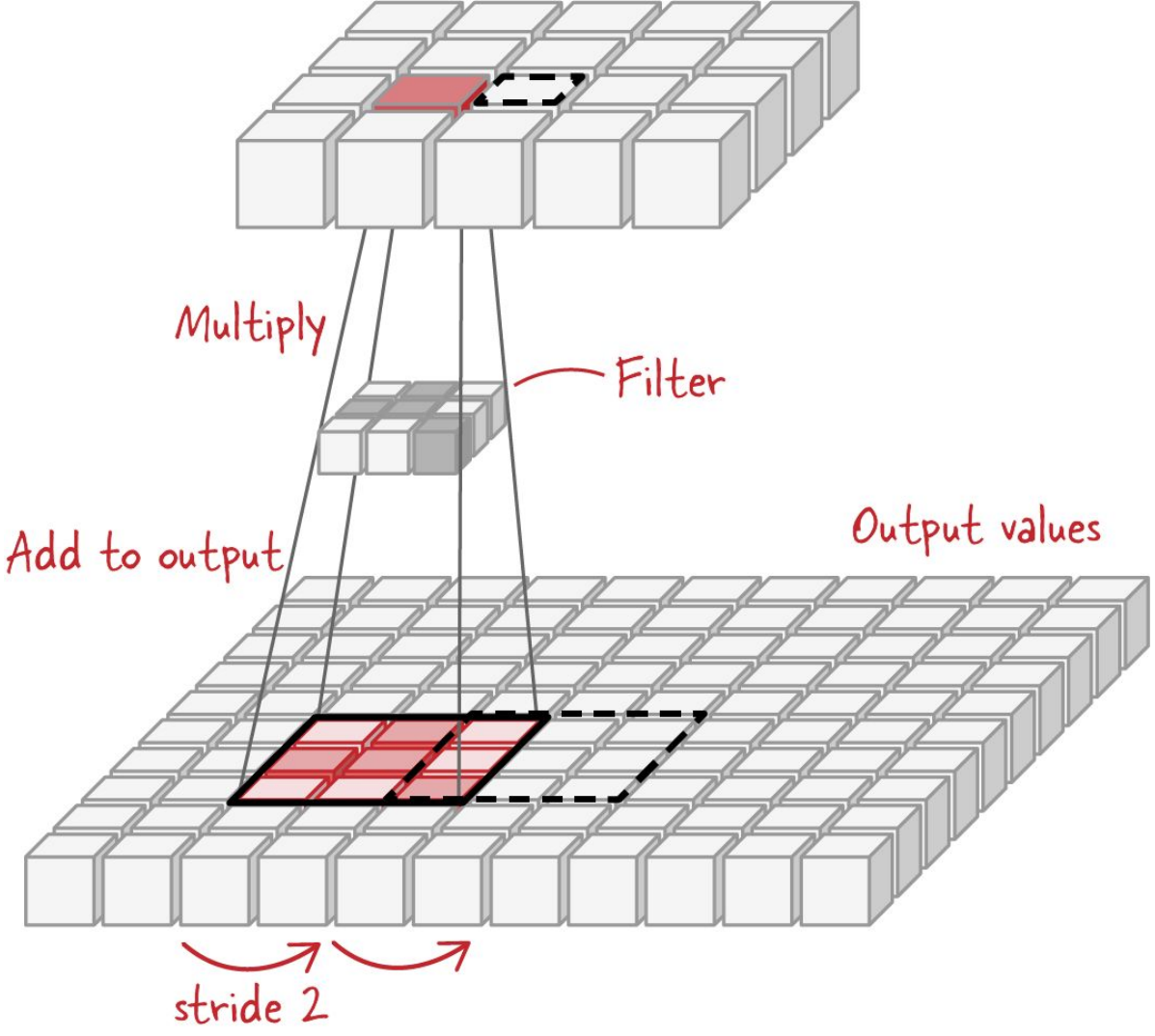
Crop & resize

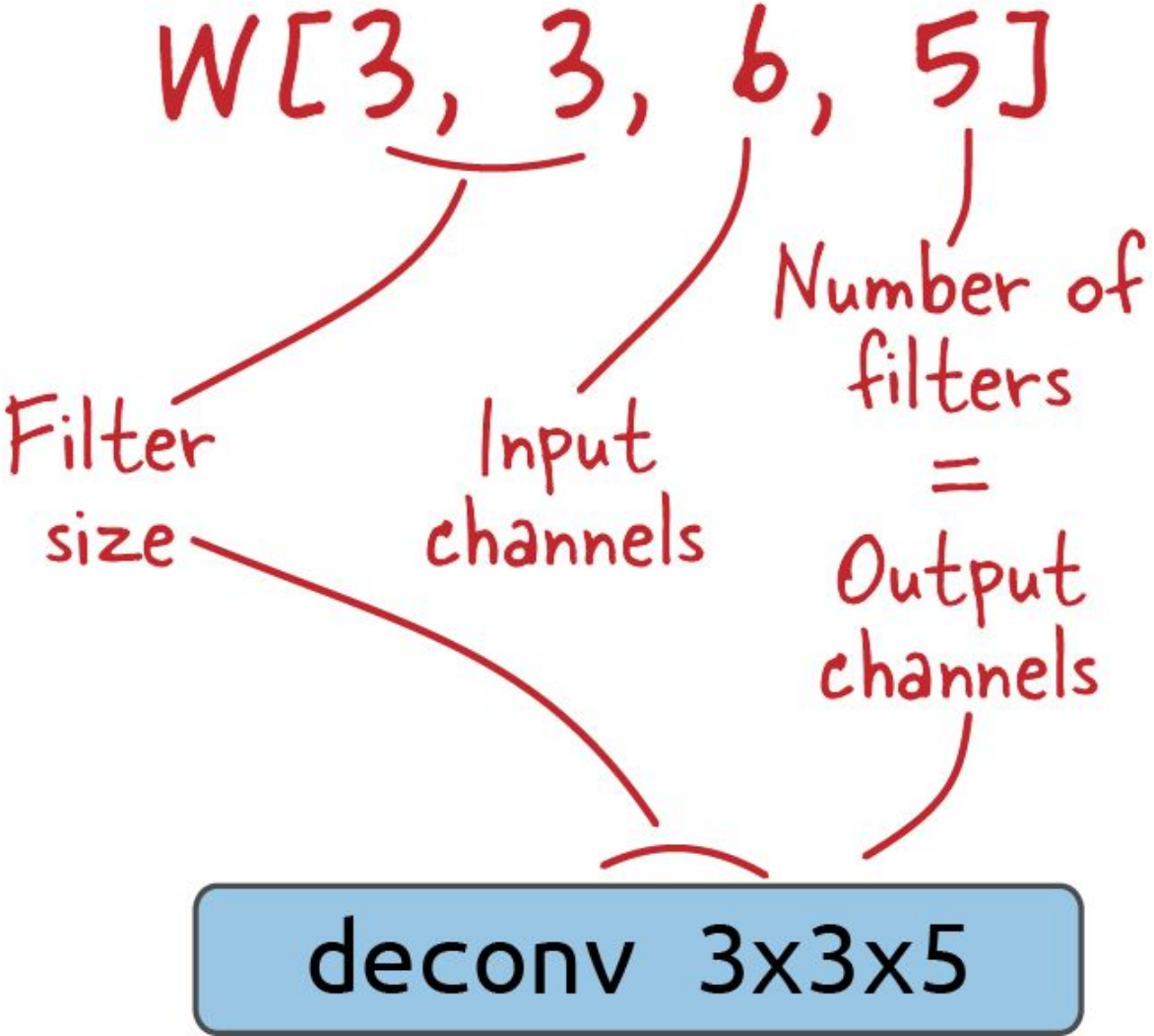
heads

Classification

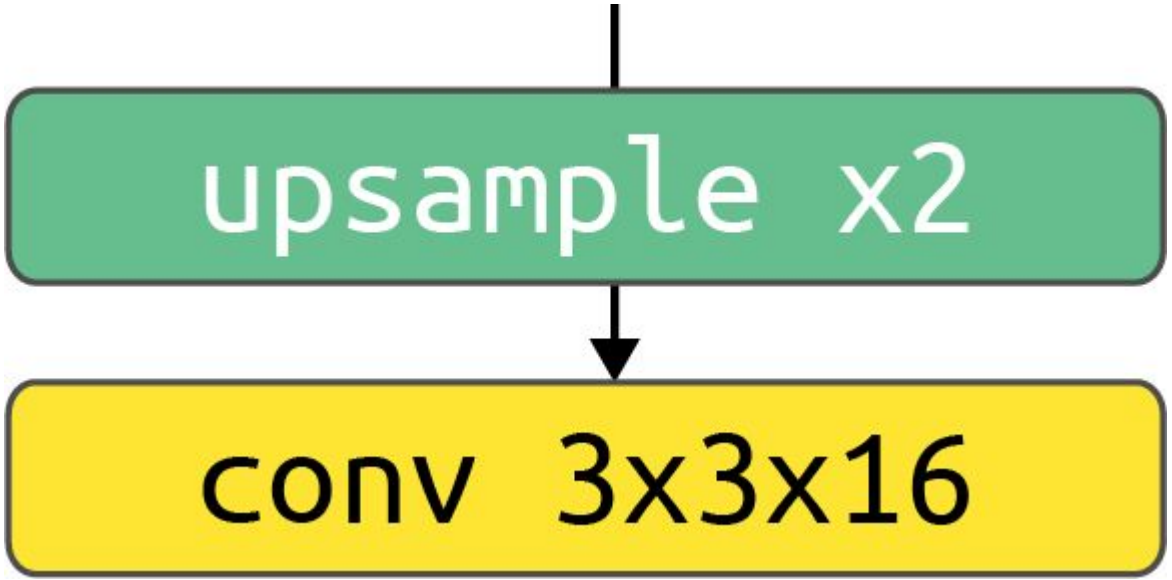
Box refinement

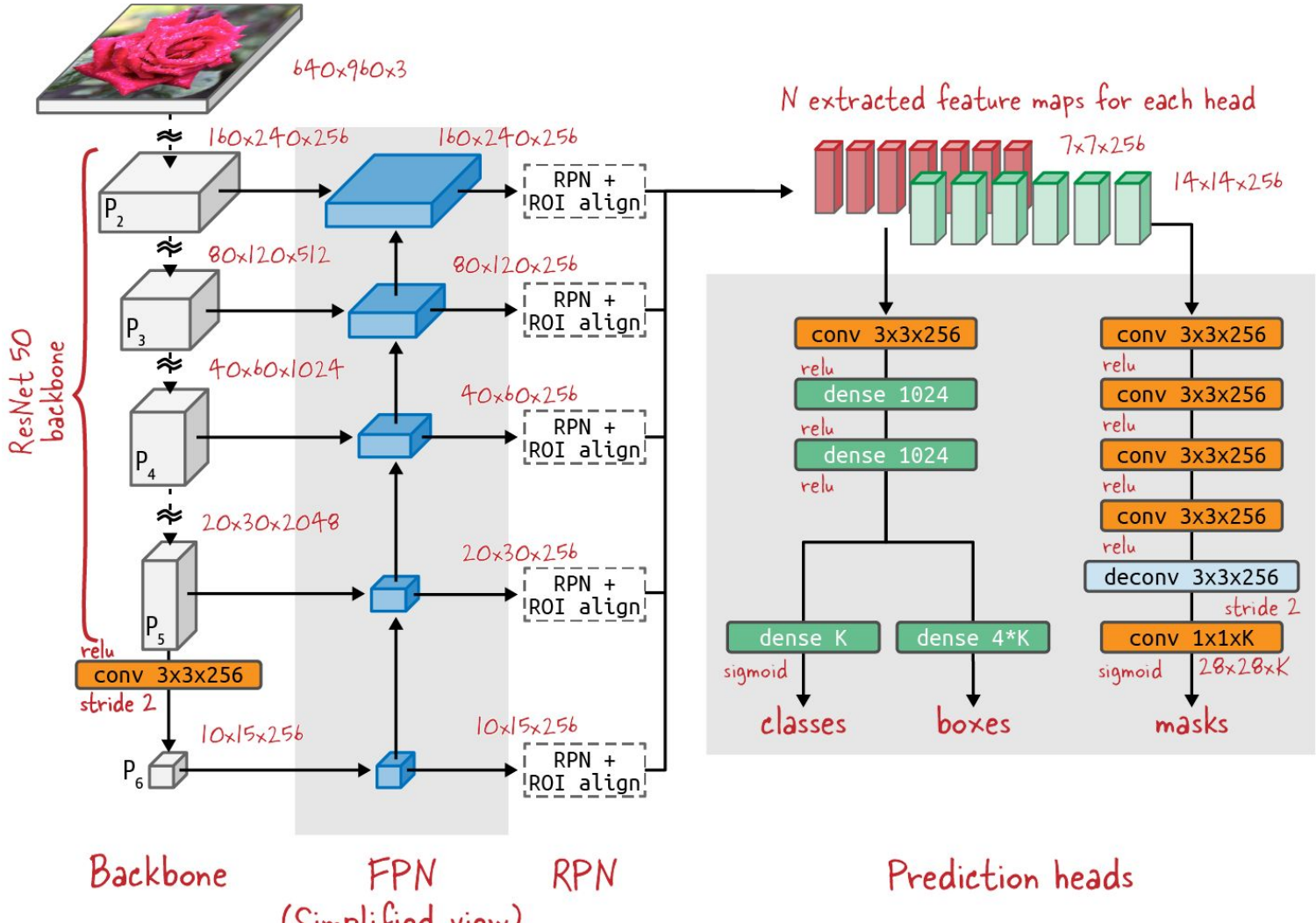
Mask generation

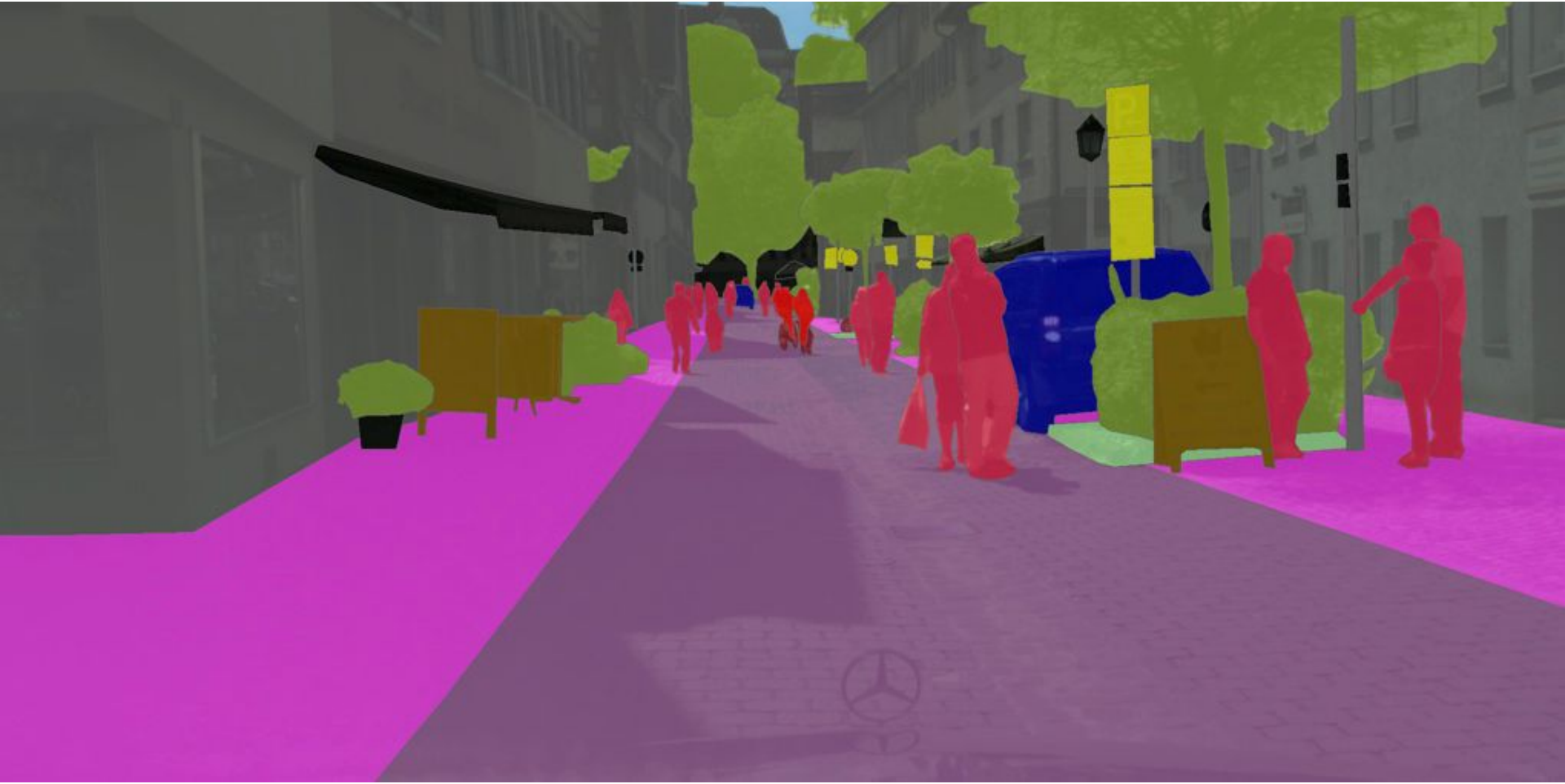

Multiply
Filter
Add to output
Output values
stride 2


W[3, 3, 6, 5]
Filter size
Input channels
Number of filters = Output channels
deconv 3x3x5

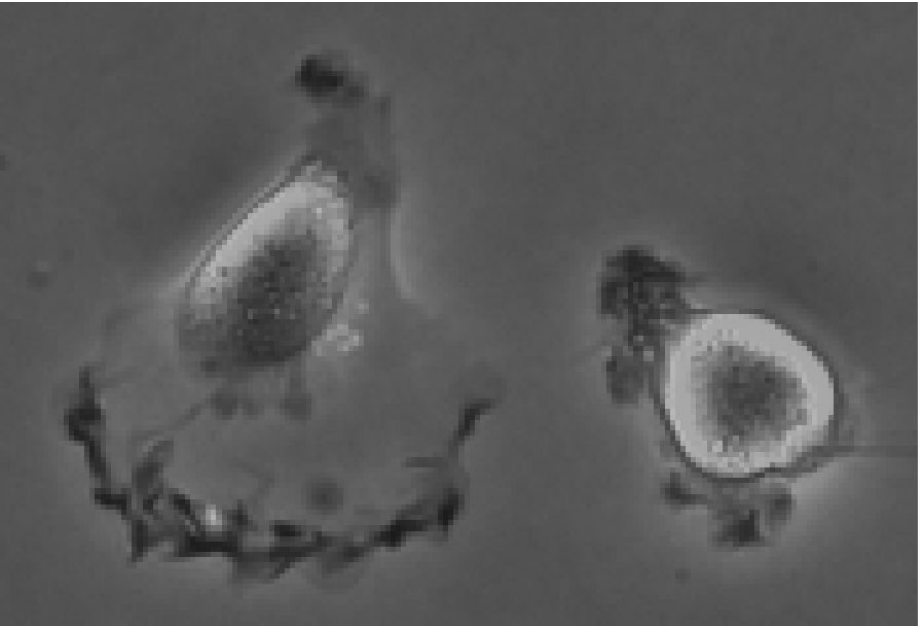
An “up-convolution” is a simple nearest neighbor upsampling operation followed by an ordinary convolutional layer.

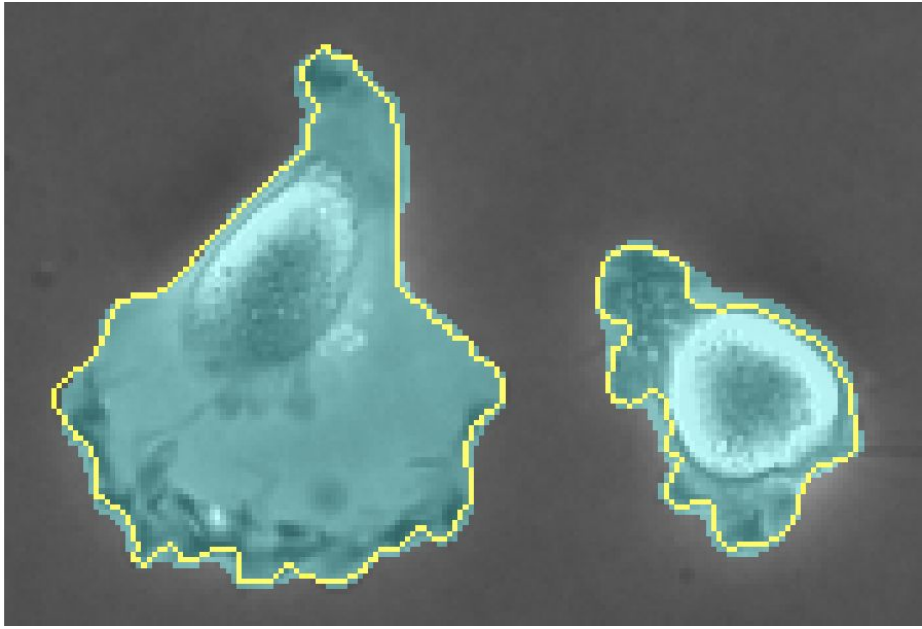


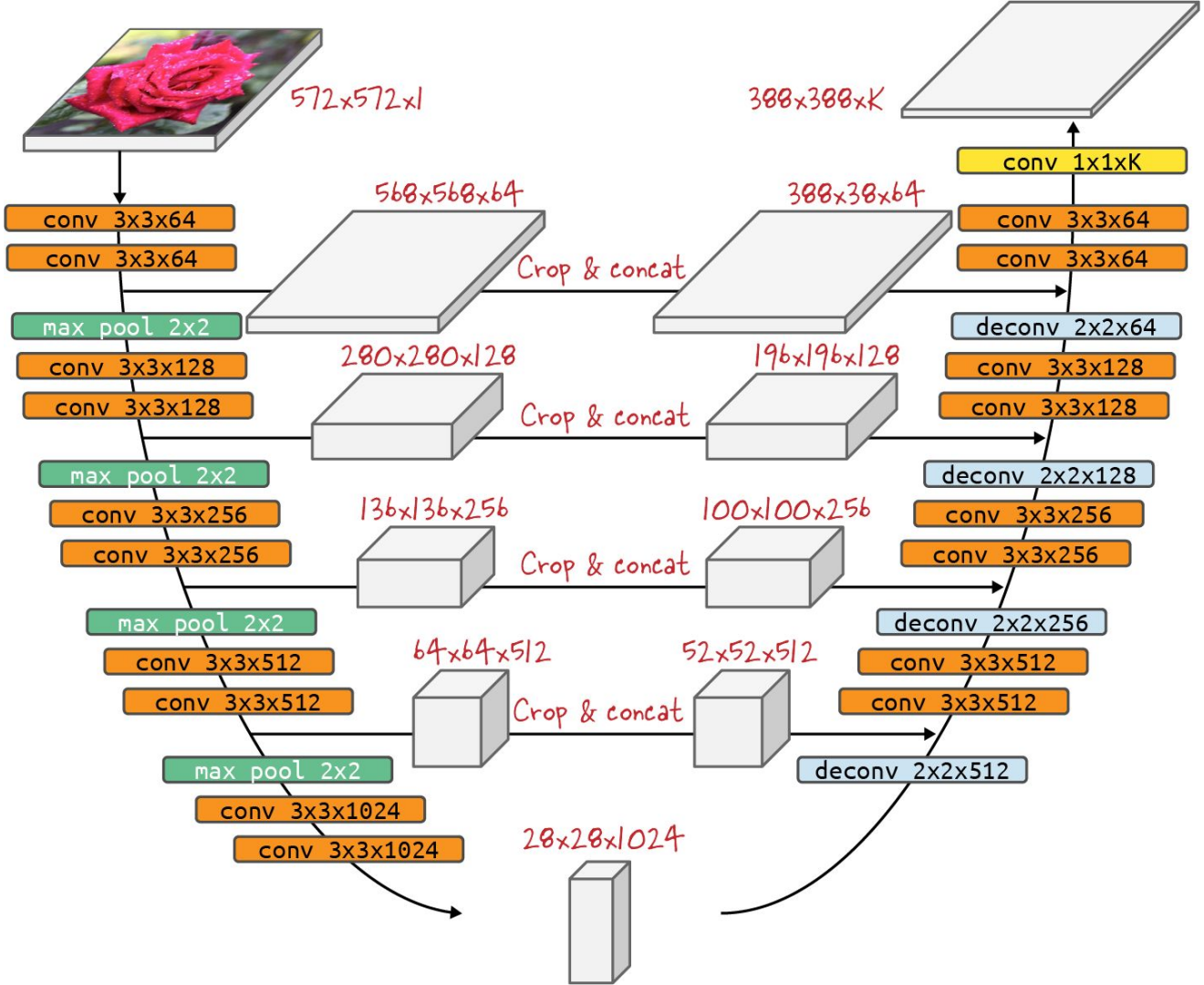

640x960x3
160x240x256
160x240x256
RPN + ROI align
80x120x512
80x120x256
RPN + ROI align
40x60x1024
40x60x256
RPN + ROI align
20x30x2048
20x30x256
RPN + ROI align
10x15x256
10x15x256
RPN + ROI align
P2
P3
P4
P5
P6
ResNet 50 backbone
relu
conv 3x3x256
stride 2
N extracted feature maps for each head
7x7x256
14x14x256
conv 3x3x256
relu
dense 1024
relu
dense 1024
relu
dense K
dense 4*K
sigmoid
classes
boxes
conv 3x3x256
relu
conv 3x3x256
relu
conv 3x3x256
relu
conv 3x3x256
relu
deconv 3x3x256
stride 2
conv 1x1xK
sigmoid
28x28xK
masks
Backbone
FPN
(Simplified view)
RPN
Prediction heads



The U-Net architecture was designed to segment biomedical images such as these microscopy cell images.





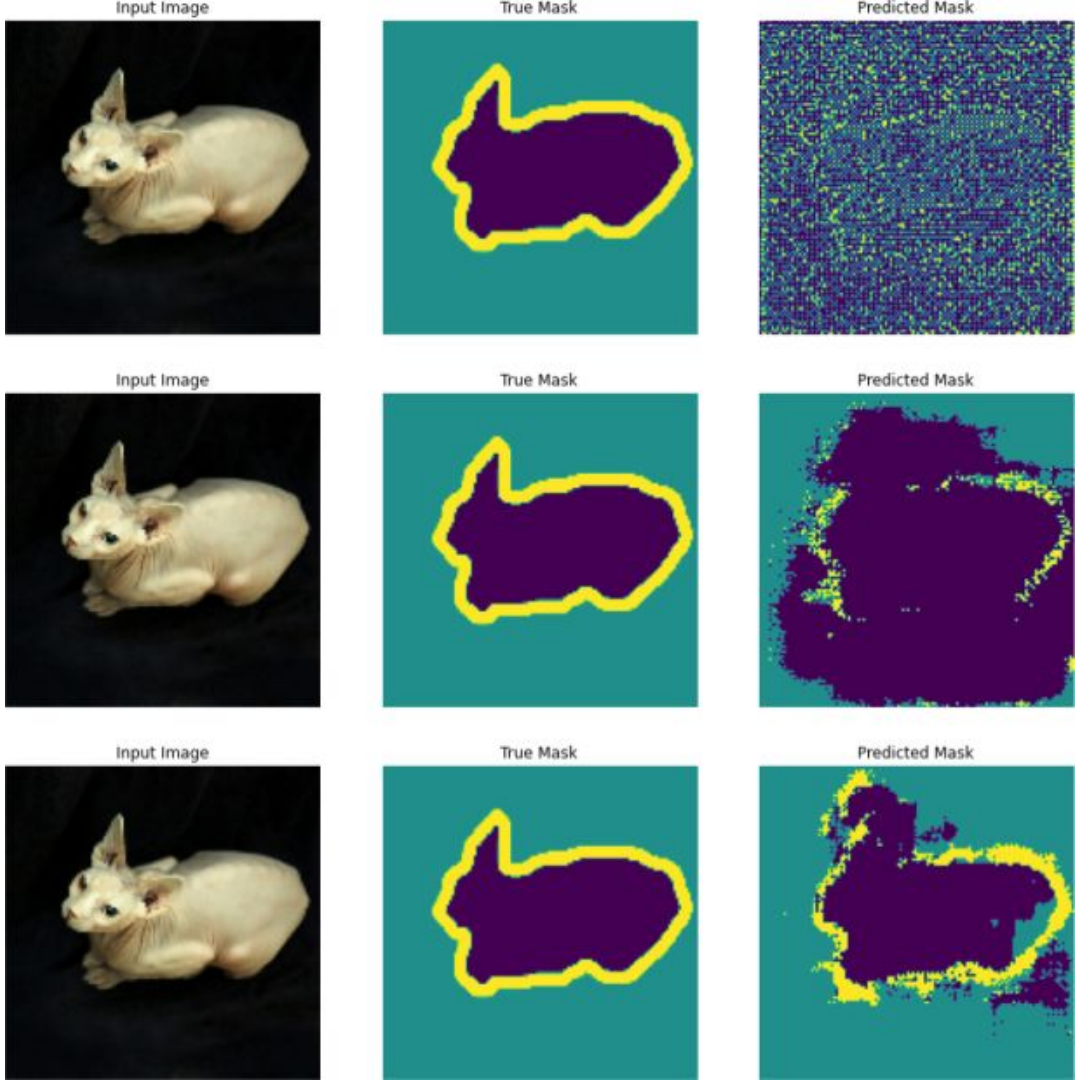

572x572x1
conv 3x3x64
conv 3x3x64
568x568x64
max pool 2x2
conv 3x3x128
conv 3x3x128
280x280x128
max pool 2x2
conv 3x3x256
conv 3x3x256
136x136x256
max pool 2x2
conv 3x3x512
conv 3x3x512
64x64x512
max pool 2x2
conv 3x3x1024
conv 3x3x1024
28x28x1024
Crop & concat
Crop & concat
Crop & concat
Crop & concat
388x388xK
conv 1x1xK
388x38x64
conv 3x3x64
conv 3x3x64
deconv 2x2x64
196x196x128
conv 3x3x128
conv 3x3x128
deconv 2x2x128
100x100x256
conv 3x3x256
conv 3x3x256
deconv 2x2x256
52x52x512
conv 3x3x512
conv 3x3x512
deconv 2x2x512



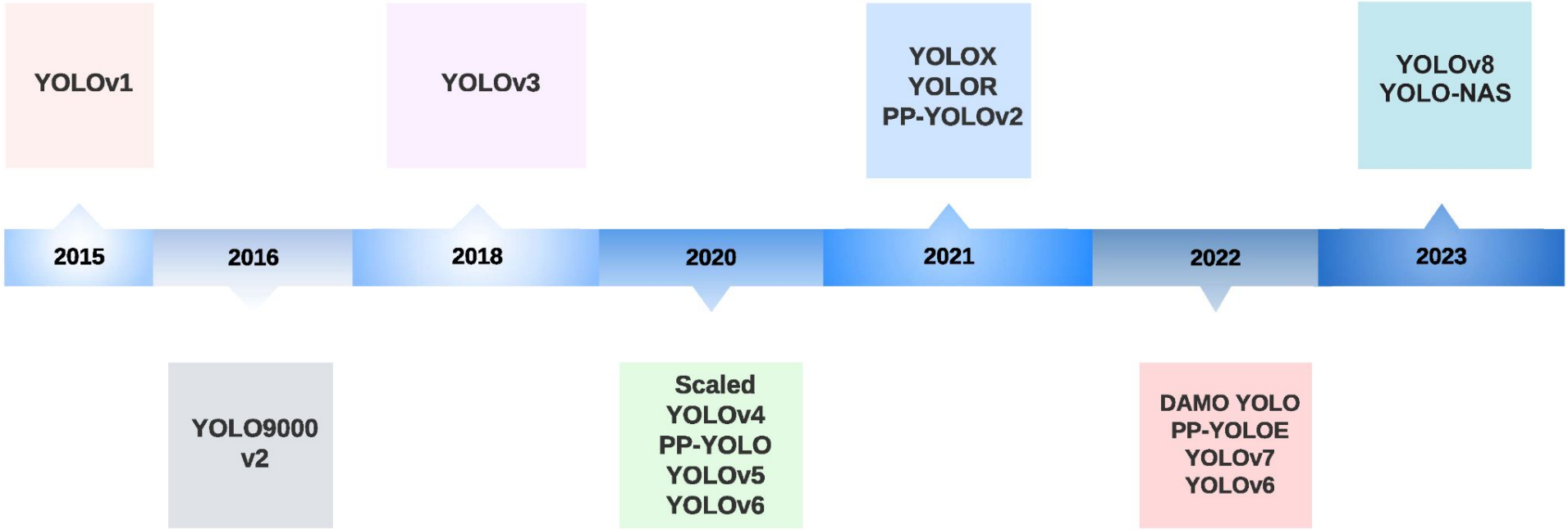




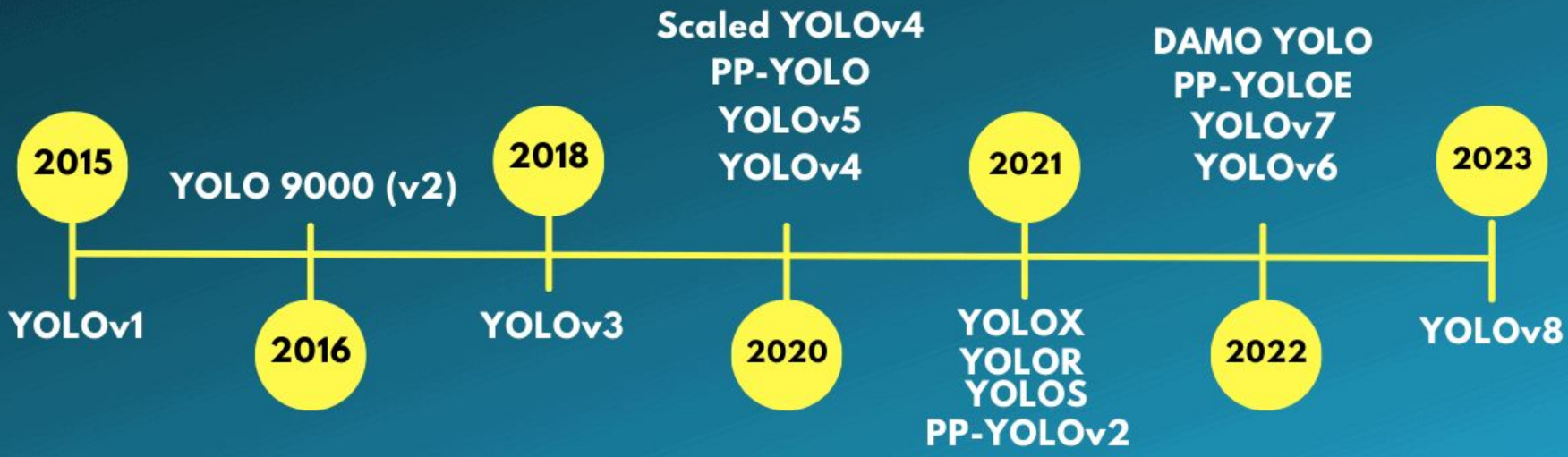


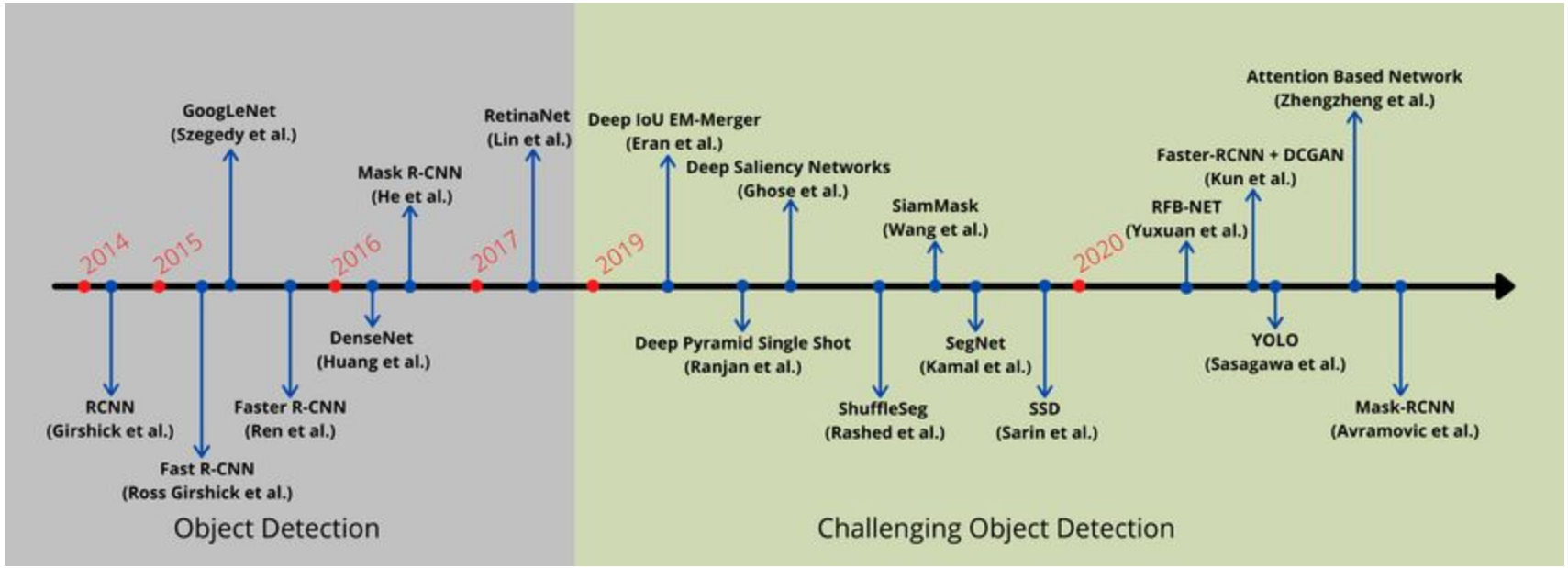


Input Image
True Mask
Predicted Mask
Input Image
True Mask
Predicted Mask
Input Image
True Mask
Predicted Mask

YOLO


YOLOv1
YOLOv3
YOLOX
YOLOR
PP-YOLOv2
YOLOv8
YOLO-NAS
2015
2016
2018
2020
2021
2022
2023
YOLO9000
v2
Scaled
YOLOv4
PP-YOLO
YOLOv5
YOLOv6
DAMO YOLO
PP-YOLOE
YOLOv7
YOLOv6

YOLO Object Detection Models Timeline



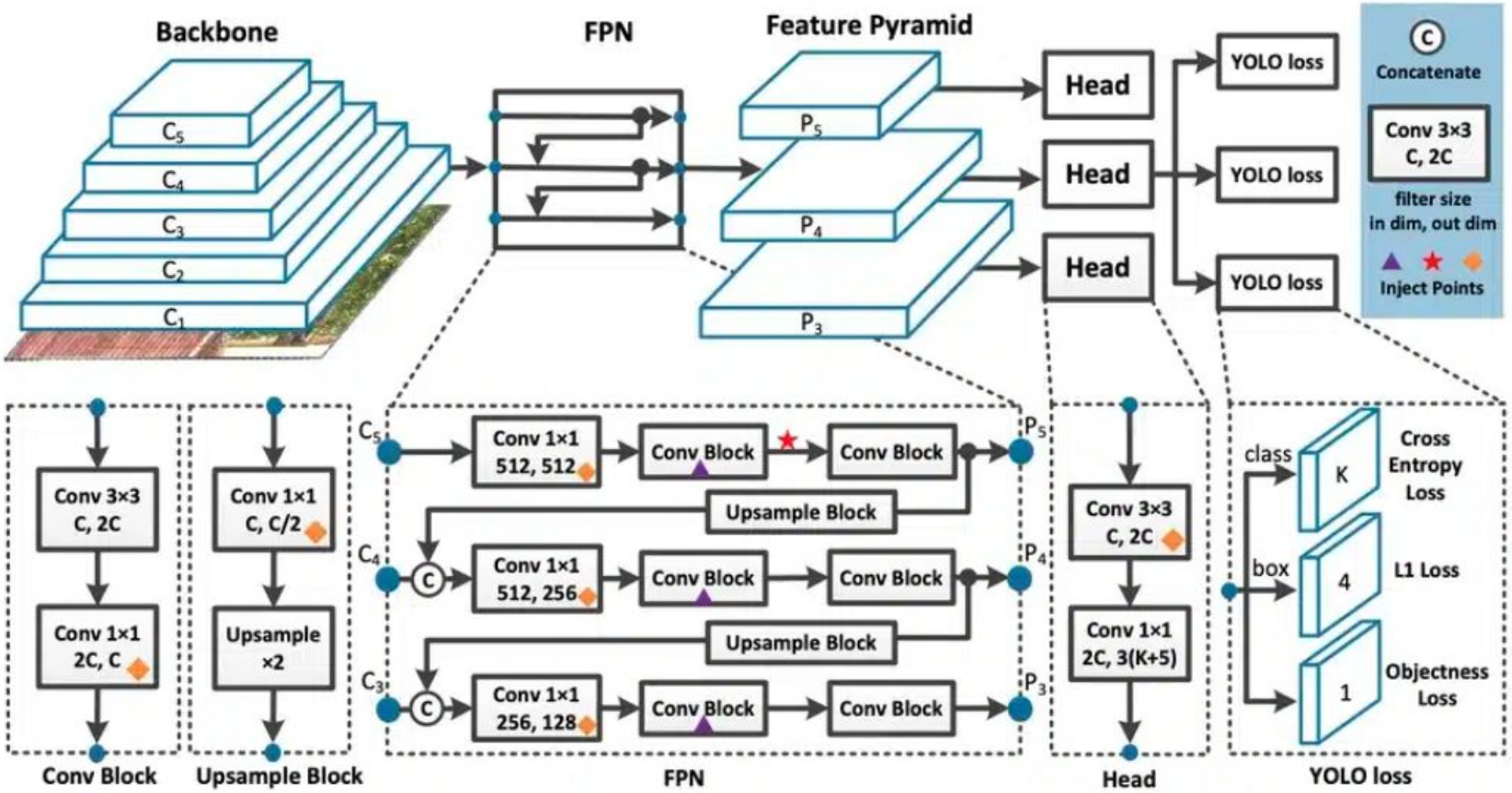

2014
2015
2016
2017
2019
2020
RCNN
(Girshick et al.)
Fast R-CNN
(Ross Girshick et al.)
Faster R-CNN
(Ren et al.)
GoogLeNet
(Szegedy et al.)
DenseNet
(Huang et al.)
Mask R-CNN
(He et al.)
RetinaNet
(Lin et al.)
Deep Pyramid Single Shot
(Ranjan et al.)
Deep IoU EM-Merger
(Eran et al.)
Deep Saliency Networks
(Ghose et al.)
ShuffleSeg
(Rashed et al.)
SegNet
(Kamal et al.)
SiamMask
(Wang et al.)
SSD
(Sarin et al.)
YOLO
(Sasagawa et al.)
RFB-NET
(Yuxuan et al.)
Faster-RCNN + DCGAN
(Kun et al.)
Mask-RCNN
(Avramovic et al.)
Attention Based Network
(Zhengzheng et al.)
Object Detection
Challenging Object Detection

pyimagesearch

Mean Average Precision (mAP) Using the COCO Evaluator

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{|TP_c|}{|FP_c| + |TP_c|}$$

YOLOv7

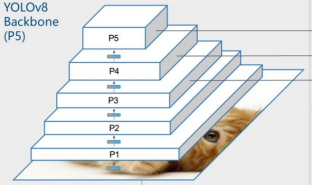

Backbone
FPN
Feature Pyramid
C5
C4
C3
C2
C1
P5
P4
P3
Head
Head
Head
YOLO loss
YOLO loss
YOLO loss
C
Concatenate
Conv 3×3
C, 2C
filter size
in dim, out dim
Inject Points
Conv 3×3
C, 2C
Conv 1×1
2C, C
Conv Block
Conv 1×1
C, C/2
Upsample
×2
Upsample Block
C5
Conv 1×1
512, 512
Conv Block
Conv Block
P5
Upsample Block
C4
C
Conv 1×1
512, 256
Conv Block
Conv Block
P4
Upsample Block
C3
C
Conv 1×1
256, 128
Conv Block
Conv Block
P3
FPN
Conv 3×3
C, 2C
Conv 1×1
2C, 3(K+5)
Head
class
K
Cross
Entropy
Loss
box
4
L1 Loss
1
Objectness
Loss
YOLO loss

YOLOv8

RangeKing

Backbone

YOLOv8 Backbone (P5)



640×640×3

Conv k=3, s=2, p=1 — 0 — P1

320×320×64×w

Conv k=3, s=2, p=1 — 1 — P2

160×160×128×w

C2f shortcut=True, n=3×d — 2

160×160×128×w

Conv k=3, s=2, p=1 — 3 — P3

80×80×256×w

C2f shortcut=True, n=6×d — 4 — 80×80×256×w Stride=8

80×80×256×w

Conv k=3, s=2, p=1 — 5 — P4

40×40×512×w

C2f shortcut=True, n=6×d — 6 — 40×40×512×w Stride=16

40×40×512×w

Conv k=3, s=2, p=1 — 7 — P5

20×20×512×w×r

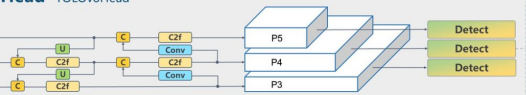
C2f shortcut=True, n=3×d — 8

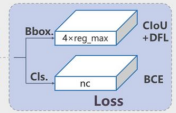
20×20×512×w×r

SPPF — 9 — 20×20×512×w×r Stride=32

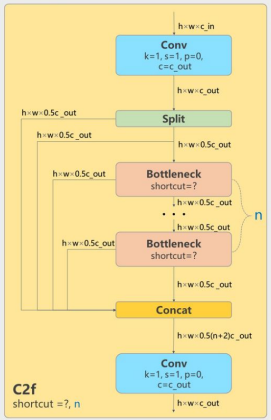
Note: height×width×channel

Head YOLOv8Head

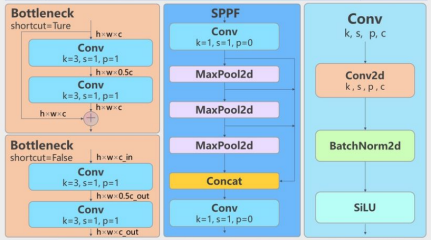


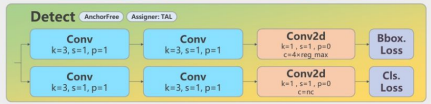


Details



model	d (depth_multiple)	w (width_multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0





C2f shortcut=False, n=3×d — 15 — P3

80×80×512×w

Concat — 14

80×80×256×w

Upsample — 13

40×40×512×w

C2f shortcut=False, n=3×d — 12

40×40×512×w(1+r)

Concat — 11

40×40×512×wxr

Upsample — 10

20×20×512×wxr

80×80×256×w → Detect

80×80×256×w

Conv k=3, s=2, p=1 — 16 — P3

40×40×256×w

Concat — 17

40×40×512×w

C2f shortcut=False, n=3×d — 18 — P4 — 40×40×512×w → Detect

40×40×512×w

Conv k=3, s=2, p=1 — 19

20×20×512×w

Concat — 20

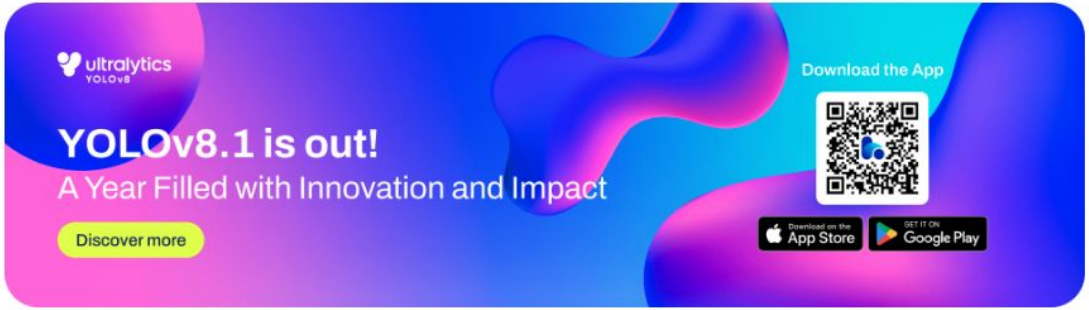
20×20×512×w(1+r)

C2f shortcut=False, n=3×d — 21 — P5 — 20×20×512×w → Detect

Backbone

Head

Home



Ultralytics CI passing codecov 79% DOI 10.5281/zenodo.7347926 docker pulls 46k Discord 369 online
Run on Gradient Open in Colab Open in Kaggle

Introducing Ultralytics YOLOv8, the latest version of the acclaimed real-time object detection and image segmentation model. YOLOv8 is built on cutting-edge advancements in deep learning and computer vision, offering unparalleled performance in terms of speed and accuracy. Its streamlined design makes it suitable for various applications and easily adaptable to different hardware platforms, from edge devices to cloud APIs.

Explore the YOLOv8 Docs, a comprehensive resource designed to help you understand and utilize its features

ultralytics / ultralytics Public

Sponsor | Notifications | Fork 4k

Code | Issues 554 | Pull requests 119 | Discussions | Actions | Projects 1 | Wiki | Security | Insights

main | 24 Branches | 1 Tags | Go to file | Code

Laughing-q and glenn-jocher ultralytics 8.1.20 add YOLOv8x-World sup... 59ed47c · 1 hour ago | 937 Commits

.github	Model typehints Docker fix (#8306)	last week
docker	ultralytics 8.1.18 add cmake for building onnxsim on a...	last week
docs	ultralytics 8.1.20 add YOLOv8x-World support (#8539)	1 hour ago
examples	OBB Docs updates (#7568)	2 months ago
tests	Add missing single-line docstrings (#8362)	last week
ultralytics	ultralytics 8.1.20 add YOLOv8x-World support (#8539)	1 hour ago
.gitignore	ultralytics 8.0.163 add new gpu-latest runner to CI ac...	7 months ago
.pre-commit-config.yaml	YAML reformat (#7669)	2 months ago
CITATION.cff	YOLOv8.1 blog, Explorer notebook and 2023 > 2024 upda...	2 months ago
CONTRIBUTING.md	ultralytics 8.0.224 Counting and Heatmaps updates (#...	3 months ago
LICENSE	Update LICENSE to AGPL-3.0 (#2031)	last year
README.md	Add https://youtu.be/3VryynorQeo to README and fix con...	2 months ago
README.zh-CN.md	Add https://youtu.be/3VryynorQeo to README and fix con...	2 months ago
mkdocs.yml	Add TFLite Docs Integrations Page (#8522)	2 hours ago
pyproject.toml	Update mkdocs-ultralytics-plugin>=0.0.44 (#8347)	last week

About

NEW - YOLOv8 🚀 in PyTorch > ONNX > OpenVINO > CoreML > TFLite

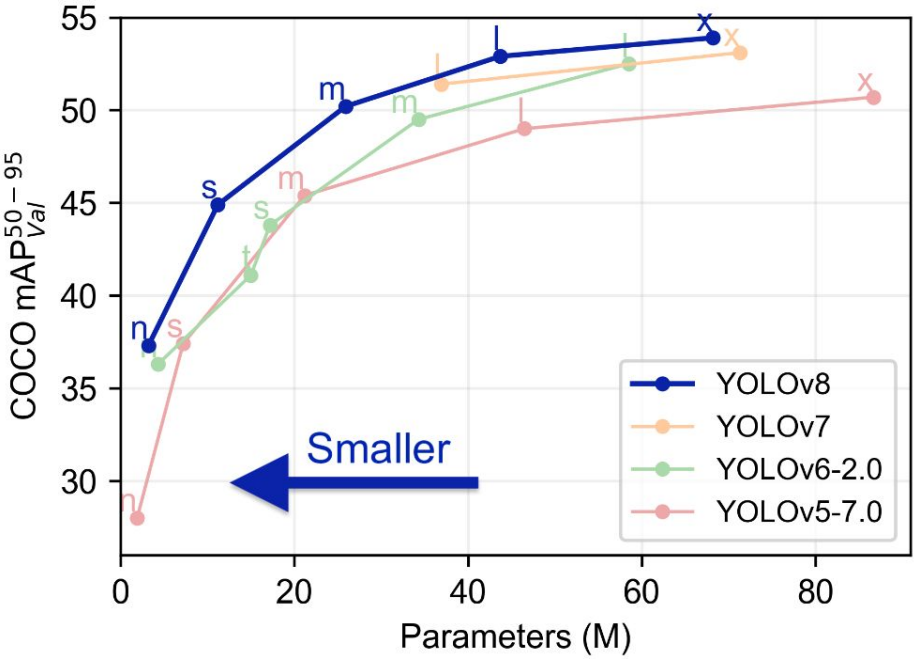
docs.ultralytics.com

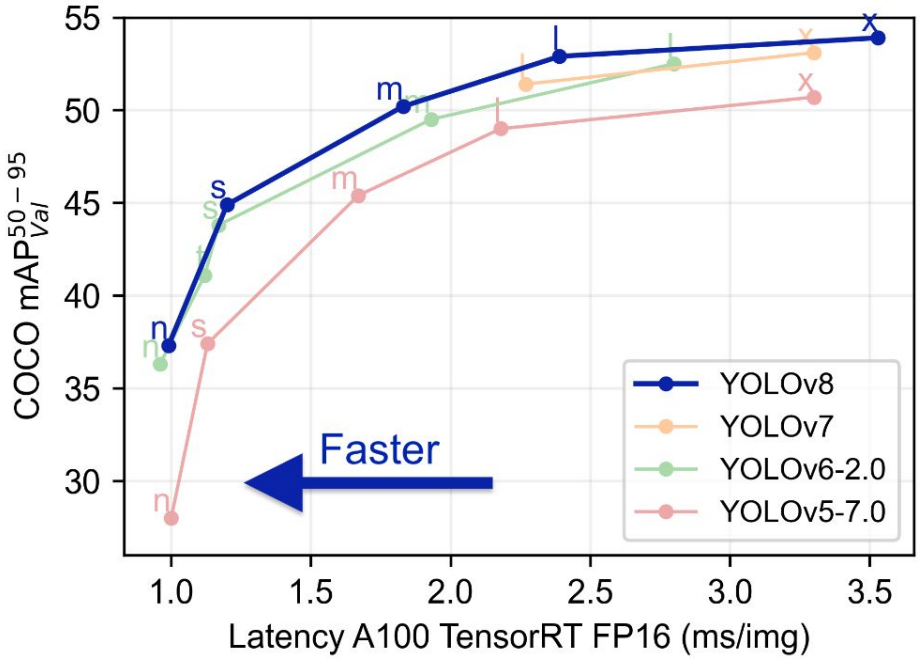
tracking | machine-learning | deep-learning | hub | pytorch | yolo | image-classification | object-detection | obb | pose | instance-segmentation | yolov3 | yolov5 | ultralytics | yolov8 | yolo-world

Readme
AGPL-3.0 license
Code of conduct
Security policy
Cite this repository
Activity
Custom properties
20.1k stars
131 watching
4k forks
Report repository

Releases 1

v8.1.0 - YOLOv8 Oriented Bo... Latest
on Jan 10


COCO mAP$^{50-95}_{Val}$
55
50
45
40
35
30
0
20
40
60
80
Parameters (M)
Smaller
YOLOv8
YOLOv7
YOLOv6-2.0
YOLOv5-7.0


COCO mAP$^{50-95}_{Val}$
55
50
45
40
35
30
1.0
1.5
2.0
2.5
3.0
3.5
Latency A100 TensorRT FP16 (ms/img)
Faster
YOLOv8
YOLOv7
YOLOv6-2.0
YOLOv5-7.0

▼ Detection (COCO)

See Detection Docs for usage examples with these models trained on COCO, which include 80 pre-trained classes.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset.
  Reproduce by `yolo val detect data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo val detect data=coco.yaml batch=1 device=0|cpu`

▼ Segmentation (COCO)

See Segmentation Docs for usage examples with these models trained on COCO-Seg, which include 80 pre-trained classes.

Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-seg	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s-seg	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m-seg	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l-seg	640	52.3	42.6	572.4	2.79	46.0	220.5
YOLOv8x-seg	640	53.4	43.4	712.1	4.02	71.8	344.1

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset.
  Reproduce by `yolo val segment data=coco-seg.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo val segment data=coco-seg.yaml batch=1 device=0|cpu`

Train

Train YOLOv8n on the COCO128 dataset for 100 epochs at image size 640. For a full list of available arguments see the Configuration page.

Example

Python CLI

```python
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.yaml')  # build a new model from YAML
model = YOLO('yolov8n.pt')  # load a pretrained model (recommended for training)
model = YOLO('yolov8n.yaml').load('yolov8n.pt')  # build from YAML and transfer weights

# Train the model
results = model.train(data='coco128.yaml', epochs=100, imgsz=640)
```

Val

Validate trained YOLOv8n model accuracy on the COCO128 dataset. No argument need to passed as the `model` retains it's training `data` and arguments as model attributes.

Example

Python CLI

```python
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.pt')  # load an official model
model = YOLO('path/to/best.pt')  # load a custom model

# Validate the model
metrics = model.val()  # no arguments needed, dataset and settings remembered
metrics.box.map    # map50-95
metrics.box.map50  # map50
metrics.box.map75  # map75
metrics.box.maps   # a list contains map50-95 of each category
```

Predict

Use a trained YOLOv8n model to run predictions on images.

Example

Python CLI

```python
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.pt')  # load an official model
model = YOLO('path/to/best.pt')  # load a custom model

# Predict with the model
results = model('https://ultralytics.com/images/bus.jpg')  # predict on an image
```

See full `predict` mode details in the Predict page.

Export

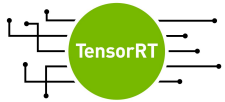
Export a YOLOv8n model to a different format like ONNX, CoreML, etc.

Example

Python CLI

```python
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.pt')  # load an official model
model = YOLO('path/to/best.pt')  # load a custom trained model

# Export the model
model.export(format='onnx')
```

Export



Export mode is used for exporting a YOLOv8 model to a format that can be used for deployment. In this mode, the model is converted to a format that can be used by other software applications or hardware devices. This mode is useful when deploying the model to production environments.

Export

Export to ONNX | **Export to TensorRT**

Export an official YOLOv8n model to TensorRT on `device=0` for acceleration on CUDA devices.

```
from ultralytics import YOLO

model = YOLO('yolov8n.pt')
model.export(format='onnx', device=0)
```

Export Examples

Python

```python
from ultralytics import YOLO

# Create a new YOLO model from scratch
model = YOLO('yolov8n.yaml')

# Load a pretrained YOLO model (recommended for training)
model = YOLO('yolov8n.pt')

# Train the model using the 'coco128.yaml' dataset for 3 epochs
results = model.train(data='coco128.yaml', epochs=3)

# Evaluate the model's performance on the validation set
results = model.val()

# Perform object detection on an image using the model
results = model('https://ultralytics.com/images/bus.jpg')

# Export the model to ONNX format
success = model.export(format='onnx')
```

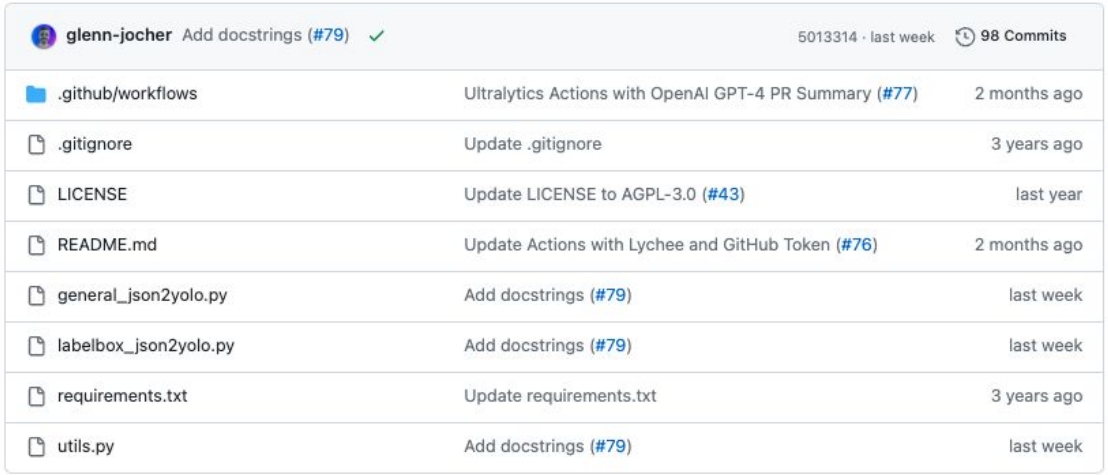
Train

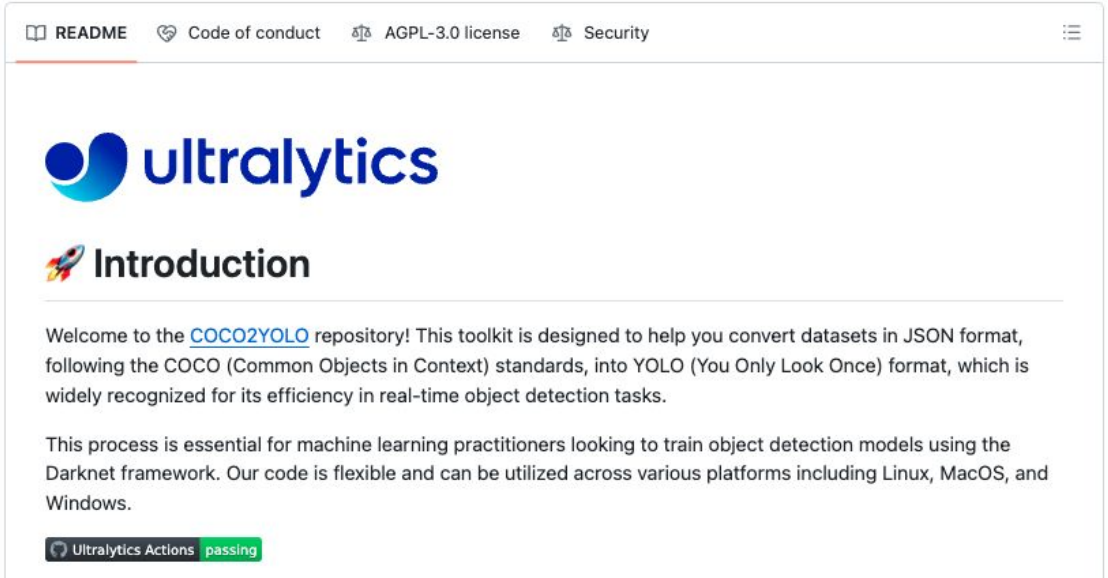
From pretrained(recommended) | **From scratch** | Resume

```
from ultralytics import YOLO

model = YOLO('yolov8n.yaml')
results = model.train(data='coco128.yaml', epochs=5)
```

Train Examples

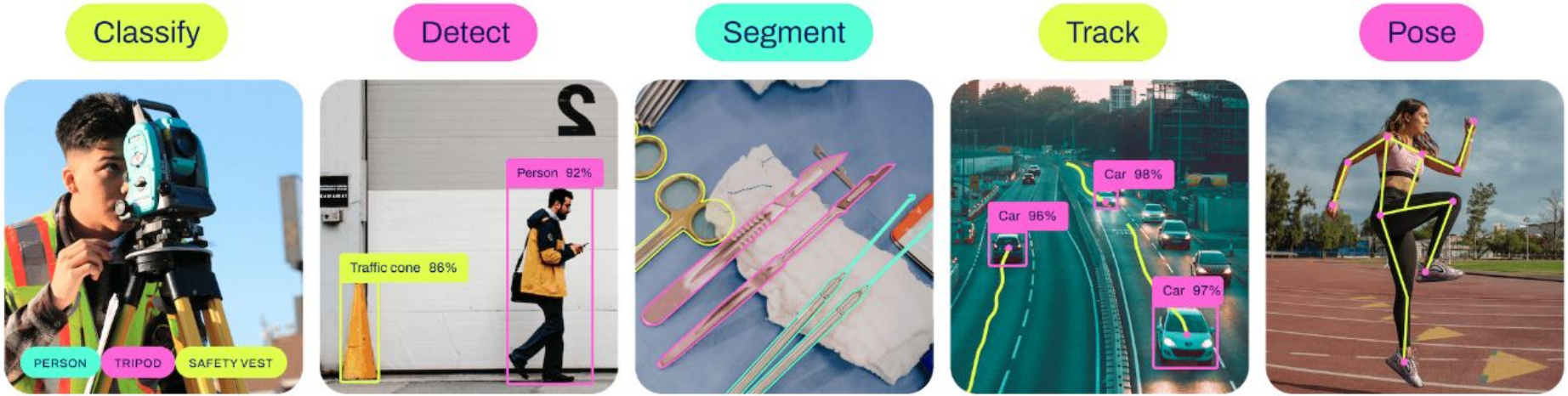
https://github.com/ultralytics/JSON2YOLO





Models

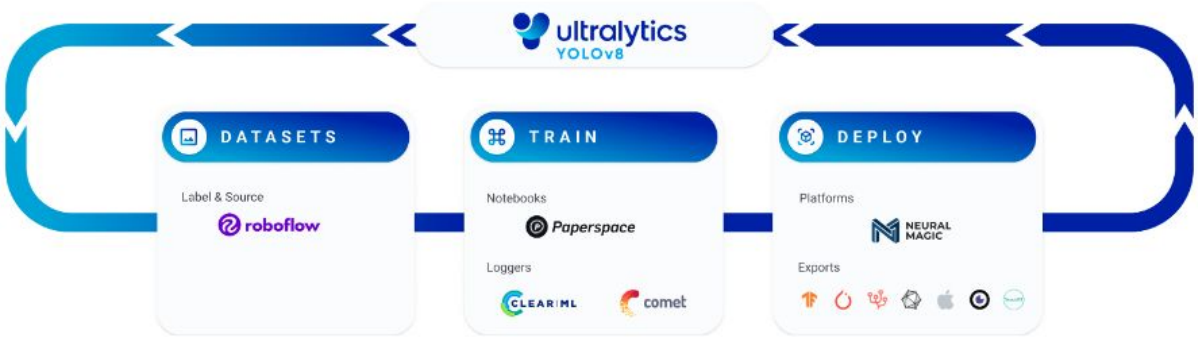
YOLOv8 Detect, Segment and Pose models pretrained on the COCO dataset are available here, as well as YOLOv8 Classify models pretrained on the ImageNet dataset. Track mode is available for all Detect, Segment and Pose models.



All Models download automatically from the latest Ultralytics release on first use.

Integrations

Our key integrations with leading AI platforms extend the functionality of Ultralytics' offerings, enhancing tasks like dataset labeling, training, visualization, and model management. Discover how Ultralytics, in collaboration with Roboflow, ClearML, Comet, Neural Magic and OpenVINO, can optimize your AI workflow.



Roboflow	ClearML ⭐ NEW	Comet ⭐ NEW	Neural Magic ⭐ NEW
Label and export your custom datasets directly to YOLOv8 for training with Roboflow	Automatically track, visualize and even remotely train YOLOv8 using ClearML (open-source!)	Free forever, Comet lets you save YOLOv8 models, resume training, and interactively visualize and debug predictions	Run YOLOv8 inference up to 6x faster with Neural Magic DeepSparse

Featured Models

Here are some of the key models supported:

1. **YOLOv3**: The third iteration of the YOLO model family, originally by Joseph Redmon, known for its efficient real-time object detection capabilities.
2. **YOLOv4**: A darknet-native update to YOLOv3, released by Alexey Bochkovskiy in 2020.
3. **YOLOv5**: An improved version of the YOLO architecture by Ultralytics, offering better performance and speed trade-offs compared to previous versions.
4. **YOLOv6**: Released by Meituan in 2022, and in use in many of the company's autonomous delivery robots.
5. **YOLOv7**: Updated YOLO models released in 2022 by the authors of YOLOv4.
6. **YOLOv8 NEW** 🚀: The latest version of the YOLO family, featuring enhanced capabilities such as instance segmentation, pose/keypoints estimation, and classification.
7. **YOLOv9**: An experimental model trained on the Ultralytics YOLOv5 codebase implementing Programmable Gradient Information (PGI).
8. **Segment Anything Model (SAM)**: Meta's Segment Anything Model (SAM).
9. **Mobile Segment Anything Model (MobileSAM)**: MobileSAM for mobile applications, by Kyung Hee University.
10. **Fast Segment Anything Model (FastSAM)**: FastSAM by Image & Video Analysis Group, Institute of Automation, Chinese Academy of Sciences.
11. **YOLO-NAS**: YOLO Neural Architecture Search (NAS) Models.
12. **Realtime Detection Transformers (RT-DETR)**: Baidu's PaddlePaddle Realtime Detection Transformer (RT-DETR) models.

Files

main

Go to file

- cfg
  - datasets
  - models
    - rt-detr
    - v3
    - v5
    - v6
    - v8
      - yolov8-cls-resnet101.yaml
      - yolov8-cls-resnet50.yaml
      - yolov8-cls.yaml
      - yolov8-ghost-p2.yaml
      - yolov8-ghost-p6.yaml
      - yolov8-ghost.yaml
      - yolov8-obb.yaml
      - yolov8-p2.yaml
      - yolov8-p6.yaml
      - yolov8-pose-p6.yaml
      - yolov8-pose.yaml
      - yolov8-rtdetr.yaml
      - yolov8-seg-p6.yaml
      - yolov8-seg.yaml
      - yolov8-world-t2i.yaml
      - yolov8-world.yaml
      - yolov8.yaml

ultralytics / ultralytics / cfg / models / v8 / **yolov8.yaml**

glenn-jocher YAML reformat (#7669)

Code | Blame | 46 lines (39 loc) · 1.84 KB

```yaml
# Ultralytics YOLO 🚀, AGPL-3.0 license
# YOLOv8 object detection model with P3-P5 outputs. For Usage examples see https://docs.ultralytics.com/tasks/detect

# Parameters
nc: 80 # number of classes
scales: # model compound scaling constants, i.e. 'model=yolov8n.yaml' will
  # [depth, width, max_channels]
  n: [0.33, 0.25, 1024] # YOLOv8n summary: 225 layers,  3157200 parameters,
  s: [0.33, 0.50, 1024] # YOLOv8s summary: 225 layers, 11166560 parameters,
  m: [0.67, 0.75, 768] # YOLOv8m summary: 295 layers, 25902640 parameters,
  l: [1.00, 1.00, 512] # YOLOv8l summary: 365 layers, 43691520 parameters,
  x: [1.00, 1.25, 512] # YOLOv8x summary: 365 layers, 68229648 parameters,

# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 3, C2f, [128, True]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 6, C2f, [256, True]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 6, C2f, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 3, C2f, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9

# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 3, C2f, [512]] # 12

  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  - [-1, 3, C2f, [256]] # 15 (P3/8-small)

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 12], 1, Concat, [1]] # cat head P4
  - [-1, 3, C2f, [512]] # 18 (P4/16-medium)

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 9], 1, Concat, [1]] # cat head P5
  - [-1, 3, C2f, [1024]] # 21 (P5/32-large)

  - [[15, 18, 21], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

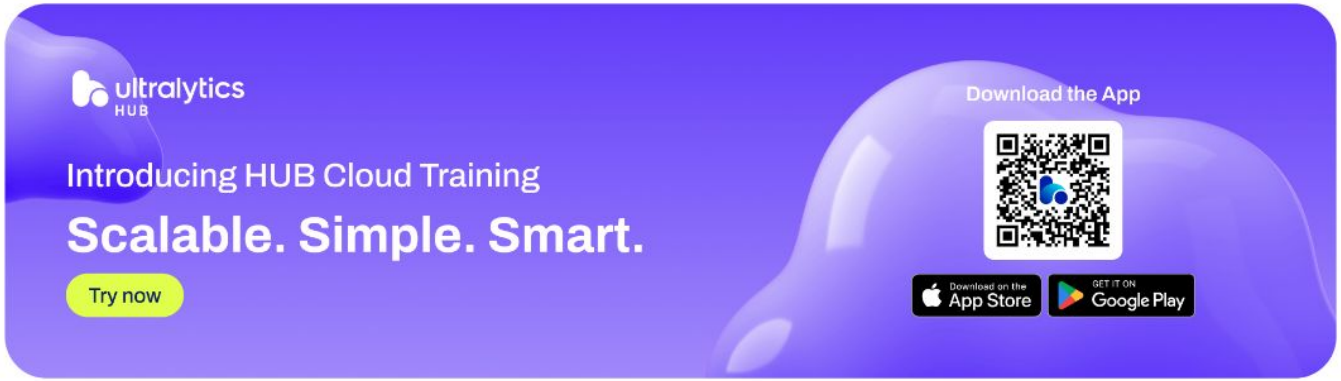


```yaml
# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 3, C2f, [128, True]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 6, C2f, [256, True]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 6, C2f, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 3, C2f, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9

# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 3, C2f, [512]] # 12

  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  - [-1, 3, C2f, [256]] # 15 (P3/8-small)

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 12], 1, Concat, [1]] # cat head P4
  - [-1, 3, C2f, [512]] # 18 (P4/16-medium)

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 9], 1, Concat, [1]] # cat head P5
  - [-1, 3, C2f, [1024]] # 21 (P5/32-large)

  - [[15, 18, 21], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

Name	Last commit message	Last commit date
..		
Argoverse.yaml	ultralytics 8.1.4 RTDETR TensorBoard graph visualization fix (#7725)	2 months ago
DOTAv1.5.yaml	YAML reformat (#7669)	2 months ago
DOTAv1.yaml	YAML reformat (#7669)	2 months ago
GlobalWheat2020.yaml	YAML reformat (#7669)	2 months ago
ImageNet.yaml	YAML reformat (#7669)	2 months ago
Objects365.yaml	YAML reformat (#7669)	2 months ago
SKU-110K.yaml	YAML reformat (#7669)	2 months ago
VOC.yaml	YAML reformat (#7669)	2 months ago
VisDrone.yaml	YAML reformat (#7669)	2 months ago
carparts-seg.yaml	ultralytics 8.1.6 revert 8.0.206 box ops box scaling (#7823)	2 months ago
coco-pose.yaml	YAML reformat (#7669)	2 months ago
coco.yaml	YAML reformat (#7669)	2 months ago
coco128-seg.yaml	YAML reformat (#7669)	2 months ago
coco128.yaml	YAML reformat (#7669)	2 months ago
coco8-pose.yaml	YAML reformat (#7669)	2 months ago
coco8-seg.yaml	YAML reformat (#7669)	2 months ago
coco8.yaml	YAML reformat (#7669)	2 months ago
crack-seg.yaml	ultralytics 8.1.6 revert 8.0.206 box ops box scaling (#7823)	2 months ago

Ultralytics HUB

Experience seamless AI with Ultralytics HUB ⭐, the all-in-one solution for data visualization, YOLOv5 and YOLOv8 🚀 model training and deployment, without any coding. Transform images into actionable insights and bring your AI visions to life with ease using our cutting-edge platform and user-friendly Ultralytics App. Start your journey for **Free** now!



Contribute

We love your input! YOLOv5 and YOLOv8 would not be possible without help from our community. Please see our Contributing Guide to get started, and fill out our Survey to send us feedback on your experience. Thank you 🙏 to all our contributors!