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

Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

Presented by:

Edward Kwao

Department of Electrical and Computer Engineering

April 23, 2025

Contents

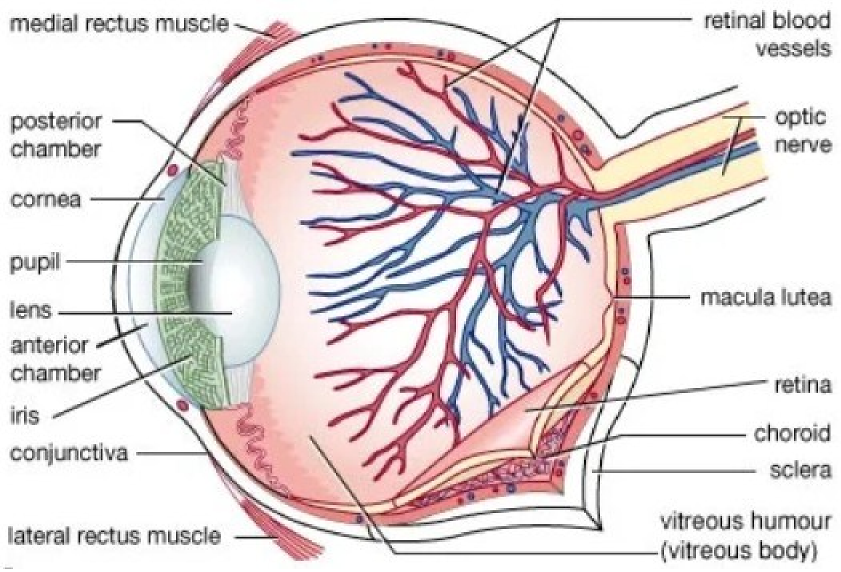
Introduction (1/3)

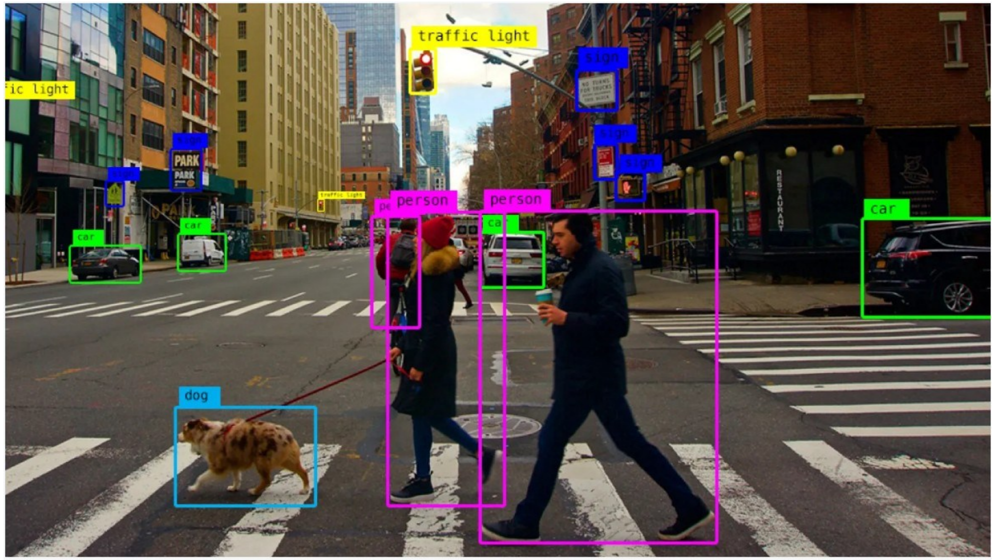
- **The human visual system is both fast and accurate**
  - Enables us to handle complex activities like driving with little conscious thought or navigating dark environments.

- To replicate this capability, we need object detection algorithms that are also fast and accurate
  - To support autonomous driving.
  - To support assistive technologies that provide real-time visual feedback to users.

- **Existing object detection systems include:**
  - DPM
    - This method uses a sliding window technique, where a classifier is applied at regular intervals across the image.
  - R-CNN
    - This approach generates potential bounding boxes using region proposals and then applies a classifier to these boxes.

- However, both DPM and R-CNN involve intricate multi-stage processes, making them relatively slow and difficult to fine-tune as each individual component requires separate training



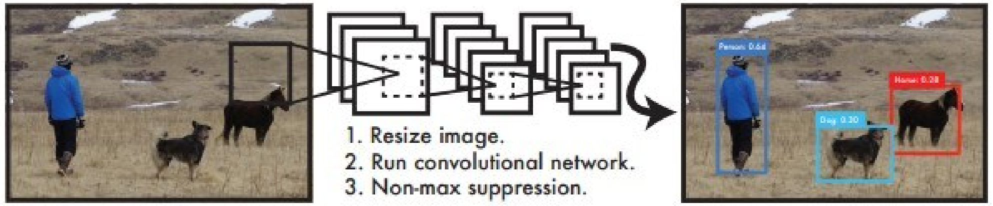


Introduction (2/3)

- **PROBLEM:** It is difficult to emulate the human eye's ability to detect objects instantly and effortlessly.
  - But we need to solve this fundamental challenge in computer vision

- Typical object detection workflows include:
  - Extracting meaningful features from the input image.
  - Using classifiers or localizers to detect objects in the feature space.
  - Running these models across the full image or selected regions, often using a sliding window approach.

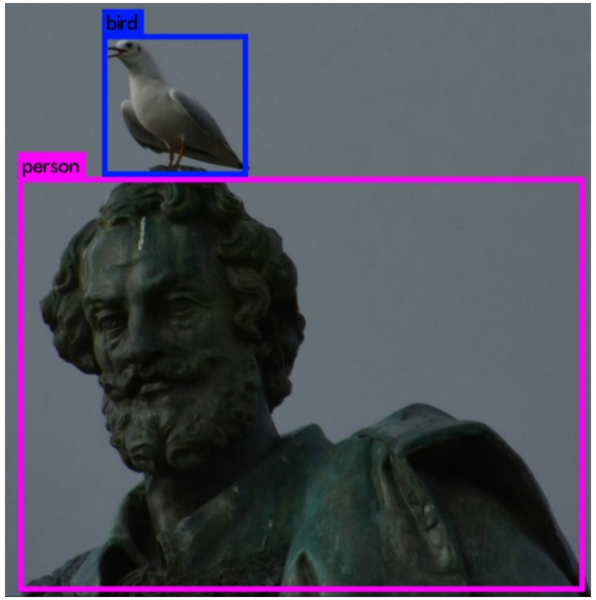
- The layered nature of these pipelines makes object detection systems slow and unsuitable for real-time applications.



- **YOLO Model**
  - Transforms object detection into a unified regression task.
  - Image pixels => Bounding boxes => Class probabilities of boxes.
  - Utilizes a single NN to predict multiple bounding. boxes and their class probabilities in one pass.
  - Designed to be fast and simple, avoiding multi-step pipelines.

Introduction (3/3)

- **YOLO vs DPM vs R-CNN: Advantage**
  - Extremely fast, capable of handling real-time video streams with only ~ 25 ms delay.
  - Learns features that generalize well.
  - Makes predictions by considering the entire image context, unlike localized approaches of DPM and R-CNN.



YOLO Unified Detection

- **YOLO splits the input image into an S x S grid**
  - Each cell in the grid predicts B bounding boxes along with a confidence score.
  - The confidence score reflects two things:
    1. The likelihood that an object exists in the box.
    2. How accurately the predicted box overlaps with the actual object
       - **Confidence = Pr(Object) * $\mathbf{IOU}_{\mathbf{Pred}}^{\mathbf{truth}}$**

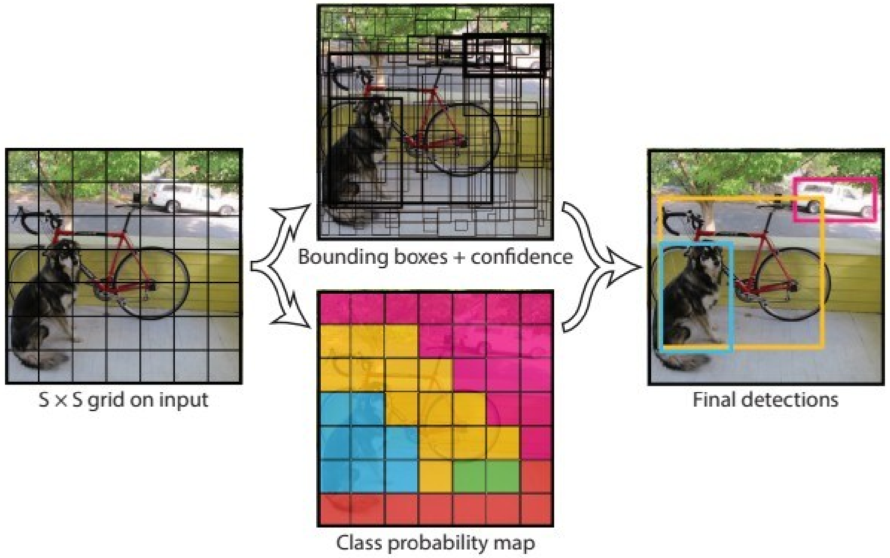
- Each bounding box includes 5 predicted values: $\boldsymbol{x, y, w, h, and\ confidence}$
  - $(\boldsymbol{x}, \boldsymbol{y})$: center of the box relative to the grid cell.
  - $\boldsymbol{w}\ and\ \boldsymbol{h}$ are relative to the overall image size.
  - Confidence prediction is the IOU between the predicted box and ground truth

- Each grid cell also predicts C conditional class probabilities

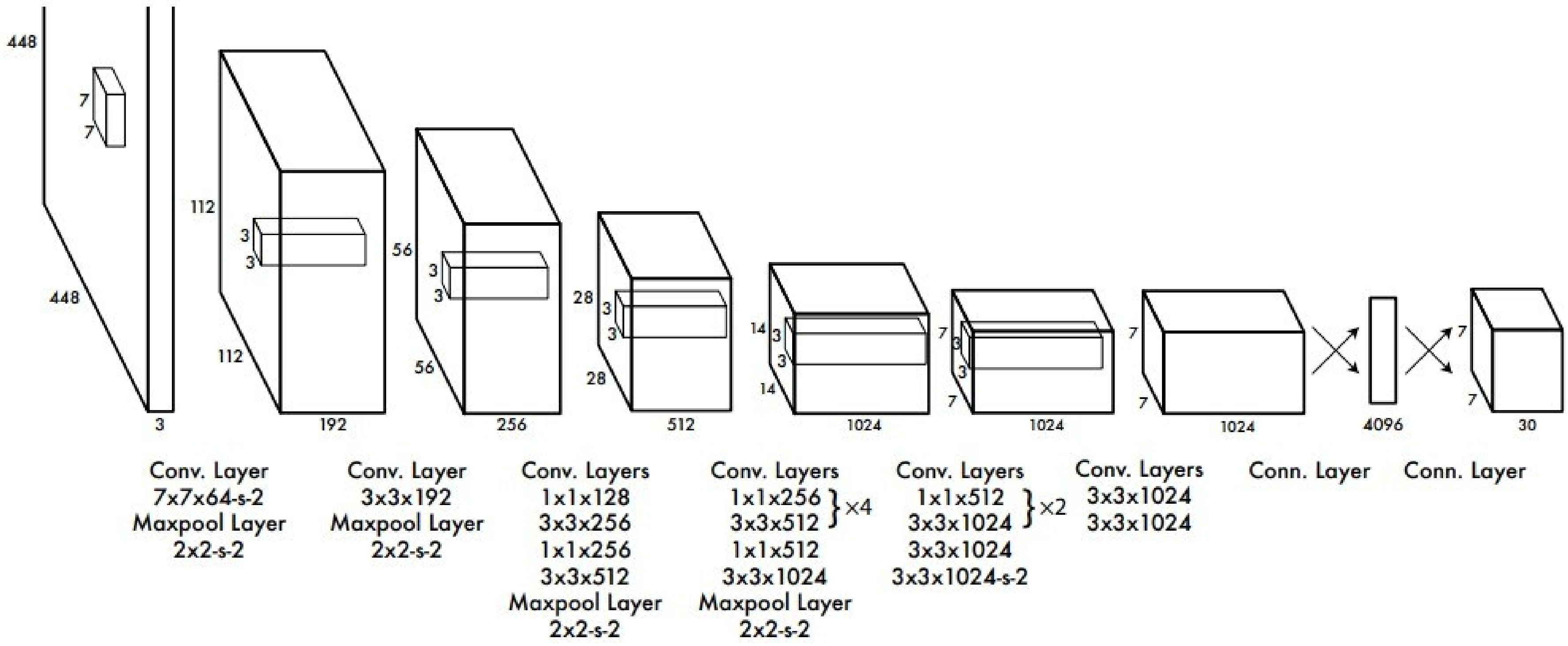
The probabilities are conditioned on the grid cell containing an object

- These are the probabilities of a class given an object is present $\mathbf{P_r(Class_i|Object)}$.
- Class prediction occurs only if the grid cell contains an object.
- Each class-specific confidence score combines:
  1. Probability of the class appearing in the box.
  2. How well the box aligns with the object it is supposed to detect.

$$\mathbf{P_r(Class_i|Object) * P_r(object) * IOU_{pred}^{truth} = P_r\left(Class_i * IOU_{pred}^{truth}\right)}$$



YOLO Network Design



- The architecture uses 24 conv layers to extract image features, followed by 2 fully connected layers that predict class probabilities and bounding box coordinates.

- Fast YOLO reduces complexity by using 9 conv layers and a smaller number of filters.

Training & Testing YOLO (1/5)

loss function:

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)
\end{aligned}
$$

where $\mathbb{1}_i^{\text{obj}}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

$\boldsymbol{\lambda_{\text{coord}}}$ = grid cell contains an object
$\boldsymbol{\lambda_{\text{noobj}}}$ = grid cell contains no object

Optimized during training:

- Model was trained for 135 epochs .
- Learning rate rises from 10^{-3} to 10^{-2}.

- The Model is trained on the Pascal Visual Object Classes (VOC) data
    - Darknet framework is used for all training and inference

Training & Testing YOLO (2/5)

```
int convolutional_out_height(convolutional_layer l)
{
    return (l.h + 2*l.pad - l.size) / l.stride + 1;
}

int convolutional_out_width(convolutional_layer l)
{
    return (l.w + 2*l.pad - l.size) / l.stride + 1;
}

image get_convolutional_image(convolutional_layer l)
{
    return float_to_image(l.out_w,l.out_h,l.out_c,l.output);
}

image get_convolutional_delta(convolutional_layer l)
{
    return float_to_image(l.out_w,l.out_h,l.out_c,l.delta);
}
```

- **Darknet C implementation of how to compute the height and width of an output feature map after applying the convolutional layer in terms of Filter size, Stride and Padding**

- **returns an image representation of the output feature map from the convolutional layer.**

- **Returns the gradient (delta) of the convolutional layer as an image.**

Training & Testing YOLO (3/5)

```
void forward_maxpool_layer(const maxpool_layer l, network net)
{
    int b,i,j,k,m,n;
    int w_offset = -l.pad/2;
    int h_offset = -l.pad/2;

    int h = l.out_h;
    int w = l.out_w;
    int c = l.c;

    for(b = 0; b < l.batch; ++b){
        for(k = 0; k < c; ++k){
            for(i = 0; i < h; ++i){
                for(j = 0; j < w; ++j){
                    int out_index = j + w*(i + h*(k + c*b));
                    float max = -FLT_MAX;
                    int max_i = -1;
                    for(n = 0; n < l.size; ++n){
                        for(m = 0; m < l.size; ++m){
                            int cur_h = h_offset + i*l.stride + n;
                            int cur_w = w_offset + j*l.stride + m;
                            int index = cur_w + l.w*(cur_h + l.h*(k + b*l.c));
                            int valid = (cur_h >= 0 && cur_h < l.h &&
                                         cur_w >= 0 && cur_w < l.w);
                            float val = (valid != 0) ? net.input[index] : -FLT_MAX;
                            max_i = (val > max) ? index : max_i;
                            max   = (val > max) ? val   : max;
                        }
                    }
                    l.output[out_index] = max;
                    l.indexes[out_index] = max_i;
                }
            }
        }
    }
}

void backward_maxpool_layer(const maxpool_layer l, network net)
{
    int i;
    int h = l.out_h;
    int w = l.out_w;
    int c = l.c;
    for(i = 0; i < h*w*c*l.batch; ++i){
        int index = l.indexes[i];
        net.delta[index] += l.delta[i];
    }
}
```

- Darknet C implementation of forward and backward passes of a Max Pooling Layer

Training & Testing YOLO (4/5)

```c
layer make_connected_layer(int batch, int inputs, int outputs, ACTIVATION activation, int batch_normalize, int adam)
{
    int i;
    layer l = {0};
    l.learning_rate_scale = 1;
    l.type = CONNECTED;

    l.inputs = inputs;
    l.outputs = outputs;
    l.batch=batch;
    l.batch_normalize = batch_normalize;
    l.h = 1;
    l.w = 1;
    l.c = inputs;
    l.out_h = 1;
    l.out_w = 1;
    l.out_c = outputs;

    l.output = calloc(batch*outputs, sizeof(float));
    l.delta = calloc(batch*outputs, sizeof(float));

    l.weight_updates = calloc(inputs*outputs, sizeof(float));
    l.bias_updates = calloc(outputs, sizeof(float));

    l.weights = calloc(outputs*inputs, sizeof(float));
    l.biases = calloc(outputs, sizeof(float));

    l.forward = forward_connected_layer;
    l.backward = backward_connected_layer;
    l.update = update_connected_layer;

    //float scale = 1./sqrt(inputs);
    float scale = sqrt(2./inputs);
    for(i = 0; i < outputs*inputs; ++i){
        l.weights[i] = scale*rand_uniform(-1, 1);
    }

    for(i = 0; i < outputs; ++i){
        l.biases[i] = 0;
    }

    if(adam){
        l.m = calloc(l.inputs*l.outputs, sizeof(float));
        l.v = calloc(l.inputs*l.outputs, sizeof(float));
        l.bias_m = calloc(l.outputs, sizeof(float));
        l.scale_m = calloc(l.outputs, sizeof(float));
        l.bias_v = calloc(l.outputs, sizeof(float));
        l.scale_v = calloc(l.outputs, sizeof(float));
    }
    if(batch_normalize){
        l.scales = calloc(outputs, sizeof(float));
        l.scale_updates = calloc(outputs, sizeof(float));
        for(i = 0; i < outputs; ++i){
            l.scales[i] = 1;
        }

        l.mean = calloc(outputs, sizeof(float));
        l.mean_delta = calloc(outputs, sizeof(float));
        l.variance = calloc(outputs, sizeof(float));
        l.variance_delta = calloc(outputs, sizeof(float));

        l.rolling_mean = calloc(outputs, sizeof(float));
        l.rolling_variance = calloc(outputs, sizeof(float));

        l.x = calloc(batch*outputs, sizeof(float));
        l.x_norm = calloc(batch*outputs, sizeof(float));
    }
```

```c
void forward_connected_layer(layer l, network net)
{
    fill_cpu(l.outputs*l.batch, 0, l.output, 1);
    int m = l.batch;
    int k = l.inputs;
    int n = l.outputs;
    float *a = net.input;
    float *b = l.weights;
    float *c = l.output;
    gemm(0,1,m,n,k,1,a,k,b,k,1,c,n);
    if(l.batch_normalize){
        forward_batchnorm_layer(l, net);
    } else {
        add_bias(l.output, l.biases, l.batch, l.outputs, 1);
    }
    activate_array(l.output, l.outputs*l.batch, l.activation);
}

void backward_connected_layer(layer l, network net)
{
    gradient_array(l.output, l.outputs*l.batch, l.activation, l.delta);

    if(l.batch_normalize){
        backward_batchnorm_layer(l, net);
    } else {
        backward_bias(l.bias_updates, l.delta, l.batch, l.outputs, 1);
    }

    int m = l.outputs;
    int k = l.batch;
    int n = l.inputs;
    float *a = l.delta;
    float *b = net.input;
    float *c = l.weight_updates;
    gemm(1,0,m,n,k,1,a,m,b,n,1,c,n);

    m = l.batch;
    k = l.outputs;
    n = l.inputs;

    a = l.delta;
    b = l.weights;
    c = net.delta;

    if(c) gemm(0,0,m,n,k,1,a,k,b,n,1,c,n);
}
```

```c
void forward_connected_layer_gpu(layer l, network net)
{
    fill_gpu(l.outputs*l.batch, 0, l.output_gpu, 1);

    int m = l.batch;
    int k = l.inputs;
    int n = l.outputs;
    float * a = net.input_gpu;
    float * b = l.weights_gpu;
    float * c = l.output_gpu;
    gemm_gpu(0,1,m,n,k,1,a,k,b,k,1,c,n);

    if (l.batch_normalize) {
        forward_batchnorm_layer_gpu(l, net);
    } else {
        add_bias_gpu(l.output_gpu, l.biases_gpu, l.batch, l.outputs, 1);
    }
    activate_array_gpu(l.output_gpu, l.outputs*l.batch, l.activation);
}

void backward_connected_layer_gpu(layer l, network net)
{
    constrain_gpu(l.outputs*l.batch, 1, l.delta_gpu, 1);
    gradient_array_gpu(l.output_gpu, l.outputs*l.batch, l.activation, l.delta_gpu);
    if(l.batch_normalize){
        backward_batchnorm_layer_gpu(l, net);
    } else {
        backward_bias_gpu(l.bias_updates_gpu, l.delta_gpu, l.batch, l.outputs, 1);
    }

    int m = l.outputs;
    int k = l.batch;
    int n = l.inputs;
    float * a = l.delta_gpu;
    float * b = net.input_gpu;
    float * c = l.weight_updates_gpu;
    gemm_gpu(1,0,m,n,k,1,a,m,b,n,1,c,n);

    m = l.batch;
    k = l.outputs;
    n = l.inputs;

    a = l.delta_gpu;
    b = l.weights_gpu;
    c = net.delta_gpu;

    if(c) gemm_gpu(0,0,m,n,k,1,a,k,b,n,1,c,n);
}
#endif
```

Training & Testing YOLO (5/5)

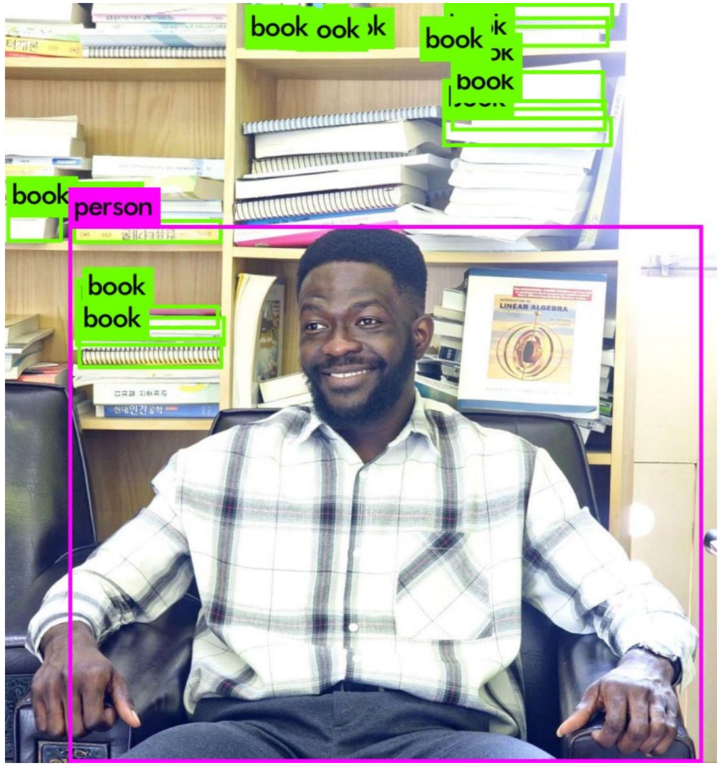
- **Testing pretrained YOLO model on images**

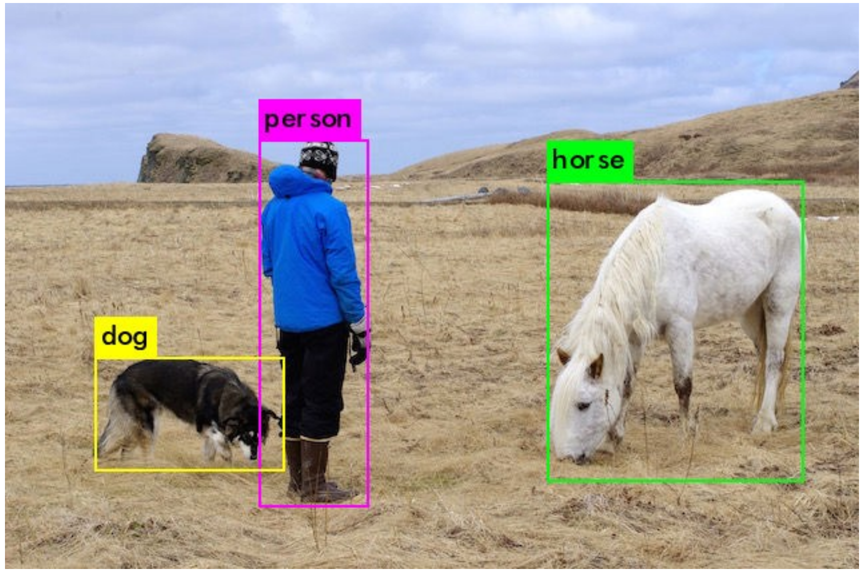
```
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   608 x 608 x   3   ->   608 x 608 x  32  0.639 BFLOPs
    1 conv     64  3 x 3 / 2   608 x 608 x  32   ->   304 x 304 x  64  3.407 BFLOPs
    2 conv     32  1 x 1 / 1   304 x 304 x  64   ->   304 x 304 x  32  0.379 BFLOPs
    3 conv     64  3 x 3 / 1   304 x 304 x  32   ->   304 x 304 x  64  3.407 BFLOPs
    4 res    1                 304 x 304 x  64   ->   304 x 304 x  64
    5 conv    128  3 x 3 / 2   304 x 304 x  64   ->   152 x 152 x 128  3.407 BFLOPs
    6 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64  0.379 BFLOPs
    7 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128  3.407 BFLOPs
    8 res    5                 152 x 152 x 128   ->   152 x 152 x 128
    9 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64  0.379 BFLOPs
   10 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128  3.407 BFLOPs
   11 res    8                 152 x 152 x 128   ->   152 x 152 x 128
   12 conv    256  3 x 3 / 2   152 x 152 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   13 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   14 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   15 res   12                  76 x  76 x 256   ->    76 x  76 x 256
   16 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   17 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   18 res   15                  76 x  76 x 256   ->    76 x  76 x 256
   19 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   20 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   21 res   18                  76 x  76 x 256   ->    76 x  76 x 256
   22 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   23 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   24 res   21                  76 x  76 x 256   ->    76 x  76 x 256
   25 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   26 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   27 res   24                  76 x  76 x 256   ->    76 x  76 x 256
   28 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   29 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   30 res   27                  76 x  76 x 256   ->    76 x  76 x 256
   31 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   32 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   33 res   30                  76 x  76 x 256   ->    76 x  76 x 256
   34 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   35 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   36 res   33                  76 x  76 x 256   ->    76 x  76 x 256
   37 conv    512  3 x 3 / 2    76 x  76 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   38 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   39 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   40 res   37                  38 x  38 x 512   ->    38 x  38 x 512
   41 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   42 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   43 res   40                  38 x  38 x 512   ->    38 x  38 x 512
   44 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   45 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   46 res   43                  38 x  38 x 512   ->    38 x  38 x 512
   47 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   48 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   49 res   46                  38 x  38 x 512   ->    38 x  38 x 512
   50 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   51 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   52 res   49                  38 x  38 x 512   ->    38 x  38 x 512
   53 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   54 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   55 res   52                  38 x  38 x 512   ->    38 x  38 x 512
   56 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   57 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   58 res   55                  38 x  38 x 512   ->    38 x  38 x 512
   59 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   60 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
```

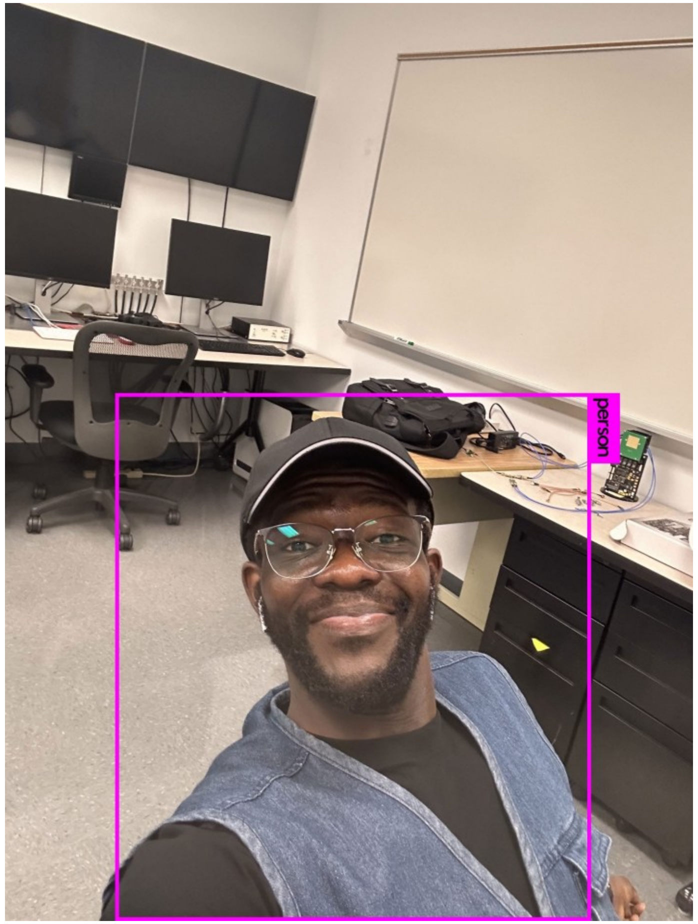
```
   61 res   58                  38 x  38 x 512   ->    38 x  38 x 512
   62 conv   1024  3 x 3 / 2    38 x  38 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   63 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   64 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   65 res   62                  19 x  19 x1024   ->    19 x  19 x1024
   66 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   67 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   68 res   65                  19 x  19 x1024   ->    19 x  19 x1024
   69 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   70 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   71 res   68                  19 x  19 x1024   ->    19 x  19 x1024
   72 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   73 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   74 res   71                  19 x  19 x1024   ->    19 x  19 x1024
   75 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   76 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   77 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   78 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   79 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   80 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   81 conv    255  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 255  0.189 BFLOPs
   82 yolo
   83 route  79
   84 conv    256  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 256  0.095 BFLOPs
   85 upsample            2x    19 x  19 x 256   ->    38 x  38 x 256
   86 route  85 61
   87 conv    256  1 x 1 / 1    38 x  38 x 768   ->    38 x  38 x 256  0.568 BFLOPs
   88 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   89 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   90 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   91 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   92 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   93 conv    255  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 255  0.377 BFLOPs
   94 yolo
   95 route  91
   96 conv    128  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 128  0.095 BFLOPs
   97 upsample            2x    38 x  38 x 128   ->    76 x  76 x 128
   98 route  97 36
   99 conv    128  1 x 1 / 1    76 x  76 x 384   ->    76 x  76 x 128  0.568 BFLOPs
  100 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  101 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
  102 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  103 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
  104 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  105 conv    255  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 255  0.754 BFLOPs
  106 yolo
Loading weights from yolov3.weights...Done!
data/horses.jpg: Predicted in 6.276225 seconds.
horse: 100%
horse: 100%
horse: 96%
horse: 95%
```

```
   55 res   52                  38 x  38 x 512   ->    38 x  38 x 512
   56 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   57 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   58 res   55                  38 x  38 x 512   ->    38 x  38 x 512
   59 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   60 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   61 res   58                  38 x  38 x 512   ->    38 x  38 x 512
   62 conv   1024  3 x 3 / 2    38 x  38 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   63 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   64 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   65 res   62                  19 x  19 x1024   ->    19 x  19 x1024
   66 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   67 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   68 res   65                  19 x  19 x1024   ->    19 x  19 x1024
   69 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   70 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   71 res   68                  19 x  19 x1024   ->    19 x  19 x1024
   72 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   73 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   74 res   71                  19 x  19 x1024   ->    19 x  19 x1024
   75 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   76 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   77 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   78 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   79 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   80 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   81 conv    255  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 255  0.189 BFLOPs
   82 yolo
   83 route  79
   84 conv    256  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 256  0.095 BFLOPs
   85 upsample            2x    19 x  19 x 256   ->    38 x  38 x 256
   86 route  85 61
   87 conv    256  1 x 1 / 1    38 x  38 x 768   ->    38 x  38 x 256  0.568 BFLOPs
   88 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   89 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   90 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   91 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   92 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   93 conv    255  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 255  0.377 BFLOPs
   94 yolo
   95 route  91
   96 conv    128  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 128  0.095 BFLOPs
   97 upsample            2x    38 x  38 x 128   ->    76 x  76 x 128
   98 route  97 36
   99 conv    128  1 x 1 / 1    76 x  76 x 384   ->    76 x  76 x 128  0.568 BFLOPs
  100 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  101 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
  102 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  103 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
  104 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  105 conv    255  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 255  0.754 BFLOPs
  106 yolo
Loading weights from yolov3.weights...Done!
data/eddy_1.jpg: Predicted in 6.005917 seconds.
book: 76%
book: 65%
book: 64%
book: 64%
book: 64%
book: 62%
book: 62%
book: 62%
book: 60%
book: 60%
book: 59%
book: 58%
book: 56%
book: 52%
book: 50%
person: 99%
```

YOLO Result







- Testing pretrained YOLO model on the image

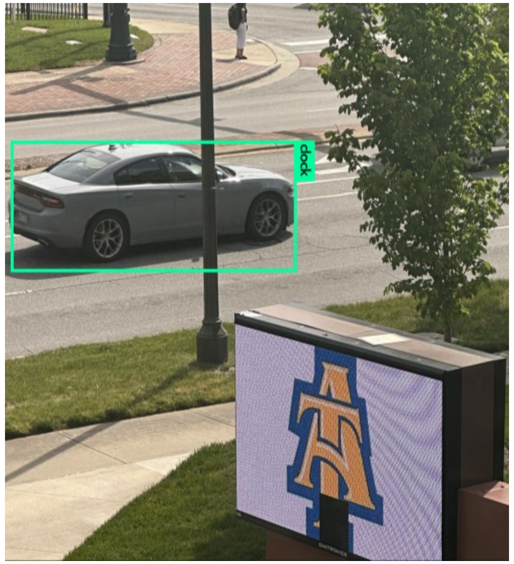
YOLO Limitations

- Because YOLO learns to estimate bounding boxes directly from the data, it has difficulty adapting to objects with unfamiliar or unusual shapes.

- The loss function does not distinguish between errors in small and large bounding boxes:
    - A minor error in a large bounding box is often negligible, but a similar error in a small box significantly impacts the IOU score.
    - Most of the errors come from inaccurate object localization.

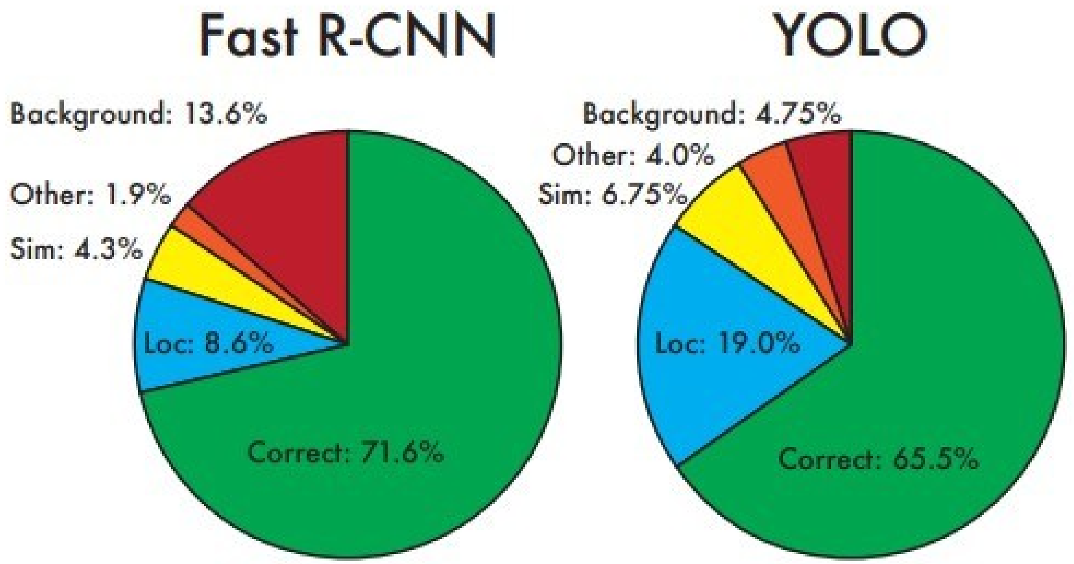
- Struggles with precision – While it quickly detects objects, it sometimes fails to accurately pinpoint their locations.

- YOLO places strict spatial constraints, which reduces its ability to detect multiple closely located objects.

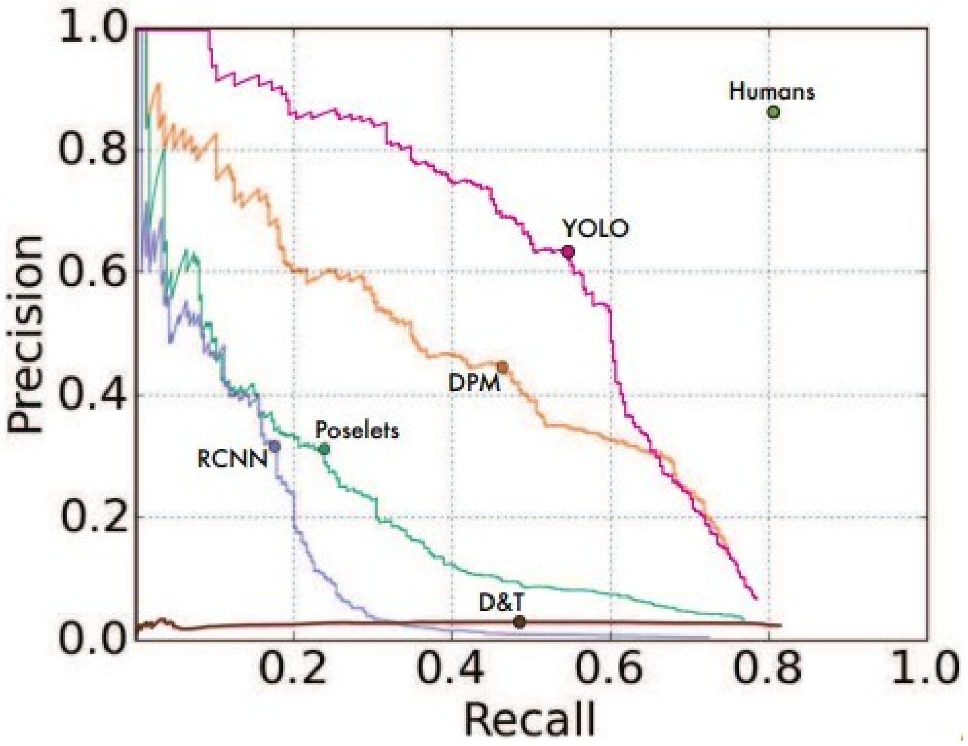


YOLO vs Other Object Detection Systems (1/2)

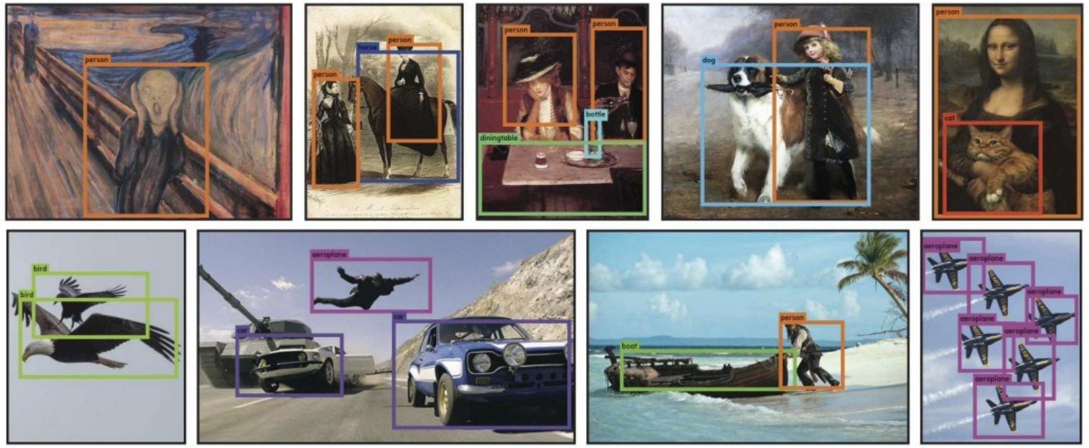
Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21



YOLO vs Other Object Detection Systems (2/2)



	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	



Conclusion

- YOLO introduces a unified framework for object detection:
  - It’s easy to build and can be trained on entire images.
  - Unlike traditional methods that treat detection as a classification problem, YOLO uses a loss function tailored for detection accuracy and trains the model end-to-end.

- YOLO adapts effectively to new environments, making it well-suited for real-time object detection tasks that require speed and reliability.

- Fast YOLO is recognized as the quickest general-purpose object detection model available and is capable of real-time detection.

E-mail: ekwao at aggies.ncat.edu