# OBJECT DETECTION WITH YOLO

...

Michał Gilski and Boda Wen

# YOLO

- YOLO (You Only Look Once)-> object detection
- DarkNet
- Earlier detection frameworks, looked at different parts of the image multiple times at different scales and repurposed image classification technique to detect objects. This approach is slow and inefficient.
- Region-based CNNs (R-CNNs):An object detector that has candidate bounding boxes that could contain objects. These regions were then passed into a CNN for classification.
- YOLO takes entirely different approach. It looks at the entire image only once and goes through the network once and detects objects. Hence the name. It is very fast. That's the reason it has got so popular.

# Yolo history

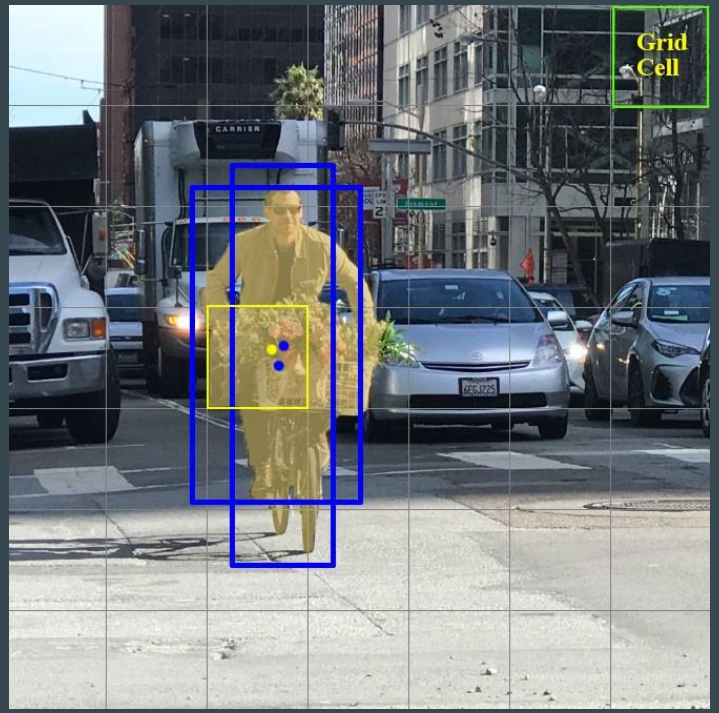
Created by Joseph Redmon and Ali Farhadi from University of Washington

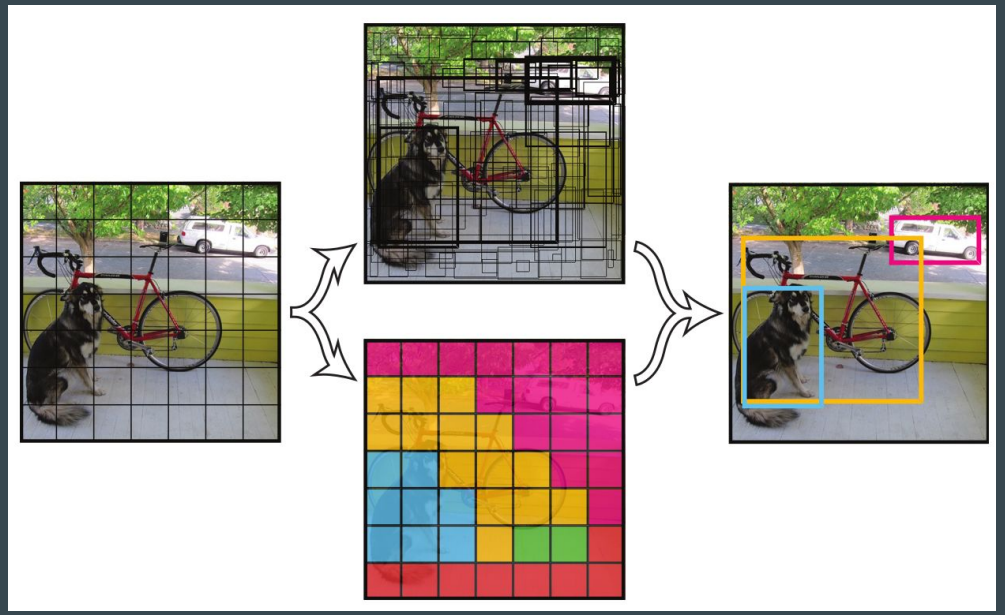
- The first YOLO - faster than any other detection approach at 55 FPS
- Fast YOLO - more shallow network, but up to 155 FPS

- YOLOv2 - **Better, Faster, Stronger** (Batch normalization, introduction of Darknet and anchor boxes, higher resolution, Word trees, etc.)
- YOLO 9000 - detects over 9000 classes

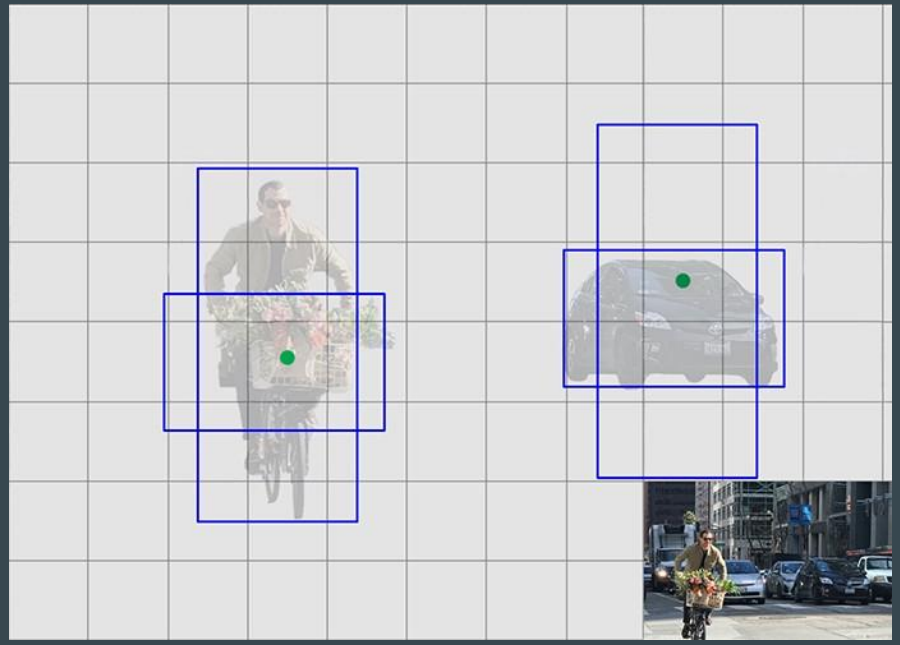
- YOLOv3 - "**We mostly took good ideas from other people**": three output layers, deeper network with shortcuts, feature extraction, etc..

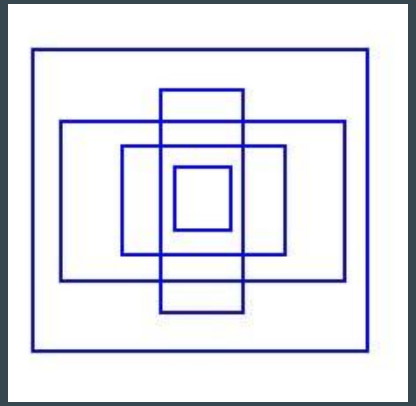
# Yolo algorithm - beginning

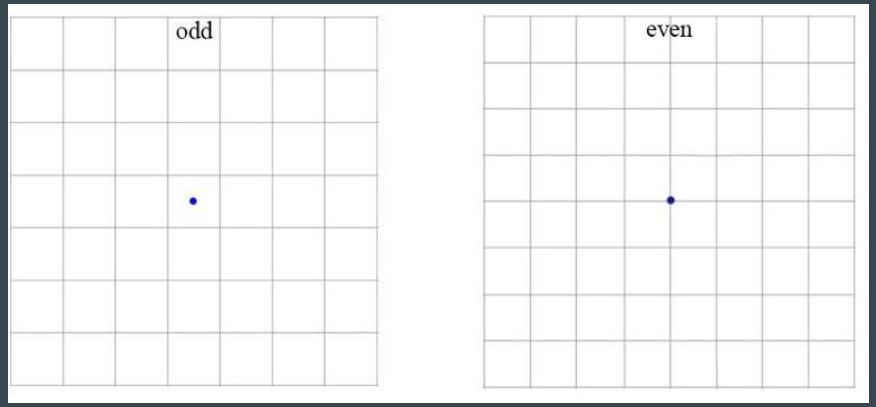


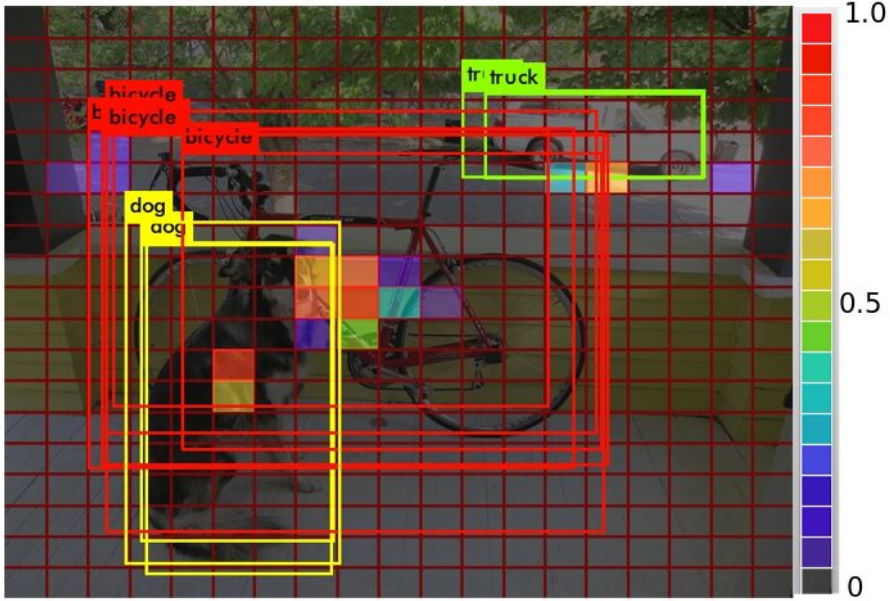
# Yolo algorithm v2 - anchors



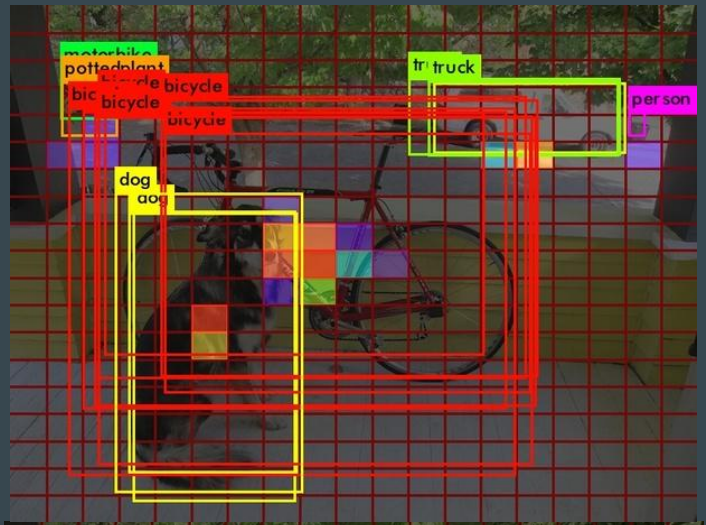


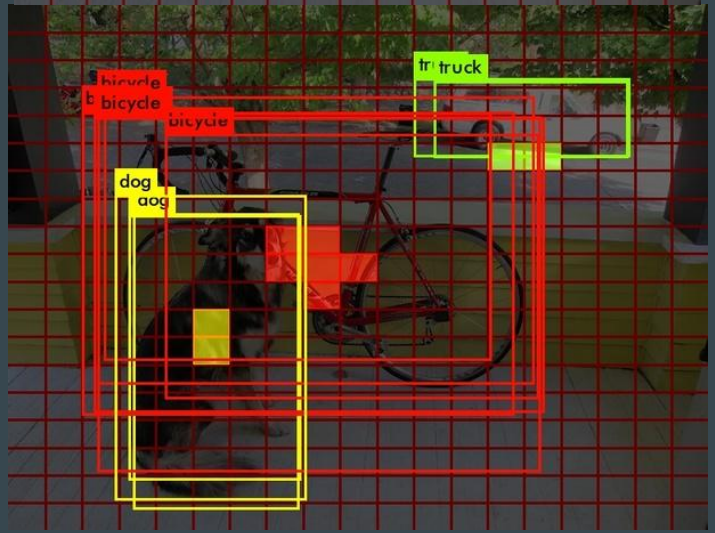


# Grid visualization

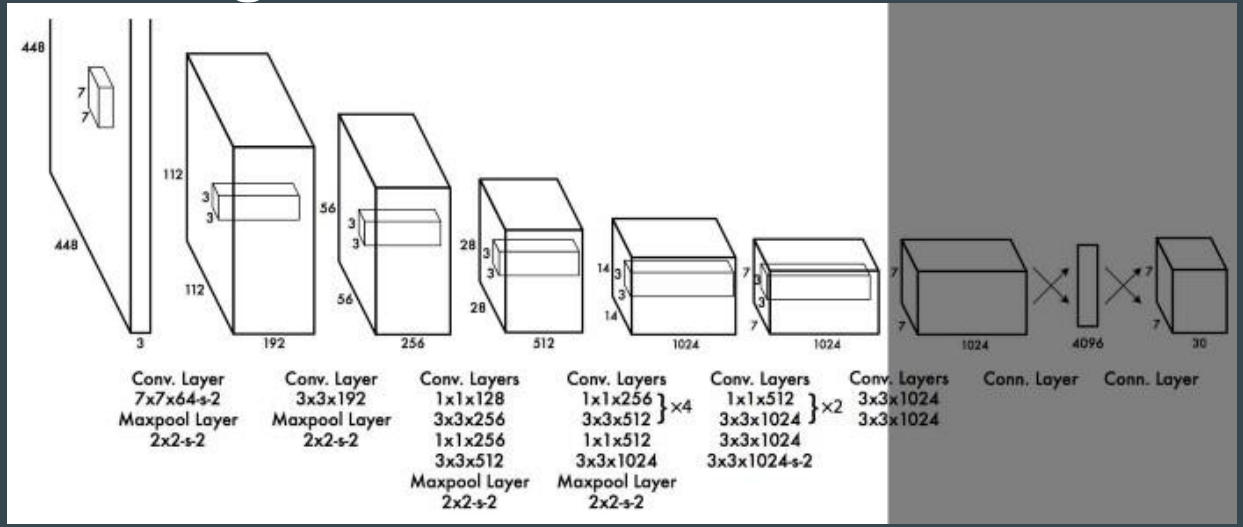



object confidence



# Yolo algorithm - architecture - from 19 to 53




| | 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.**

```
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky
```

```
[convolutional]
size=1
stride=1
pad=1
filters=255
activation=linear
```

```
[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,
33,23,  30,61,  62,45,
59,119,  116,90,
156,198,  373,326
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
```

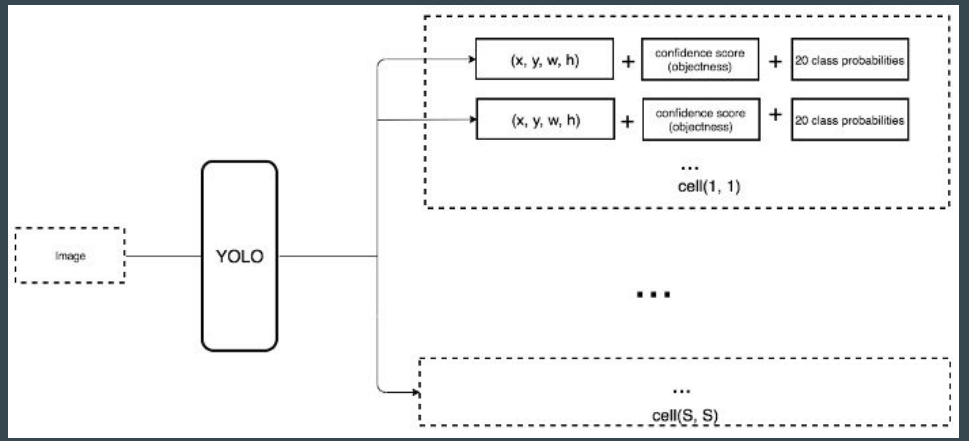
# Bounding box parameters

The network has 3 output layers (13x13, 26x26, 52x52)

Each grid element has 3 bounding box proposals

The output layers of the network return 85 values:

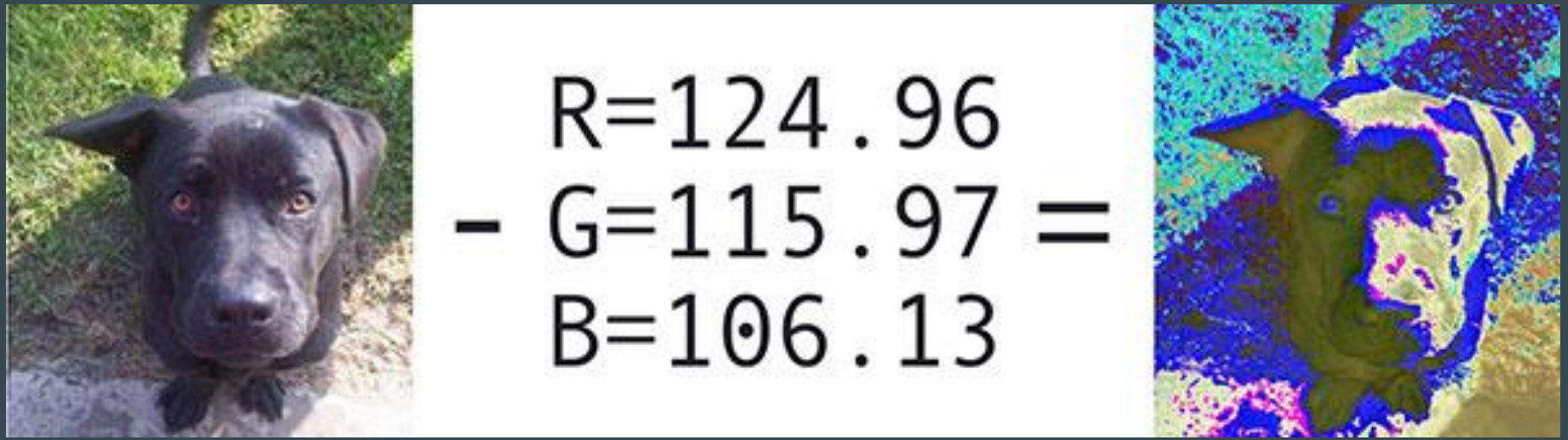
- 4 bounding box coordinates
- 1 objectness (used for training)
- 80 confidences for the 80 classes



# Preprocessing

- blobFromImage:
  - scale:  scale images, pixels divide by 255 =>in range 0...1
  - size
  - swapRB: OpenCV: BGR, normal image: RGB
  - mean: mean subtraction values => normalization

# Blobs



# FRAMEWORKS

- YOLOv3
- Python
- OpenCV
- COCO database: (common objects in context) from Microsoft. It is capable of detecting 80 common objects
- Test database: VOC 2012

# Yolo implementation

- Setting parameters: confidence and threshold
- Preprocessing the images & frames with OpenCV
- Using Yolo handle images/ videos
- Implementing the evaluation class
- Evaluating results

# Parameters

- **Confidence**: Minimum probability to filter weak detections

- **Threshold**: ignore bounding boxes that significantly overlap each other.

# Confidence

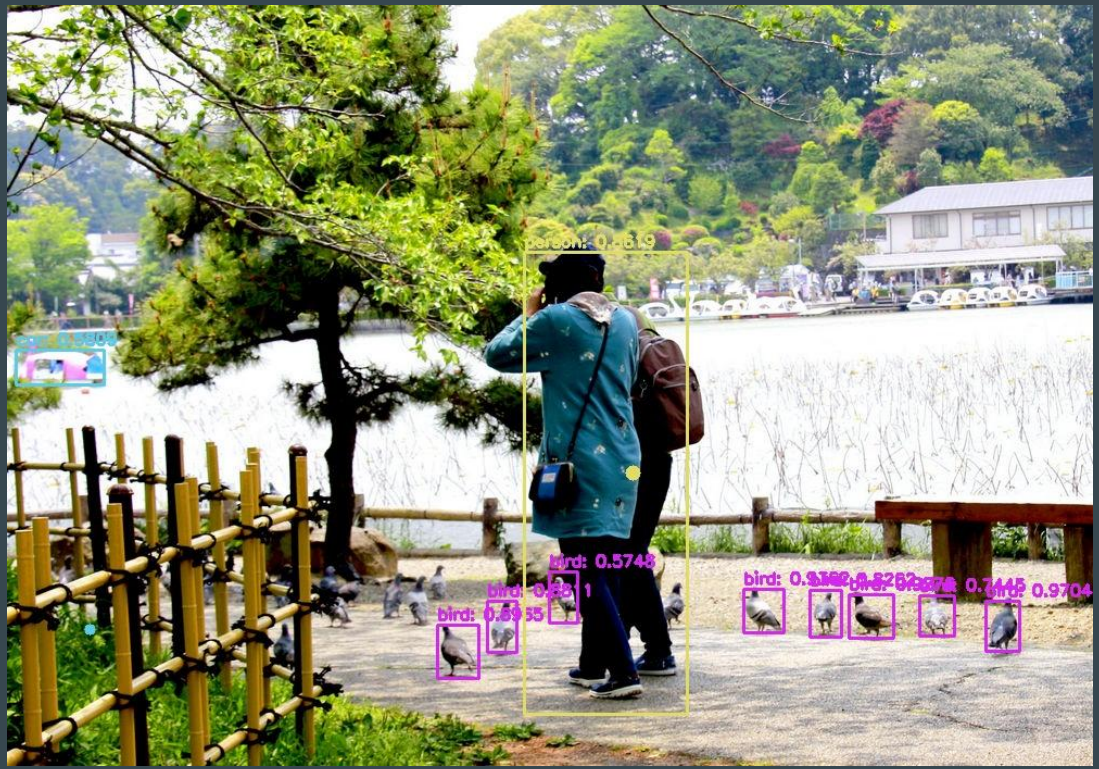
Confidence = 0.5

# Confidence

Confidence = 0.7

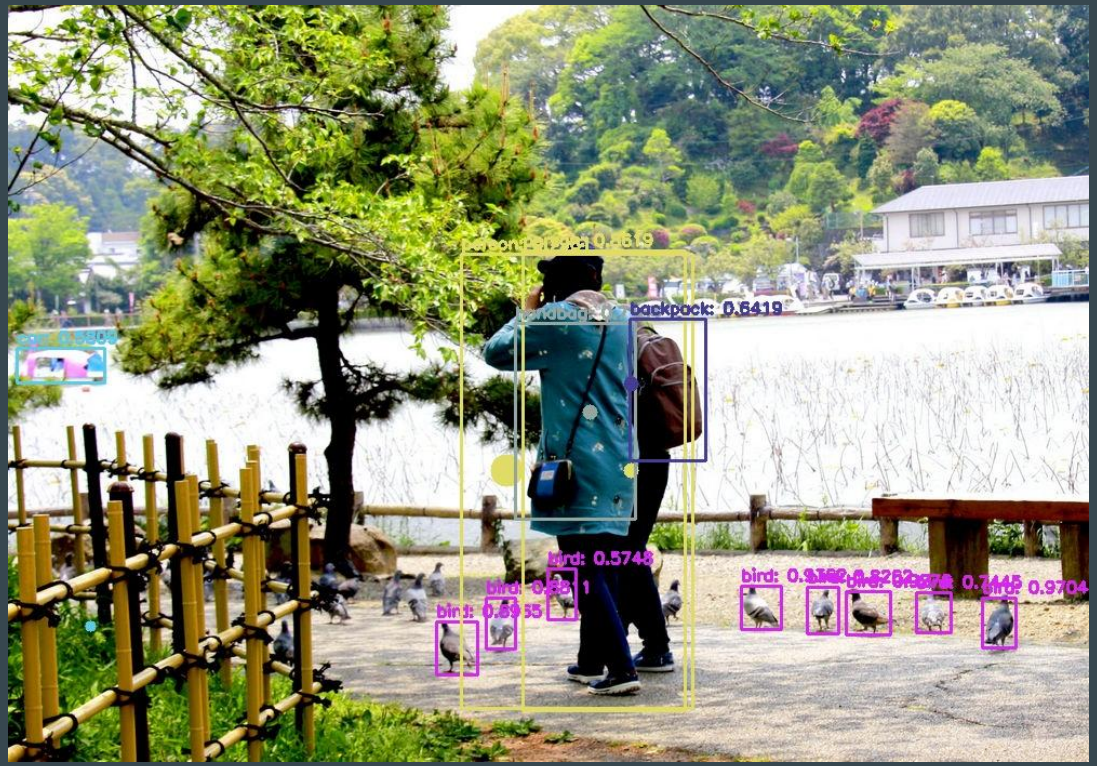
# Threshold

Threshold = 0.1



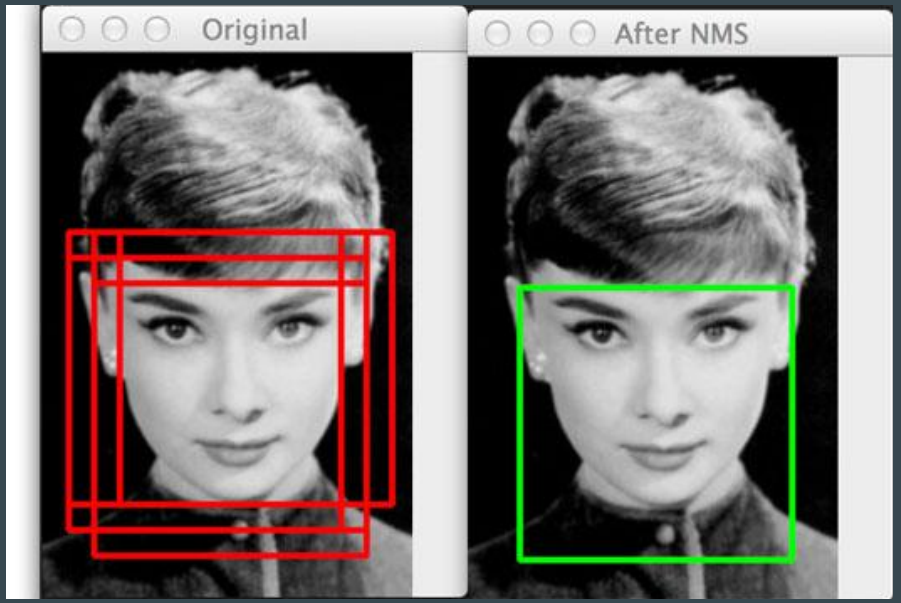
# Threshold

Threshold = 0.7

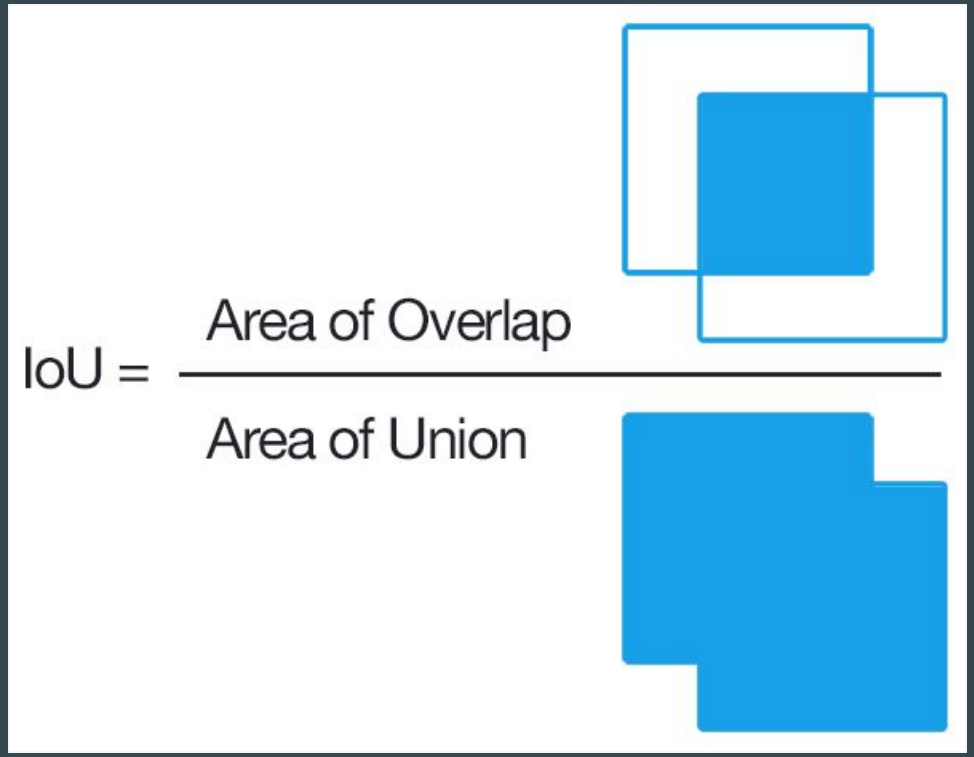


# Non-Maximum Suppression

- Multiple bounding boxes
- Need to apply Non-Maximum Suppression
- Threshold high: multi-box
- Threshold low: one box



# IOU: Intersection over Union



# ACCURACY

- $Accuracy = \left( \frac{\sum i\left(IOU(x_i.t, x_i.p)\right)}{|x|} + \left(\frac{\sum i(Label(x_i.t, x_i.p))}{|x|}\right) + Diff(X.t, X.p) \right) / 3$
- $Label(x, y) = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}$
- $Diff(x, y) = \left| \frac{|x| - |y|}{\max(|x|, |y|)} \right|$
- **X.t = actual classes**
- **X.p = prediction classes**

# Test dataset building

Images and bounding boxes from VOC 2012:

General conditions for dataset:

- Image contains only labels that are also in the COCO dataset
- Image contains more than 5 objects

Datasets:

- Tiny - 20 images
- Small - 112 images
- Medium - 205 images
- Big - 1366 images

```
test/small/2011_001625.jpg
horse 154 96 420 279
horse 131 106 374 277
person 73 112 118 269
person 108 120 128 268
person 65 109 139 224
car 444 197 500 228
;
```

# Accuracy graph

# Average Time

# Test results for best values

The best values seem to be:
confidence: 0.2 (or 0 or 0.1)
and threshold 0.6

Results:

- Tiny: 77.28%
- Small: 74.03%
- Medium: 73.87%
- Big: 73.44%

Results for base test
(conf: 0.5, thresh 0.3:

- Tiny: 65.33%
- Small: 66.63%
- Medium: 67.7%
- Big: 67.48%

Results for base test
(conf: 0.5, thresh 0.3:

- Tiny: 76.23%
- Small: 73.05%
- Medium: 72.88%
- Big: 72.38%

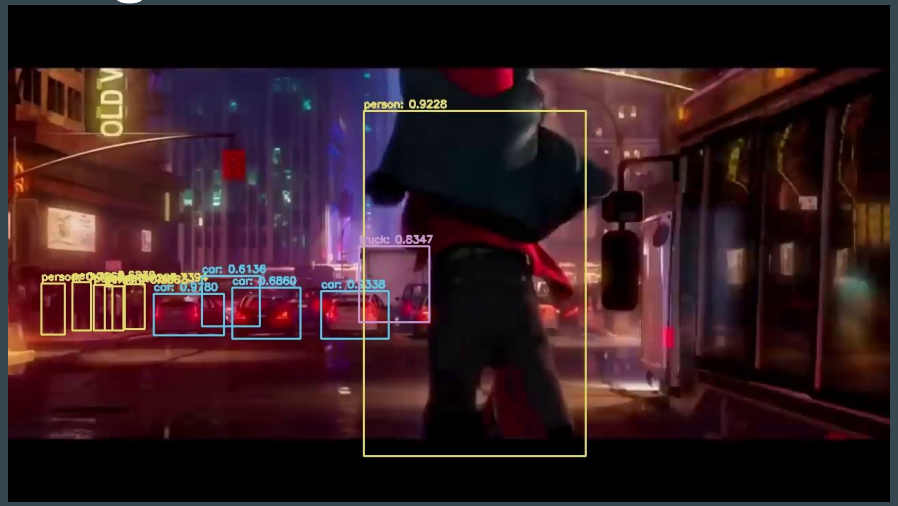
# Speed (different laptops)

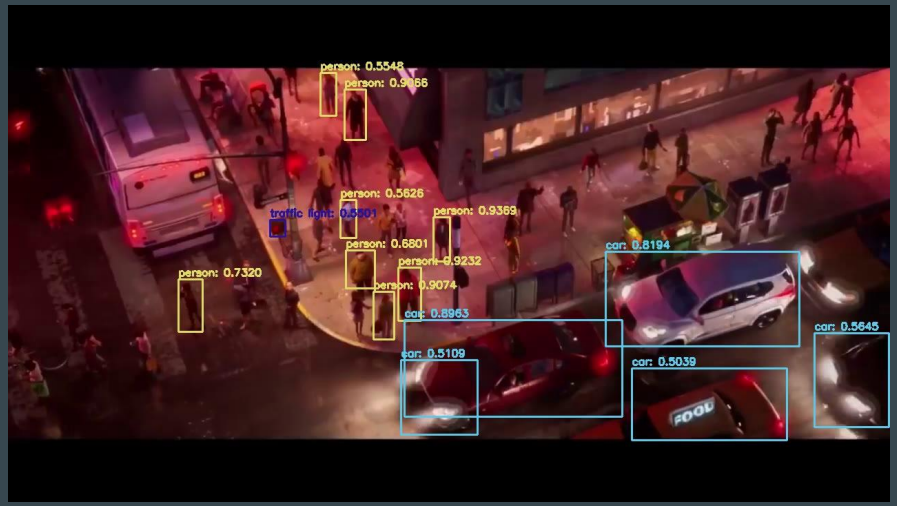
350 frames

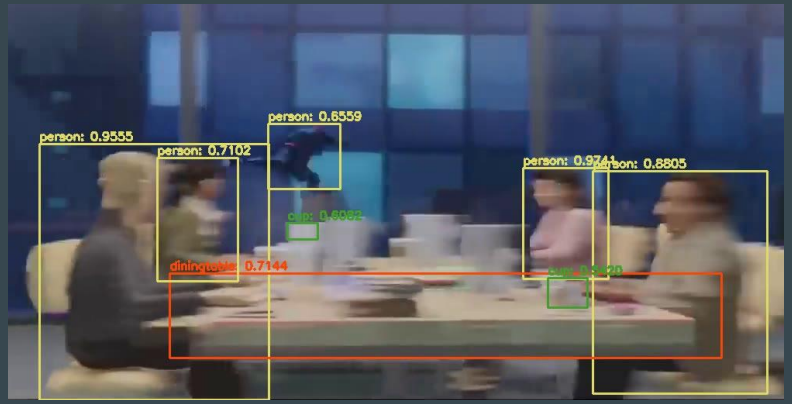
- i7 4700MQ CPU @ 2.40GHz × 8
  - Total time: 180 s
  - Average time: 0.514 seconds per frame
- 2.7 GHz Intel Core i5
  - Total time:  290s
  - Average time: 0.829 seconds per frame
- i5 4200U CPU @ 1.6GHZ x 4
  - Total time: 339 s
  - Average time: 0.96 seconds per frame

# Demo movies

# The good

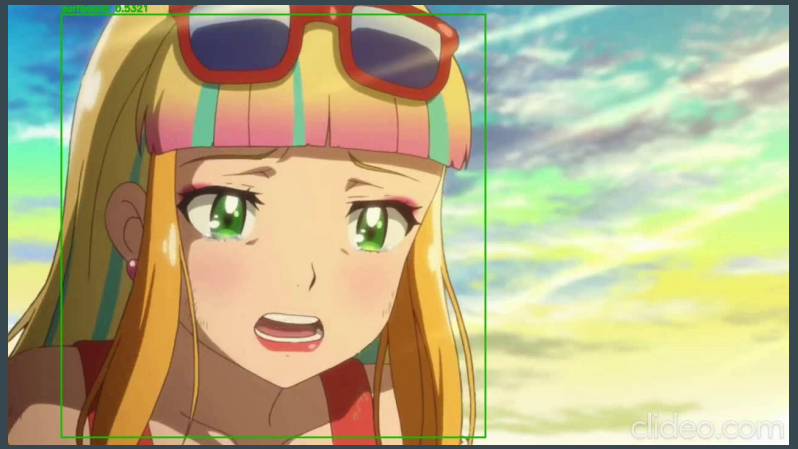

person: 0.9228
truck: 0.8347
car: 0.6136
car: 0.6860
car: 0.9780


person: 0.5548
person: 0.9166
person: 0.5626
person: 0.9369
person: 0.6801
person: 0.9232
person: 0.7320
person: 0.9074
car: 0.8963
car: 0.8194
car: 0.5109
car: 0.5039
car: 0.5645


person: 0.9555
person: 0.7102
person: 0.6559
person: 0.9741
person: 0.8805
dining table: 0.7144

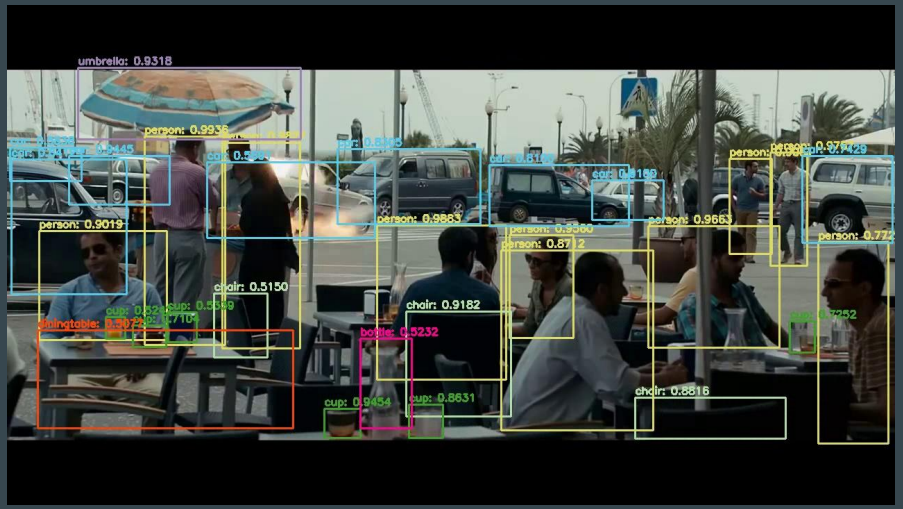

person: 0.9920
person: 0.8624
person: 0.6730

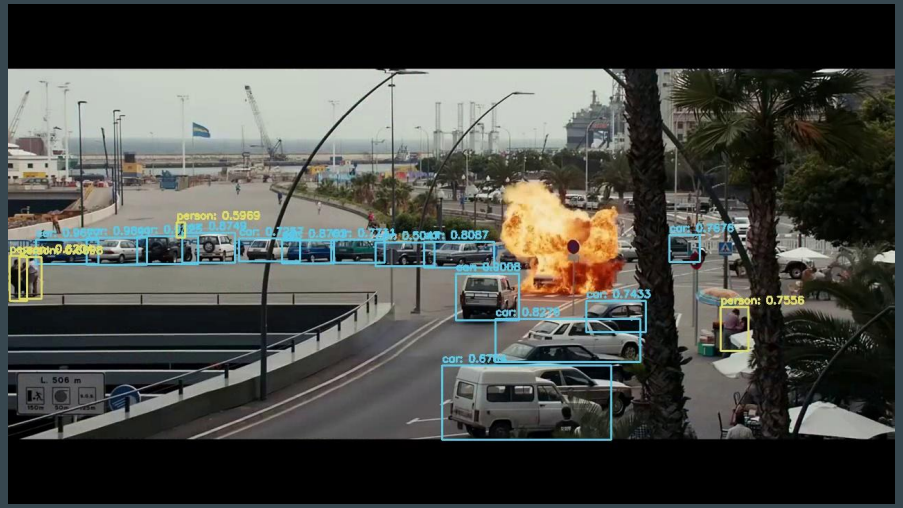
# The bad

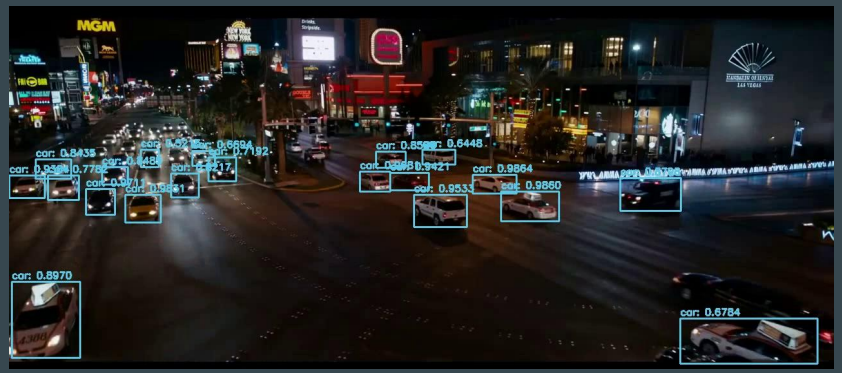



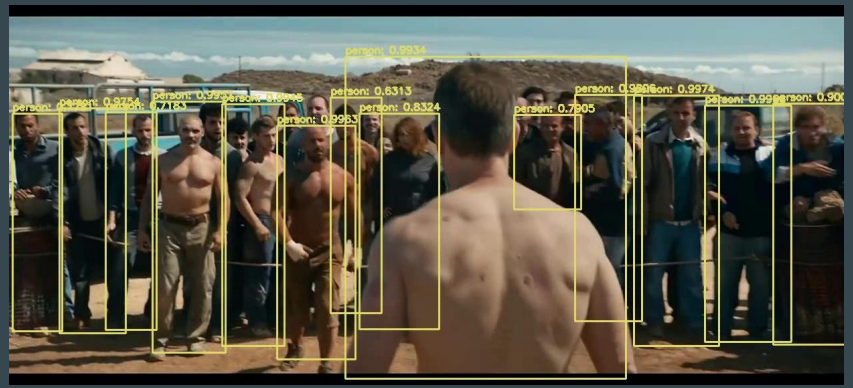


# The many

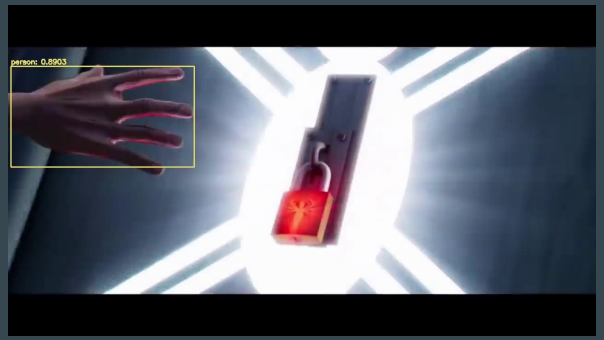

umbrella: 0.9318
person: 0.9019
chair: 0.5150
chair: 0.9182
bottle: 0.5232
cup: 0.9454
cup: 0.8631
chair: 0.8816

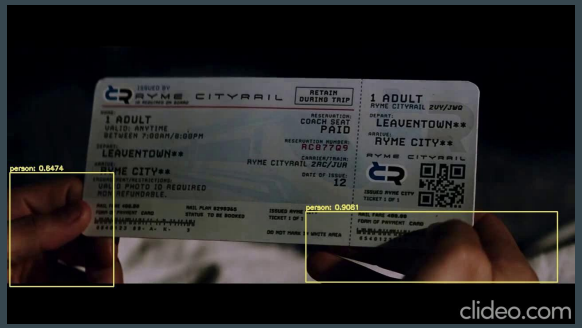

person: 0.5969
person: 0.8087
car: 0.7433
person: 0.7556


MGM
car: 0.8435
car: 0.8970
car: 0.6784
car: 0.9864
car: 0.9860

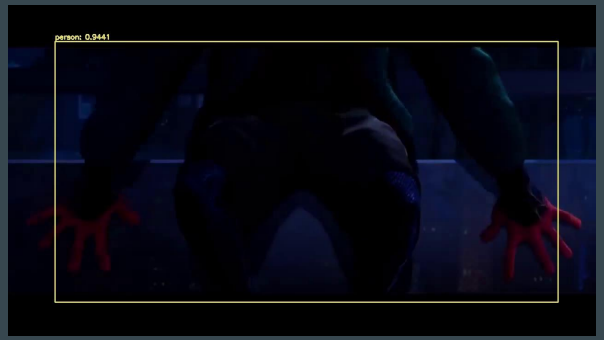

person: 0.6313
person: 0.8324
person: 0.7805
person: 0.9974

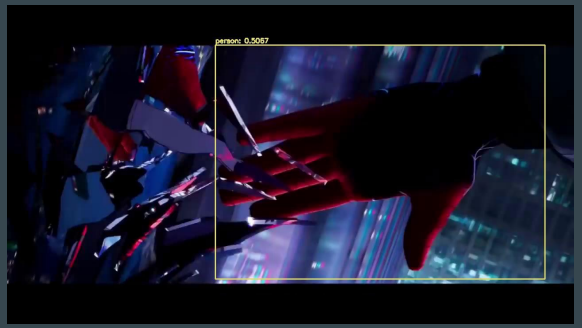
# Human detection

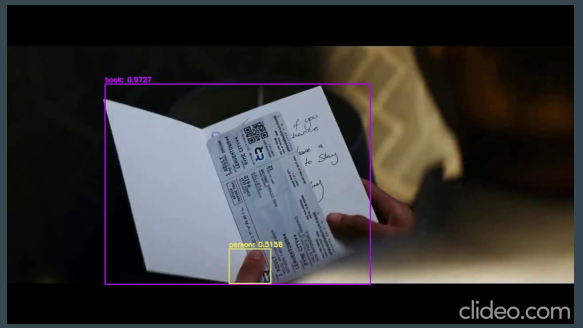


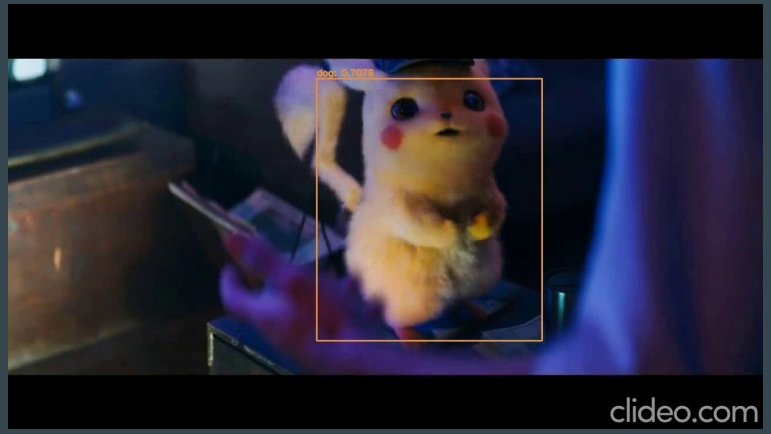


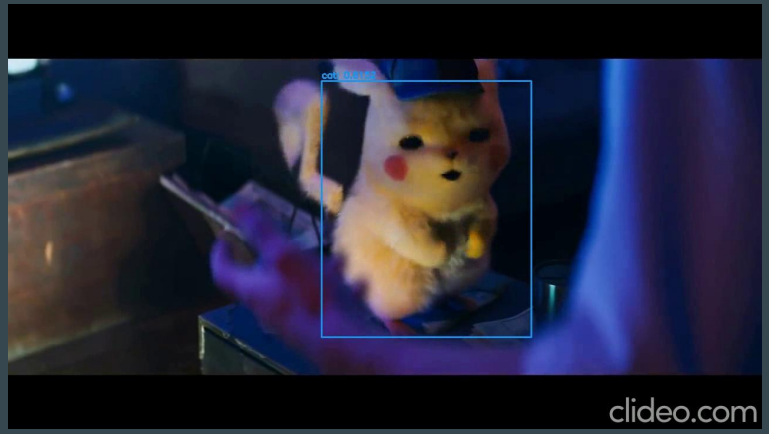


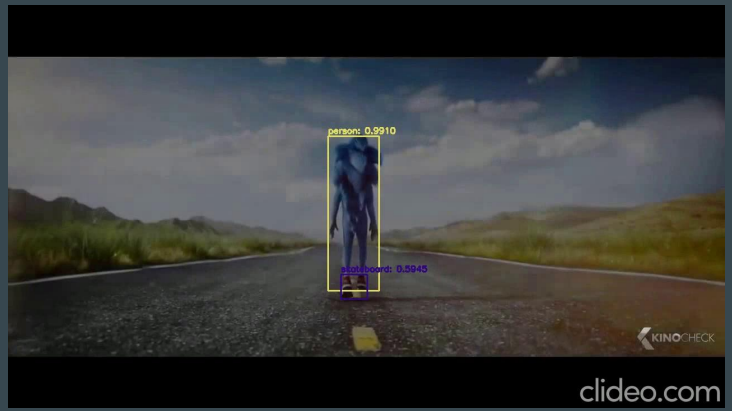
# The fur of  Pikachu





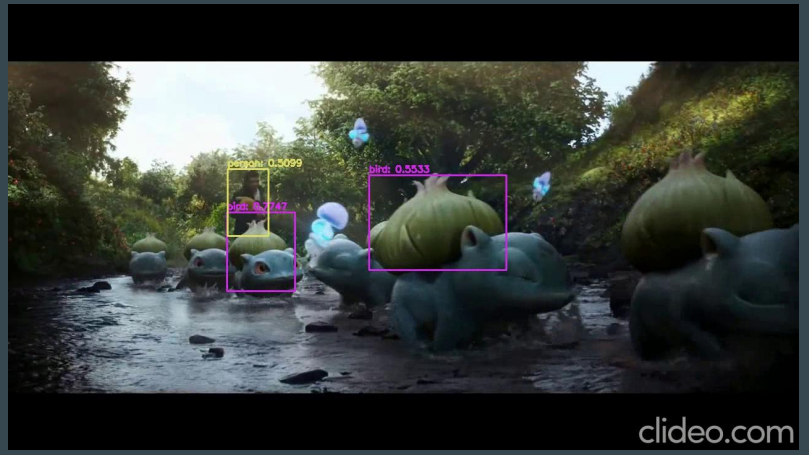
# The pose of Sonic






# Pokemon


clideo.com

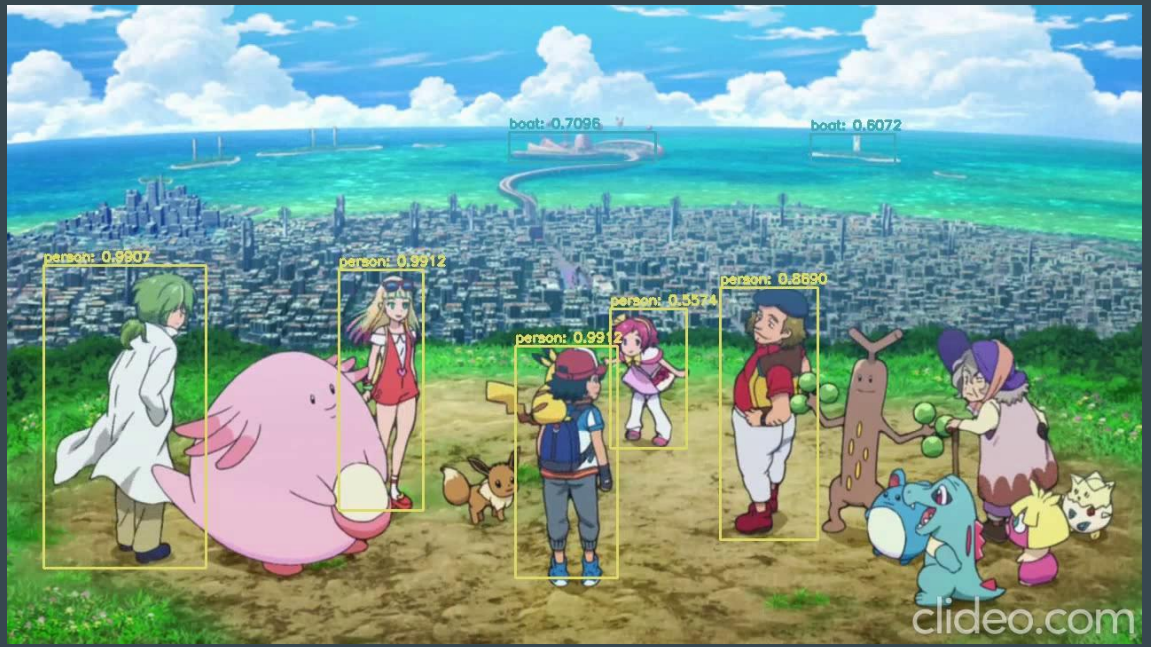

person: 0.5099
bird: 0.5533
clideo.com


teddy bear: 0.6525
clideo.com


person: 0.8060
clideo.com

# The air and sea




# The invisible skateboard


# Bibliography


- https://pjreddie.com/darknet/yolo/
- https://pjreddie.com/media/files/papers/yolo.pdf
- https://arxiv.org/pdf/1612.08242.pdf
- https://pjreddie.com/media/files/papers/YOLOv3.pdf
- https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/
- https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088
- https://github.com/xueeinstein/darknet-vis
- https://towardsdatascience.com/review-yolov2-yolo9000-you-only-look-once-object-detection-7883d2b02a65
- https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/
- http://host.robots.ox.ac.uk/pascal/VOC/voc2012/#testdata
- https://www.youtube.com - (trailers)