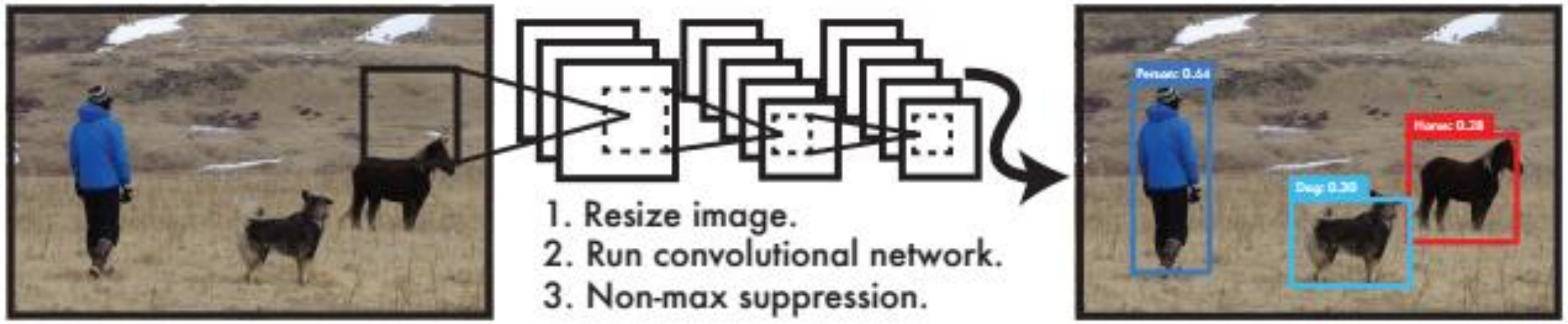
# You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon
University of Washington
pjreddie@cs.washington.edu

Santosh Divvala
Allen Institute for Artificial Intelligence
santoshd@allenai.org

Ross Girshick
Facebook AI Research
rbg@fb.com

Ali Farhadi
University of Washington
ali@cs.washington.edu

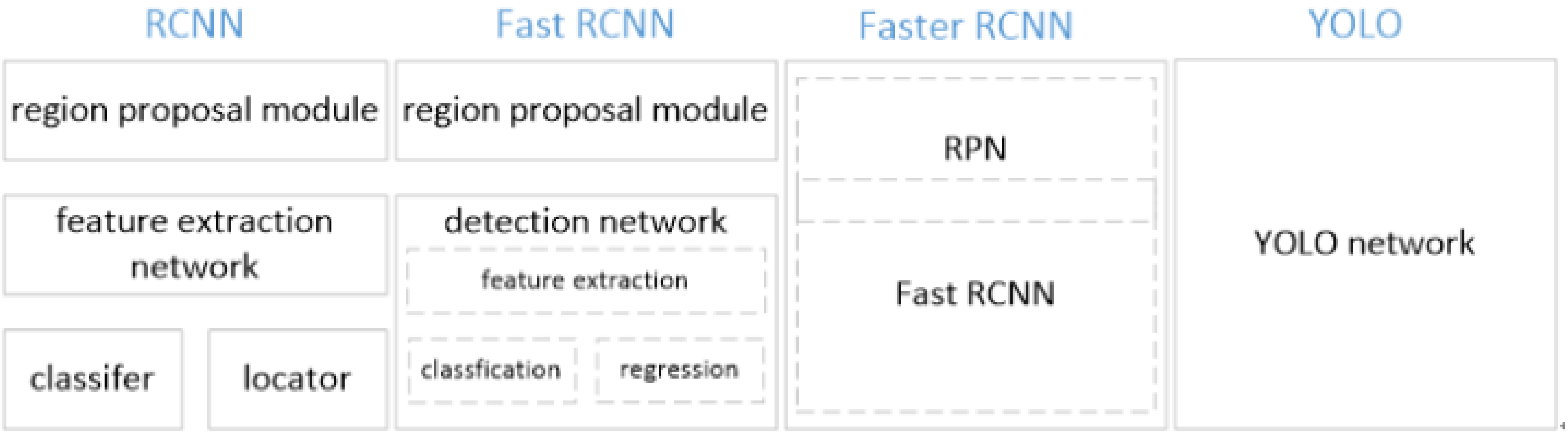


# Contents

# Main ideas

1. Convert object detection to a **regression problem** to spatially separated bounding boxes and associated class probabilities.
2. **A single neural network** predicts bounding boxes and class probabilities directly from full images in one evaluation.
3. **End to end**, extremely fast.

# Object detection



# Overall Structure

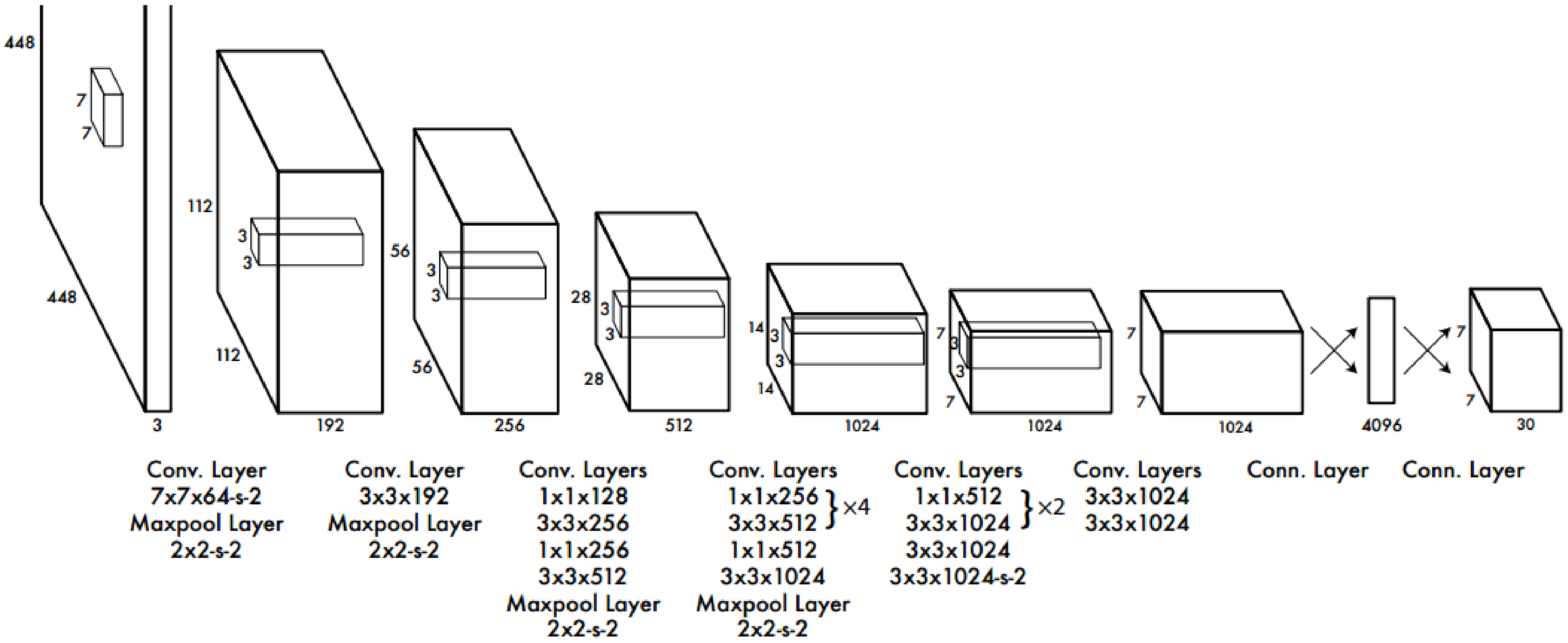



**Figure 3: The Architecture.** Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating $1 \times 1$ convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution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

# Overall Structure

1. Inspired by the GoogLeNet model for image classification
2. Use $1 \times 1$ reduction layers followed by $3 \times 3$ convolutional layers for channel reduction
3. Use a linear activation function for the final layer and all other layers use the following leaky RELU

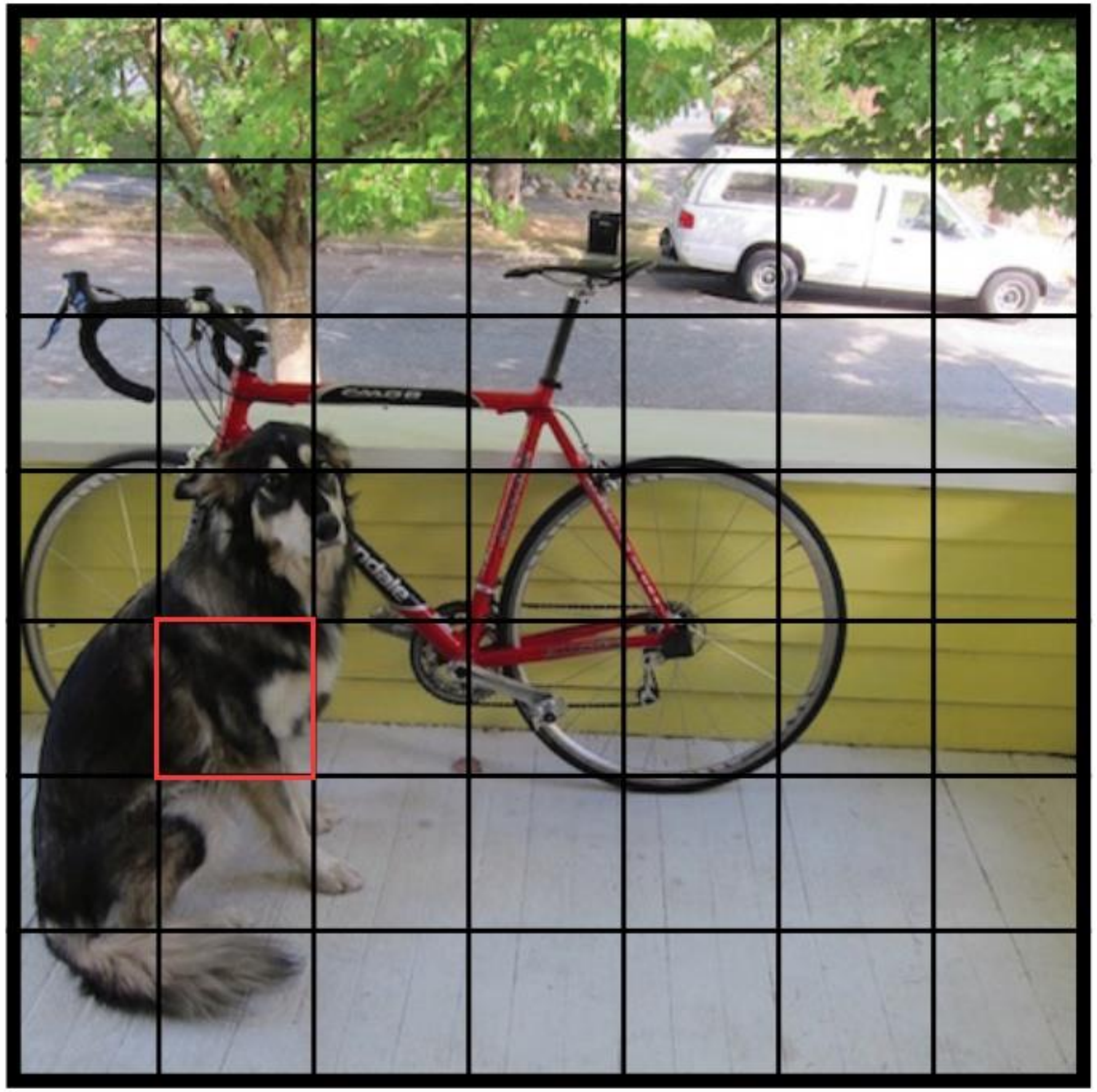
$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

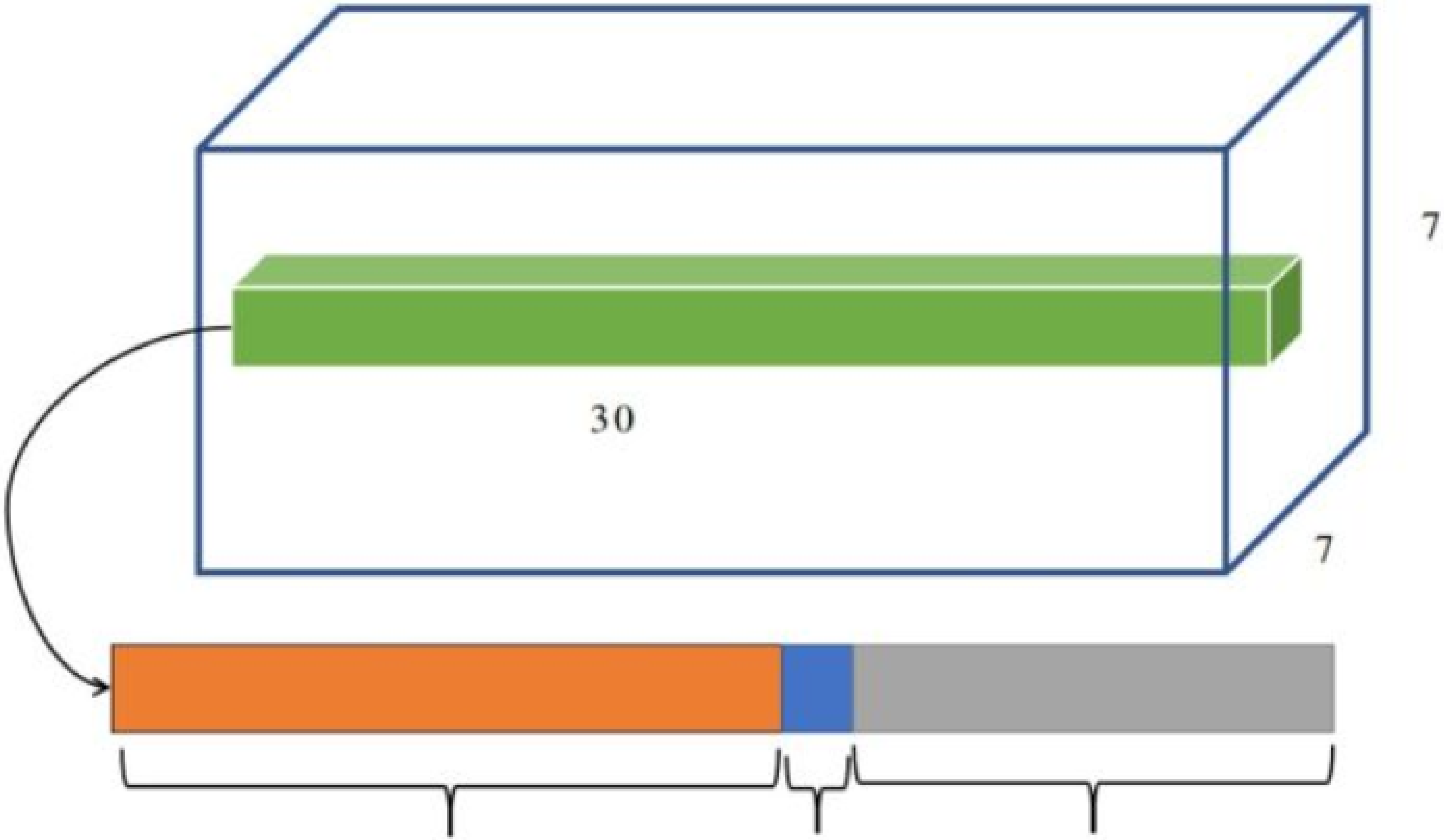
4. Build a smaller and faster version of YOLO to push the boundaries of fast object detection (9 instead of 24 convolution layers)

1. The output of the network is a tensor with size:

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

2. S: Number of grids
3. B: Number of Bbox each grid responsible for
4. C: Number of categories






20 probabilities for 20 classes

**2***4 bounding box regression (x, y, w, h)

**2** confidence score

# Loss function

1. Bounding box regression
2. Confidence regression

$\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$

$$\lambda_{\text{coord}} = 5 \text{ and } \lambda_{\text{noobj}} = .5.$$

3. Class probability regression

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

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

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

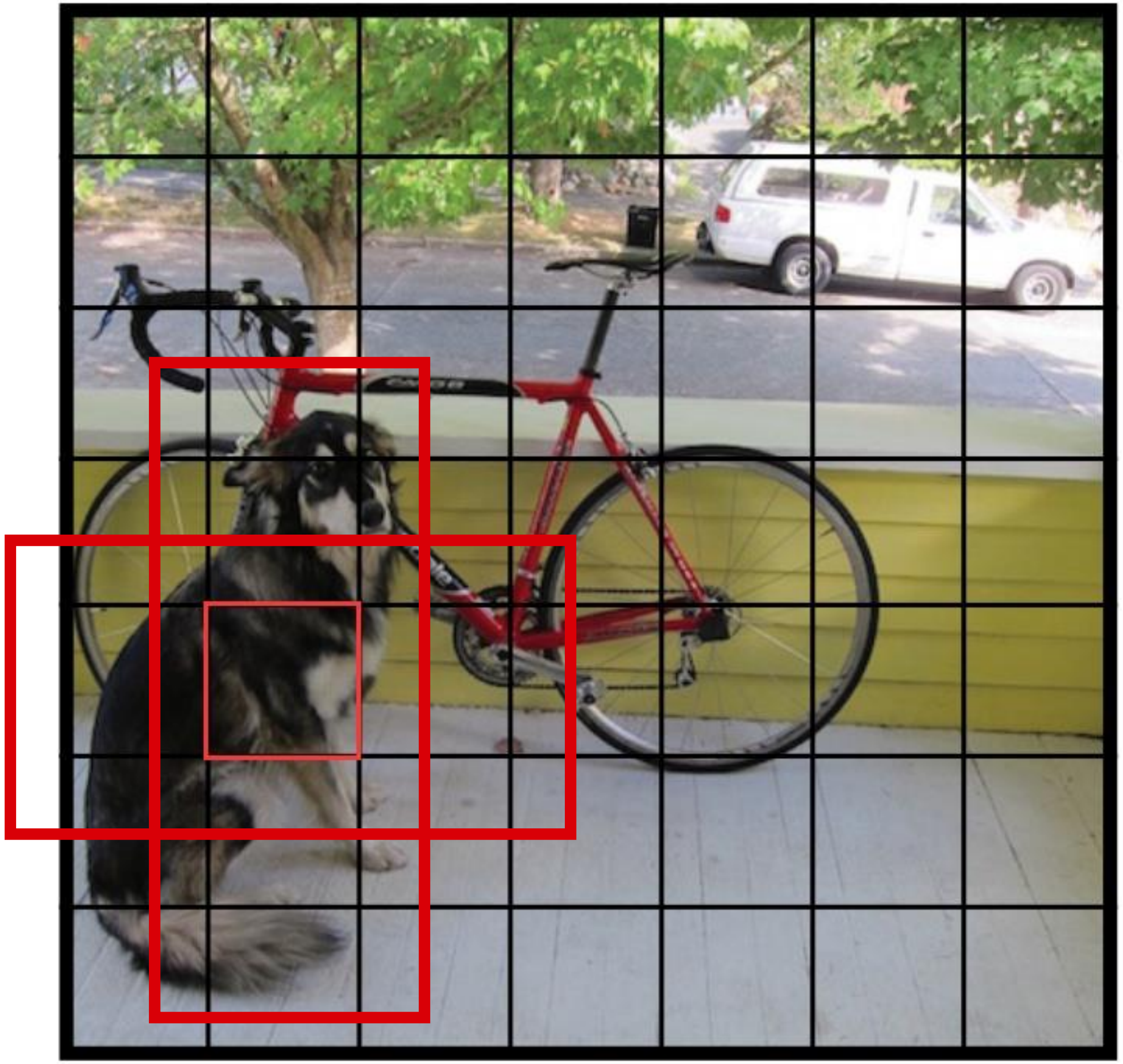
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

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

# Implementing details

1. Why B=2 bounding boxes for each cell?

Looks like if we have multiple workers working at the same time, we can choose the best one of them. Makes the box ratio variable and more likely to be consistent with the real object.



# Experiments

| Real-Time Detectors | Train | mAP | FPS |
|---|---|---|---|
| 100Hz DPM [30] | 2007 | 16.0 | 100 |
| 30Hz DPM [30] | 2007 | 26.1 | 30 |
| Fast YOLO | 2007+2012 | 52.7 | **155** |
| YOLO | 2007+2012 | **63.4** | 45 |
| Less Than Real-Time | | | |
| Fastest DPM [37] | 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[27] | 2007+2012 | 73.2 | 7 |
| Faster R-CNN ZF [27] | 2007+2012 | 62.1 | 18 |

**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.

# Experiments

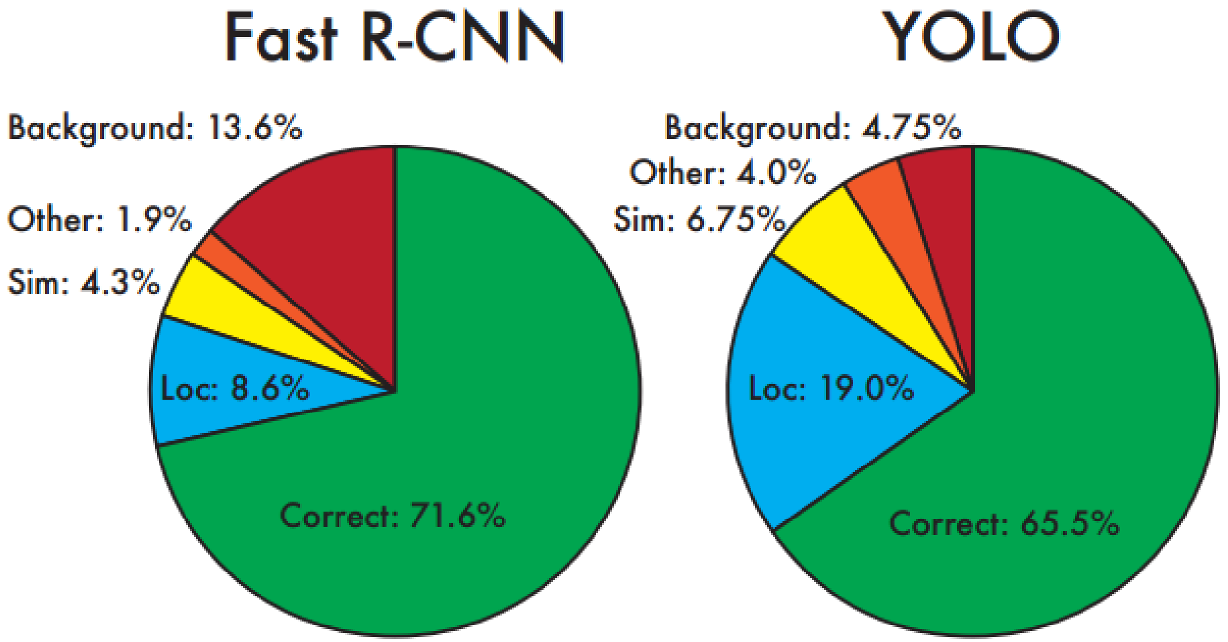



**Figure 4: Error Analysis: Fast R-CNN vs. YOLO** These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

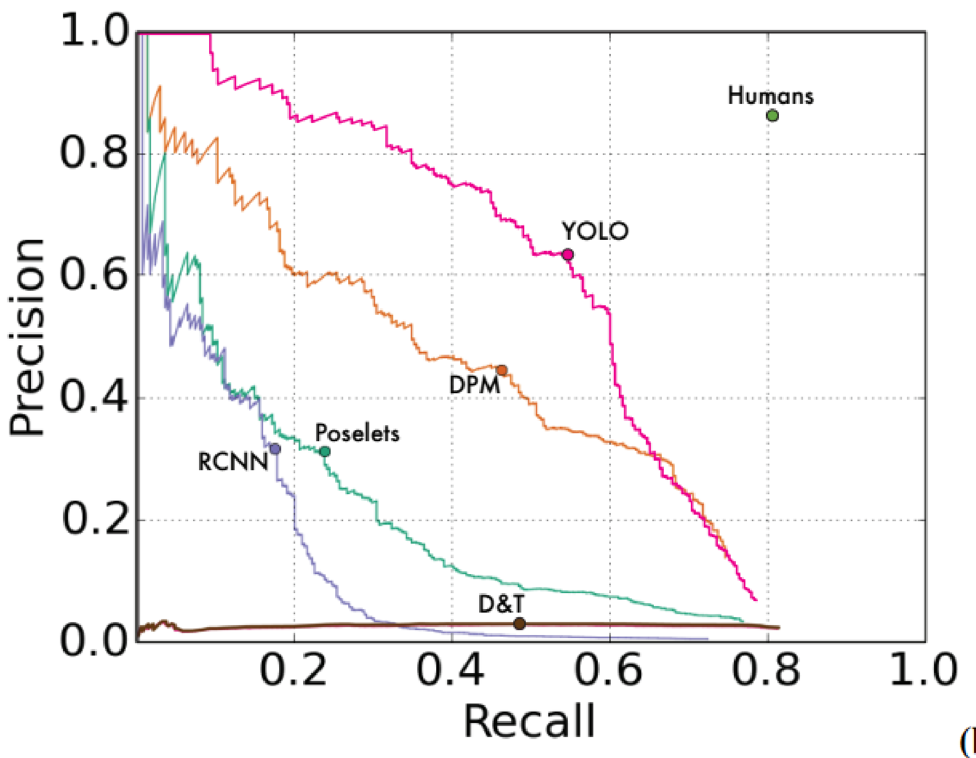
# Experiments

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

**Table 2: Model combination experiments on VOC 2007.** We examine the effect of combining various models with the best version of Fast R-CNN. Other versions of Fast R-CNN provide only a small benefit while YOLO provides a significant performance boost.

| VOC 2012 test | mAP | aero | bike | bird | boat | bottle | bus | car | cat | chair | cow | table | dog | horse | mbike | person | plant | sheep | sofa | train | tv |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MR_CNN_MORE_DATA [11] | **73.9** | **85.5** | **82.9** | **76.6** | **57.8** | **62.7** | **79.4** | 77.2 | 86.6 | **55.0** | **79.1** | **62.2** | 87.0 | **83.4** | **84.7** | 78.9 | 45.3 | 73.4 | 65.8 | 80.3 | 74.0 |
| HyperNet_VGG | 71.4 | 84.2 | 78.5 | 73.6 | 55.6 | 53.7 | 78.7 | **79.8** | 87.7 | 49.6 | 74.9 | 52.1 | 86.0 | 81.7 | 83.3 | **81.8** | **48.6** | **73.5** | 59.4 | 79.9 | 65.7 |
| HyperNet_SP | 71.3 | 84.1 | 78.3 | 73.3 | 55.5 | 53.6 | 78.6 | 79.6 | 87.5 | 49.5 | 74.9 | 52.1 | 85.6 | 81.6 | 83.2 | 81.6 | 48.4 | 73.2 | 59.3 | 79.7 | 65.6 |
| **Fast R-CNN + YOLO** | 70.7 | 83.4 | 78.5 | 73.5 | 55.8 | 43.4 | 79.1 | 73.1 | **89.4** | 49.4 | 75.5 | 57.0 | **87.5** | 80.9 | 81.0 | 74.7 | 41.8 | 71.5 | 68.5 | **82.1** | 67.2 |
| MR_CNN_S_CNN [11] | 70.7 | 85.0 | 79.6 | 71.5 | 55.3 | 57.7 | 76.0 | 73.9 | 84.6 | 50.5 | 74.3 | 61.7 | 85.5 | 79.9 | 81.7 | 76.4 | 41.0 | 69.0 | 61.2 | 77.7 | 72.1 |
| Faster R-CNN [27] | 70.4 | 84.9 | 79.8 | 74.3 | 53.9 | 49.8 | 77.5 | 75.9 | 88.5 | 45.6 | 77.1 | 55.3 | 86.9 | 81.7 | 80.9 | 79.6 | 40.1 | 72.6 | 60.9 | 81.2 | 61.5 |
| DEEP_ENS_COCO | 70.1 | 84.0 | 79.4 | 71.6 | 51.9 | 51.1 | 74.1 | 72.1 | 88.6 | 48.3 | 73.4 | 57.8 | 86.1 | 80.0 | 80.7 | 70.4 | 46.6 | 69.6 | **68.8** | 75.9 | 71.4 |
| NoC [28] | 68.8 | 82.8 | 79.0 | 71.6 | 52.3 | 53.7 | 74.1 | 69.0 | 84.9 | 46.9 | 74.3 | 53.1 | 85.0 | 81.3 | 79.5 | 72.2 | 38.9 | 72.4 | 59.5 | 76.7 | 68.1 |
| Fast R-CNN [14] | 68.4 | 82.3 | 78.4 | 70.8 | 52.3 | 38.7 | 77.8 | 71.6 | 89.3 | 44.2 | 73.0 | 55.0 | **87.5** | 80.5 | 80.8 | 72.0 | 35.1 | 68.3 | 65.7 | 80.4 | 64.2 |
| UMICH_FGS_STRUCT | 66.4 | 82.9 | 76.1 | 64.1 | 44.6 | 49.4 | 70.3 | 71.2 | 84.6 | 42.7 | 68.6 | 55.8 | 82.7 | 77.1 | 79.9 | 68.7 | 41.4 | 69.0 | 60.0 | 72.0 | 66.2 |
| NUS_NIN_C2000 [7] | 63.8 | 80.2 | 73.8 | 61.9 | 43.7 | 43.0 | 70.3 | 67.6 | 80.7 | 41.9 | 69.7 | 51.7 | 78.2 | 75.2 | 76.9 | 65.1 | 38.6 | 68.3 | 58.0 | 68.7 | 63.3 |
| BabyLearning [7] | 63.2 | 78.0 | 74.2 | 61.3 | 45.7 | 42.7 | 68.2 | 66.8 | 80.2 | 40.6 | 70.0 | 49.8 | 79.0 | 74.5 | 77.9 | 64.0 | 35.3 | 67.9 | 55.7 | 68.7 | 62.6 |
| NUS_NIN | 62.4 | 77.9 | 73.1 | 62.6 | 39.5 | 43.3 | 69.1 | 66.4 | 78.9 | 39.1 | 68.1 | 50.0 | 77.2 | 71.3 | 76.1 | 64.7 | 38.4 | 66.9 | 56.2 | 66.9 | 62.7 |
| R-CNN VGG BB [13] | 62.4 | 79.6 | 72.7 | 61.9 | 41.2 | 41.9 | 65.9 | 66.4 | 84.6 | 38.5 | 67.2 | 46.7 | 82.0 | 74.8 | 76.0 | 65.2 | 35.6 | 65.4 | 54.2 | 67.4 | 60.3 |
| R-CNN VGG [13] | 59.2 | 76.8 | 70.9 | 56.6 | 37.5 | 36.9 | 62.9 | 63.6 | 81.1 | 35.7 | 64.3 | 43.9 | 80.4 | 71.6 | 74.0 | 60.0 | 30.8 | 63.4 | 52.0 | 63.5 | 58.7 |
| **YOLO** | 57.9 | 77.0 | 67.2 | 57.7 | 38.3 | 22.7 | 68.3 | 55.9 | 81.4 | 36.2 | 60.8 | 48.5 | 77.2 | 72.3 | 71.3 | 63.5 | 28.9 | 52.2 | 54.8 | 73.9 | 50.8 |
| Feature Edit [32] | 56.3 | 74.6 | 69.1 | 54.4 | 39.1 | 33.1 | 65.2 | 62.7 | 69.7 | 30.8 | 56.0 | 44.6 | 70.0 | 64.4 | 71.1 | 60.2 | 33.3 | 61.3 | 46.4 | 61.7 | 57.8 |
| R-CNN BB [13] | 53.3 | 71.8 | 65.8 | 52.0 | 34.1 | 32.6 | 59.6 | 60.0 | 69.8 | 27.6 | 52.0 | 41.7 | 69.6 | 61.3 | 68.3 | 57.8 | 29.6 | 57.8 | 40.9 | 59.3 | 54.1 |
| SDS [16] | 50.7 | 69.7 | 58.4 | 48.5 | 28.3 | 28.8 | 61.3 | 57.5 | 70.8 | 24.1 | 50.7 | 35.9 | 64.9 | 59.1 | 65.8 | 57.1 | 26.0 | 58.8 | 38.6 | 58.9 | 50.7 |
| R-CNN [13] | 49.6 | 68.1 | 63.8 | 46.1 | 29.4 | 27.9 | 56.6 | 57.0 | 65.9 | 26.5 | 48.7 | 39.5 | 66.2 | 57.3 | 65.4 | 53.2 | 26.2 | 54.5 | 38.1 | 50.6 | 51.6 |

**Table 3: PASCAL VOC 2012 Leaderboard.** YOLO compared with the full `comp4` (outside data allowed) public leaderboard as of November 6th, 2015. Mean average precision and per-class average precision are shown for a variety of detection methods. YOLO is the only real-time detector. Fast R-CNN + YOLO is the forth highest scoring method, with a 2.3% boost over Fast R-CNN.




**(a)** Picasso Dataset precision-recall curves.

| | VOC 2007 | Picasso | | People-Art |
|---|---|---|---|---|
| | AP | AP | Best $F_1$ | AP |
| **YOLO** | **59.2** | **53.3** | **0.590** | **45** |
| R-CNN | 54.2 | 10.4 | 0.226 | 26 |
| DPM | 43.2 | 37.8 | 0.458 | 32 |
| Poselets [2] | 36.5 | 17.8 | 0.271 | |
| D&T [4] | - | 1.9 | 0.051 | |

**(b)** Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best $F_1$ score.

**Figure 5: Generalization results on Picasso and People-Art.**

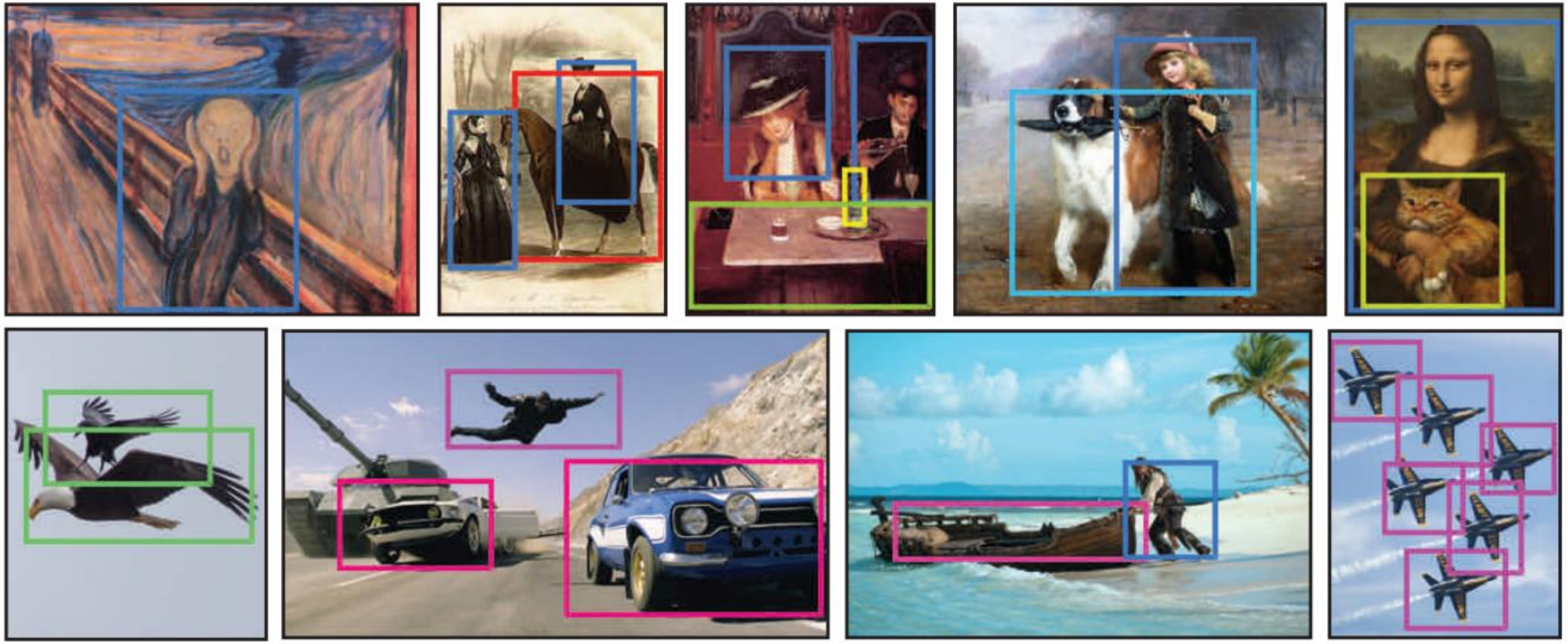


**Figure 6: Qualitative Results.** YOLO running on artwork and natural images. It is mostly accurate although it does think one person in an image is an airplane.

# Conclusion

1. YOLO is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly.
2. YOLO is highly dependent on labeling data
3. YOLO performs bad when detecting small objects or very close objects due to the grids they set
4. When one object is variable in shape and size, YOLO is difficult to tell it
5. The loss function in YOLO version 1 is still very unstable
6. YOLO V2, YOLO 9000 to be continued

```python
def _build_net(self):
    """build the network"""
    if self.verbose:
        print("Start to build the network ...")
    self.images = tf.placeholder(tf.float32, [None, 448, 448, 3])
    net = self._conv_layer(self.images, 1, 64, 7, 2)
    net = self._maxpool_layer(net, 1, 2, 2)
    net = self._conv_layer(net, 2, 192, 3, 1)
    net = self._maxpool_layer(net, 2, 2, 2)
    net = self._conv_layer(net, 3, 128, 1, 1)
    net = self._conv_layer(net, 4, 256, 3, 1)
    net = self._conv_layer(net, 5, 256, 1, 1)
    net = self._conv_layer(net, 6, 512, 3, 1)
    net = self._maxpool_layer(net, 6, 2, 2)
    net = self._conv_layer(net, 7, 256, 1, 1)
    net = self._conv_layer(net, 8, 512, 3, 1)
    net = self._conv_layer(net, 9, 256, 1, 1)
    net = self._conv_layer(net, 10, 512, 3, 1)
    net = self._conv_layer(net, 11, 256, 1, 1)
    net = self._conv_layer(net, 12, 512, 3, 1)
    net = self._conv_layer(net, 13, 256, 1, 1)
    net = self._conv_layer(net, 14, 512, 3, 1)
    net = self._conv_layer(net, 15, 512, 1, 1)
    net = self._conv_layer(net, 16, 1024, 3, 1)
    net = self._maxpool_layer(net, 16, 2, 2)
    net = self._conv_layer(net, 17, 512, 1, 1)
    net = self._conv_layer(net, 18, 1024, 3, 1)
    net = self._conv_layer(net, 19, 512, 1, 1)
    net = self._conv_layer(net, 20, 1024, 3, 1)
    net = self._conv_layer(net, 21, 1024, 3, 1)
    net = self._conv_layer(net, 22, 1024, 3, 2)
    net = self._conv_layer(net, 23, 1024, 3, 1)
    net = self._conv_layer(net, 24, 1024, 3, 1)
    net = self._flatten(net)
    net = self._fc_layer(net, 25, 512, activation=leak_relu)
    net = self._fc_layer(net, 26, 4096, activation=leak_relu)
    net = self._fc_layer(net, 27, self.S*self.S*(self.C+5*self.B))
    self.predicts = net
```