/ OSEDEA

ConFoo

Golang VS Python: Serve an AI model for object detection

/OSEDEA

We are a Montreal-based innovation and technology firm, founded in 2011

We design intelligent, high-performance applications, tailored to the specific needs of our customers

Our services

/ Software engineering
/ User Experience Design (UX/UI)
/ Artificial Intelligence
/ Robotics

Technology, for us, is a means within our innovation process, not the ultimate goal

https://github.com/ArmandBriere/confoo2024

Objective of the presentation

Go
Programming language

Python
Programming language

+ Add comparison

Worldwide | Past 5 years | All categories | Web Search

Interest over time


Python
Golang
ME RUNNING AI MODEL IN PRODUCTION
JAKE-CLARK.TUMBLR

Tools

- Python 3.11.7
- Go version go1.22.0 linux/amd64
- OpenCV version: 4.9.0
- YoloV4

What is Python?

Python is a high-level, general-purpose programming language

What is Go?

Go is a statically typed, compiled high-level programming language designed at Google

Hello Software Crafters

```python
# python
print("Hello Software Crafters!")
```

```go
// go
package main

import "fmt"

func main() {
    fmt.Println("Hello Software Crafters!")
}
```

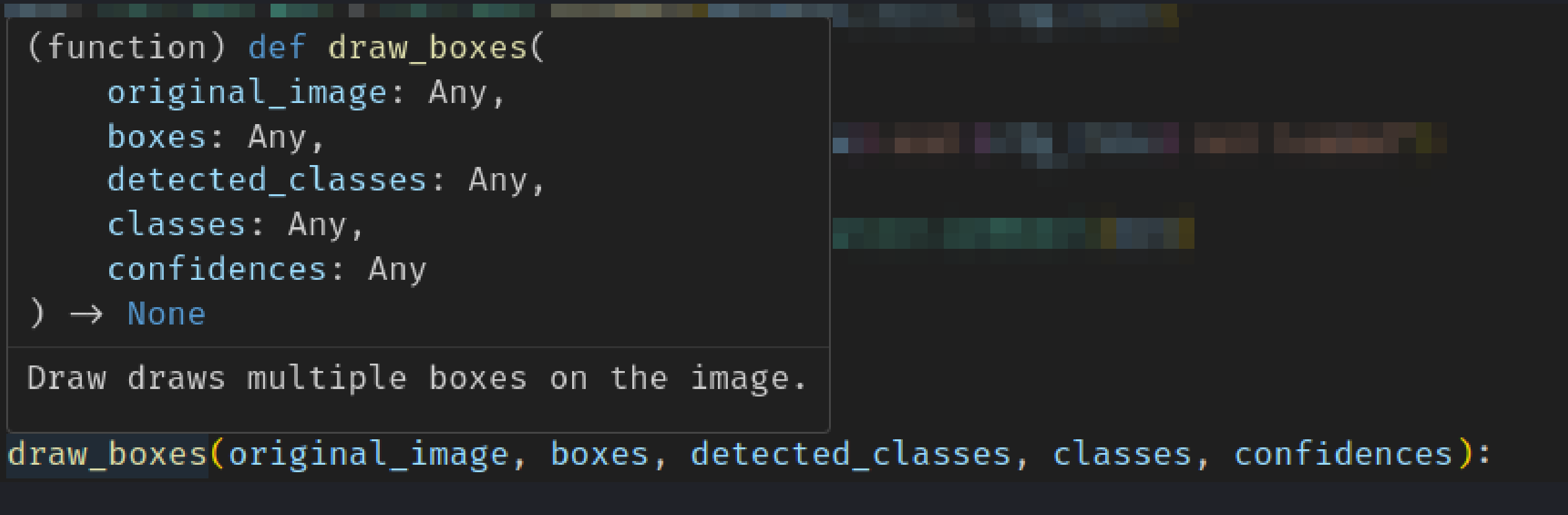
How to run them?

```
python main.py

go build main.go
./main
```

Faster way to run code in dev

```
go run main.go
```

Typed vs. Untyped


(function) def draw_boxes(
original_image: Any,
boxes: Any,
detected_classes: Any,
classes: Any,
confidences: Any
) → None
Draw draws multiple boxes on the image.
draw_boxes(original_image, boxes, detected_classes, classes, confidences):

```
(function) def draw_boxes(
    original_image: ndarray,
    boxes: List[List[int]],
    detected_classes: List[int],
    classes: List[str],
    confidences: List[float]
) → None
```

Draw draws multiple boxes on the image.

```
draw_boxes(original_image, boxes, detected_classes, classes, scores)
```

You should type

```python
# python
i = 3

j: int = 3
```

```go
// go
var i int
i = 3

j := 4
```

Functions

```python
# python
def addition(x: int, y: int) -> int:
    return x + y
```

```go
// go
func addition(x int, y int) int {
    return x + y
}
```

Bonus points

```go
// go
func subtraction(x int, y int) int {
    var unused_var int
    return x - y
}
```

```
# softwareCrafters.inference/playground
./playground.go:23:6: unused_var declared and not used
```

```go
// go
func subtraction(x int, y int) (result int) {
    result = x - y
    return
}
```

Let's dive in

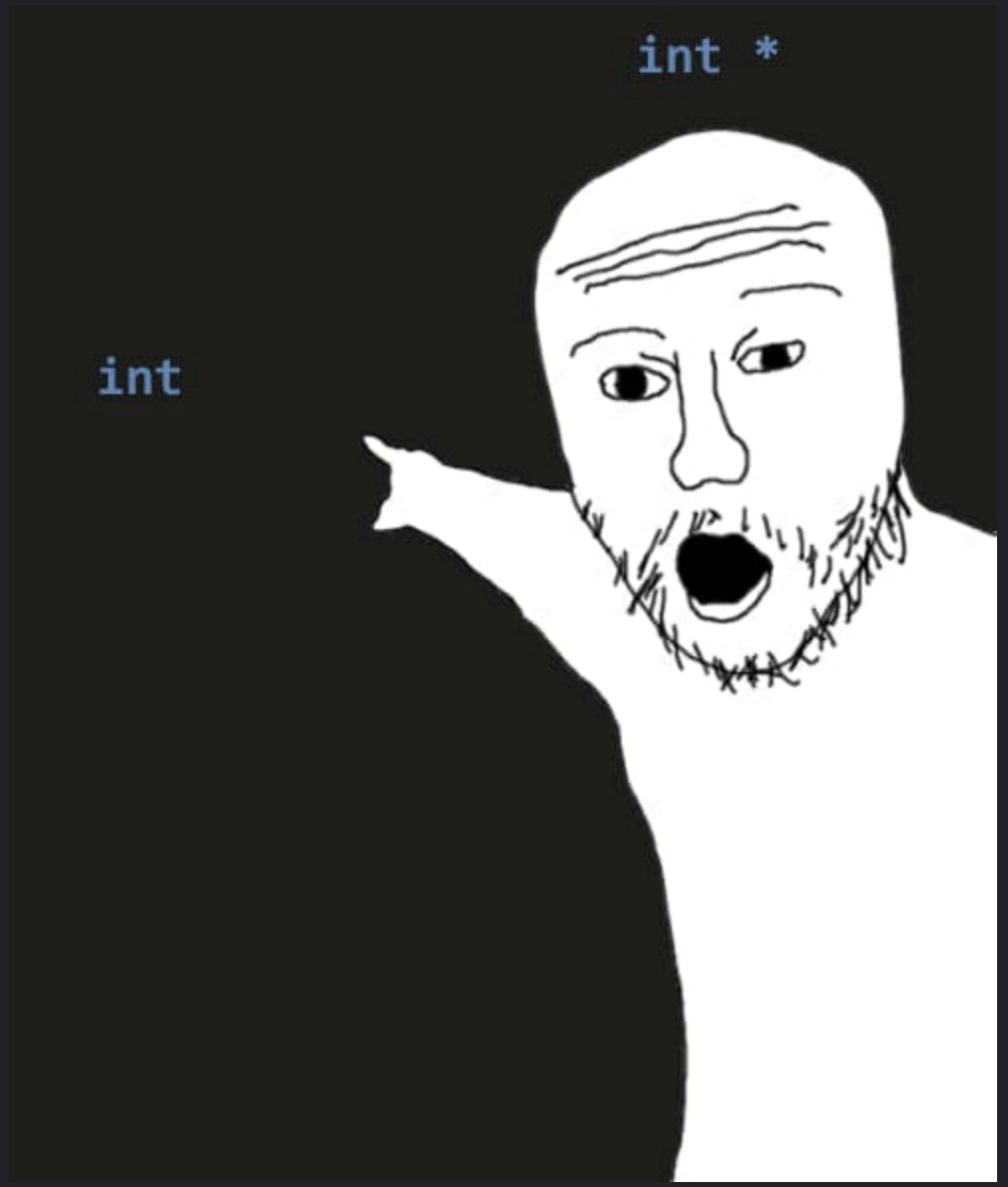
```python
# python
if __name__ == "__main__":
    # Parse the command line arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--img", default=str("person.jpg"), help="Path to input image.")
    args = parser.parse_args()

    # Load the model and classes
    net_model: cv2.dnn.DetectionModel = load_model(
        "yolov4-tiny.weights", "yolov4-tiny.cfg"
    )
    with open("classes.yml", "r") as file:
        config: dict = yaml.safe_load(file)
        classes: List[str] = config["names"]

    detect(net_model, args.img, classes)
```

```go
// go
// main function
func main() {
    // Parse the command line arguments
    img := flag.String("img", "person.jpg", "Path to input image.")
    flag.Parse()

    // Load the model and classes
    netModel := loadModel("yolov4-tiny.weights", "yolov4-tiny.cfg")
    classes := loadYAML("classes.yml")

    detect(netModel, *img, classes)
}
```

What is `*img`?

Oh that's a pointer...

How Pointers works in Go


int *
int

```go
// go
func main() {
    x := 42
    p := &x

    fmt.Println("x", x)
    fmt.Println("p", p)
    fmt.Println("&x", &x)
    fmt.Println("*p", *p)
    fmt.Println("*&x", *&x)
    fmt.Println("*&*&*&*&*&*&*&*&x", *&*&*&*&*&*&*&*&x)
}
```

```
x    42
p    0xc000012158
&x   0xc000012158
*p   42
*&x  42
*&*&*&*&*&*&*&*&x 42
```

Back to main

`*img` is just the file name

```
python main.py --img bus.jpg

go run main.go --img bus.jpg
```

Load model

```python
# python
def load_model(weights_file: str, cfg_file: str) -> cv2.dnn.DetectionModel:
    """Load yolo model with cv.dnn."""

    net: cv2.dnn.Net = cv2.dnn.readNet(weights_file, cfg_file)

    if net.empty():
        raise ValueError(f"Model {weights_file} and {cfg_file} not loaded")

    model: cv2.dnn.DetectionModel = cv2.dnn.DetectionModel(net)

    return model
```

What is `cv2.dnn.readNet`?

```
// go
// init loads the Yolo model.
func loadModel(yoloModelName string, yoloModelCfg string) gocv.Net {
    net := gocv.ReadNet(yoloModelName, yoloModelCfg)

    if net.Empty() {
        panic("Failed to load the model")
    }

    return net
}
```


OpenCV

```
pip install opencv-python
```

```
go get -u -d gocv.io/x/gocv
```

What is this yolo model?

You only look once (YOLO)

=

- Image Recognition
- Image Localization


M1 SOL/SEVILLA
EMT MADRID
Un minibús 100% eléctrico si es plan
prueba
eléctricamEMTe
cero emisiones


bus (0.66)
person (0.99)
person (0.93)
person (0.7
EMT MADRID
Un minibús 100% eléctrico si es plan
cero emisiones

Load yaml

```python
# python
with open("classes.yml", "r") as file:
    config: dict = yaml.safe_load(file)
    classes: List[str] = config["names"]
```

```go
// go
// loadYAML loads the YAML file and returns the Config struct
func loadYAML(filename string) []string {
    var config Config
    data, err := os.ReadFile(filename)
    if err != nil {
        panic(err)
    }
    err = yaml.Unmarshal(data, &config)
    if err != nil {
        panic(err)
    }
    return config.Names
}
```

> Marshalling is the process of transforming memory representation of an object into a data format suitable for storage

Marshalling Wikipedia


Checkpoint

- Basic Python and Go structure
  - How to run program
  - Variables, function
  - Standard file structure
- Typing
- Pointers
- Basic Yolo concept
- File read and data marshalling

```python
# python
def detect(net: cv2.dnn.DetectionModel, input_image: str, classes: List[str]):
    """Perform inference on the input image."""

    # Read the image
    original_image: np.ndarray = cv2.imread(input_image)

    # Prepare the model
    net.setInputParams(size=(416, 416), scale=1 / 255, swapRB=True)

    # Perform inference
    detected_classes, scores, boxes = net.detect(original_image, THRESHOLD, 0.25)

    # Draw boxes on image
    draw_boxes(original_image, boxes, detected_classes, classes, scores)

    # Save the image
    cv2.imwrite("python_inference.jpg", original_image)
```

```go
// go
// detect perform inference on the input image
func detect(net gocv.Net, imgPath string, classes []string) {

    // Read the image
    b, err := os.ReadFile(imgPath)
    img, err := gocv.IMDecode(b, gocv.IMReadAnyColor)

    // Prepare the model
    blob := gocv.BlobFromImage(img, 1/255.0, image.Pt(416, 416), gocv.NewScalar(0, 0, 0, 0), true, false)
    net.SetInput(blob, "")

    // Perform inference
    probs := net.ForwardLayers(getOutputLayerNames(&net))

    // Postprocess the output
    detected := postProcess(probs, 0.4, float32(img.Cols()), float32(img.Rows()), classes)

    // Draw boxes on image
    drawBoxes(&img, detected)

    // Save the image
    gocv.IMWrite("golang_inference.jpg", img)
}
```

Reading the image

```python
# python
# Read the image
original_image: np.ndarray = cv2.imread(input_image)
```

```go
// go
// Read the image
b, err := os.ReadFile(imgPath)
if err != nil {
    panic(err)
}

img, err := gocv.IMDecode(b, gocv.IMReadAnyColor)
if err != nil {
    panic(err)
}
defer img.Close()
```

Prepare the network

Define model input:

```python
# python
net.setInputParams(size=(416, 416), scale=1 / 255, swapRB=True)
```

```go
// go
gocv.BlobFromImage(img, 1/255.0, image.Pt(416, 416), gocv.NewScalar(0, 0, 0, 0), true, false)
```

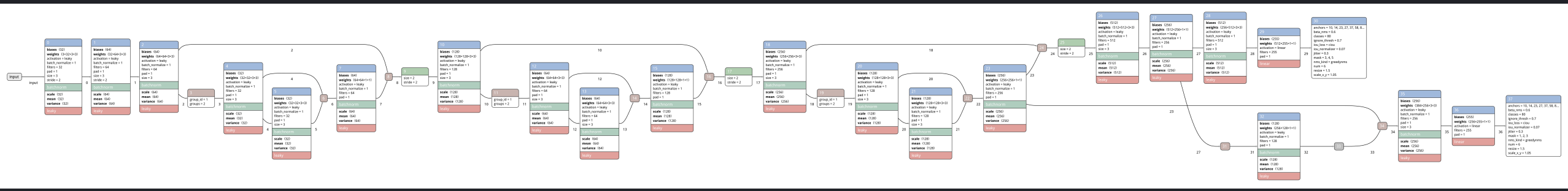
Image, Scale, Size, Mean value to subtract for channels, swapRB, crop

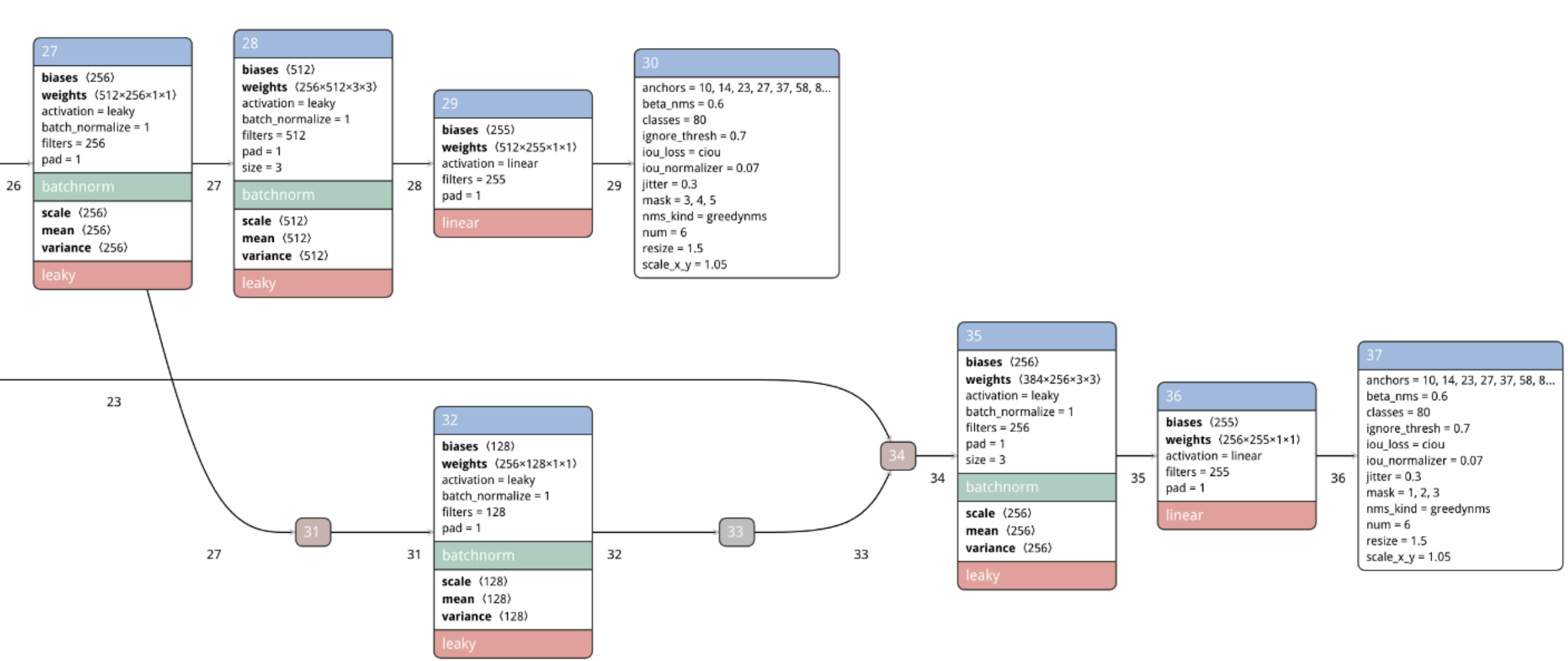
Detect objects

```python
# python
detected_classes, scores, boxes = net.detect(original_image, THRESHOLD, 0.25)
```

```go
// go
// Perform inference
probs := net.ForwardLayers(getOutputLayerNames(&net))

// Postprocess the output
detected := postProcess(probs, 0.4, float32(img.Cols()), float32(img.Rows()), classes)
```

```
net.ForwardLayers(getOutputLayerNames(&net))
```


input
0
biases (32)
weights (3×32×3×3)
activation = leaky
batch_normalize = 1
filters = 32
pad = 1
size = 3
stride = 2
batchnorm
scale (32)
mean (32)
variance (32)
leaky
1
biases (64)
weights (32×64×3×3)
activation = leaky
batch_normalize = 1
filters = 64
pad = 1
size = 3
stride = 2
batchnorm
scale (64)
mean (64)
variance (64)
leaky
2
biases (64)
weights (64×64×3×3)
activation = leaky
batch_normalize = 1
filters = 64
pad = 1
size = 3
batchnorm
scale (64)
mean (64)
variance (64)
leaky
3
group_id = 1
groups = 2
4
biases (32)
weights (32×32×3×3)
activation = leaky
batch_normalize = 1
filters = 32
pad = 1
size = 3
batchnorm
scale (32)
mean (32)
variance (32)
leaky
5
biases (32)
weights (32×32×3×3)
activation = leaky
batch_normalize = 1
filters = 32
pad = 1
size = 3
batchnorm
scale (32)
mean (32)
variance (32)
leaky
6
7
biases (64)
weights (64×64×1×1)
activation = leaky
batch_normalize = 1
filters = 64
pad = 1
batchnorm
scale (64)
mean (64)
variance (64)
leaky
8
size = 2
stride = 2
9
10
biases (128)
weights (128×128×3×3)
activation = leaky
batch_normalize = 1
filters = 128
pad = 1
size = 3
batchnorm
scale (128)
mean (128)
variance (128)
leaky
11
group_id = 1
groups = 2
12
biases (64)
weights (64×64×3×3)
activation = leaky
batch_normalize = 1
filters = 64
pad = 1
size = 3
batchnorm
scale (64)
mean (64)
variance (64)
leaky
13
biases (64)
weights (64×64×3×3)
activation = leaky
batch_normalize = 1
filters = 64
pad = 1
size = 3
batchnorm
scale (64)
mean (64)
variance (64)
leaky
14
15
biases (128)
weights (128×128×1×1)
activation = leaky
batch_normalize = 1
filters = 128
pad = 1
batchnorm
scale (128)
mean (128)
variance (128)
leaky
16
17
size = 2
stride = 2
18
biases (256)
weights (256×256×3×3)
activation = leaky
batch_normalize = 1
filters = 256
pad = 1
size = 3
batchnorm
scale (256)
mean (256)
variance (256)
leaky
19
group_id = 1
groups = 2
20
biases (128)
weights (128×128×3×3)
activation = leaky
batch_normalize = 1
filters = 128
pad = 1
size = 3
batchnorm
scale (128)
mean (128)
variance (128)
leaky
21
biases (128)
weights (128×128×3×3)
activation = leaky
batch_normalize = 1
filters = 128
pad = 1
size = 3
batchnorm
scale (128)
mean (128)
variance (128)
leaky
22
23
biases (256)
weights (256×256×1×1)
activation = leaky
batch_normalize = 1
filters = 256
pad = 1
batchnorm
scale (256)
mean (256)
variance (256)
leaky
24
25
size = 2
stride = 2
26
biases (512)
weights (512×512×3×3)
activation = leaky
batch_normalize = 1
filters = 512
pad = 1
size = 3
batchnorm
scale (512)
mean (512)
variance (512)
leaky
27
biases (256)
weights (512×256×1×1)
activation = leaky
batch_normalize = 1
filters = 256
pad = 1
batchnorm
scale (256)
mean (256)
variance (256)
leaky
28
biases (512)
weights (256×512×3×3)
activation = leaky
batch_normalize = 1
filters = 512
pad = 1
size = 3
batchnorm
scale (512)
mean (512)
variance (512)
leaky
29
biases (255)
weights (512×255×1×1)
activation = linear
filters = 255
pad = 1
linear
30
anchors = 10, 14, 23, 27, 37, 58, 8...
beta_nms = 0.6
classes = 80
ignore_thresh = 0.7
iou_loss = ciou
iou_normalizer = 0.07
jitter = 0.3
mask = 3, 4, 5
nms_kind = greedynms
num = 6
resize = 1.5
scale_x_y = 1.05
31
32
biases (128)
weights (256×128×1×1)
activation = leaky
batch_normalize = 1
filters = 128
pad = 1
batchnorm
scale (128)
mean (128)
variance (128)
leaky
33
34
35
biases (256)
weights (384×256×3×3)
activation = leaky
batch_normalize = 1
filters = 256
pad = 1
size = 3
batchnorm
scale (256)
mean (256)
variance (256)
leaky
36
biases (255)
weights (256×255×1×1)
activation = linear
filters = 255
pad = 1
linear
37
anchors = 10, 14, 23, 27, 37, 58, 8...
beta_nms = 0.6
classes = 80
ignore_thresh = 0.7
iou_loss = ciou
iou_normalizer = 0.07
jitter = 0.3
mask = 1, 2, 3
nms_kind = greedynms
num = 6
resize = 1.5
scale_x_y = 1.05


27
biases ⟨256⟩
weights ⟨512×256×1×1⟩
activation = leaky
batch_normalize = 1
filters = 256
pad = 1
batchnorm
scale ⟨256⟩
mean ⟨256⟩
variance ⟨256⟩
leaky
28
biases ⟨512⟩
weights ⟨256×512×3×3⟩
activation = leaky
batch_normalize = 1
filters = 512
pad = 1
size = 3
batchnorm
scale ⟨512⟩
mean ⟨512⟩
variance ⟨512⟩
leaky
29
biases ⟨255⟩
weights ⟨512×255×1×1⟩
activation = linear
filters = 255
pad = 1
linear
30
anchors = 10, 14, 23, 27, 37, 58, 8...
beta_nms = 0.6
classes = 80
ignore_thresh = 0.7
iou_loss = ciou
iou_normalizer = 0.07
jitter = 0.3
mask = 3, 4, 5
nms_kind = greedynms
num = 6
resize = 1.5
scale_x_y = 1.05
26
27
28
29
23
31
27
31
32
biases ⟨128⟩
weights ⟨256×128×1×1⟩
activation = leaky
batch_normalize = 1
filters = 128
pad = 1
batchnorm
scale ⟨128⟩
mean ⟨128⟩
variance ⟨128⟩
leaky
32
33
33
34
34
35
biases ⟨256⟩
weights ⟨384×256×3×3⟩
activation = leaky
batch_normalize = 1
filters = 256
pad = 1
size = 3
batchnorm
scale ⟨256⟩
mean ⟨256⟩
variance ⟨256⟩
leaky
35
36
biases ⟨255⟩
weights ⟨256×255×1×1⟩
activation = linear
filters = 255
pad = 1
linear
36
37
anchors = 10, 14, 23, 27, 37, 58, 8...
beta_nms = 0.6
classes = 80
ignore_thresh = 0.7
iou_loss = ciou
iou_normalizer = 0.07
jitter = 0.3
mask = 1, 2, 3
nms_kind = greedynms
num = 6
resize = 1.5
scale_x_y = 1.05

```go
// go
// getOutputLayerNames outputs layer names from the Yolo model.
func getOutputLayerNames(net *gocv.Net) []string {
    var outputLayers []string
    for _, i := range net.GetUnconnectedOutLayers() {
        layer := net.GetLayer(i)
        layerName := layer.GetName()
        outputLayers = append(outputLayers, layerName)
    }

    return outputLayers
}
```

Post processing

```
// go
// Postprocess the output
detected := postProcess(probs, 0.4, float32(img.Cols()), float32(img.Rows()), classes)
```

How it works

1. Loop over the output of the model
2. Read and compute fancy values
3. Save it to a structure
4. Clean that structure

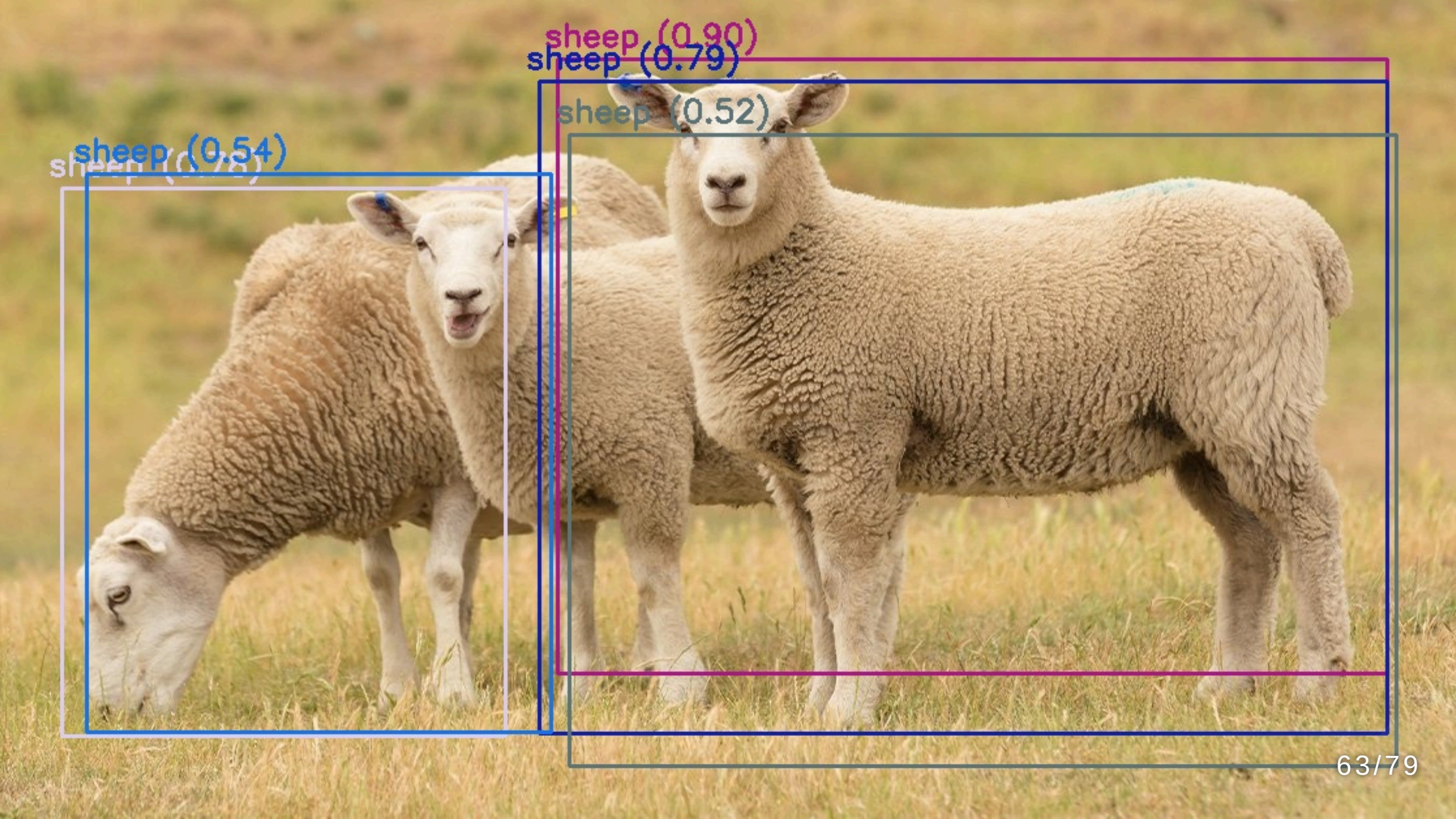
```go
// go
// postProcess processes the output of the Yolo model.
func postProcess(detections []gocv.Mat, nmsThreshold float32, frameWidth, frameHeight float32, netClasses []string) []*DetectedObject {
    var detectedObjects []*DetectedObject
    var boundingBoxes []image.Rectangle
    var accuracyList []float32

    for i, yoloLayer := range detections {
        cols := yoloLayer.Cols()
        data, err := detections[i].DataPtrFloat32()
        if err != nil {
            panic(err)
        }

        // Iterate Yolo results
        for j := 0; j < yoloLayer.Total(); j += cols {
            row := data[j : j+cols]
            scores := row[5:]

            classID, accuracy := getClassIDAndAccuracy(scores)
            className := netClasses[classID]

            // Remove the bounding boxes with low accuracy
            if accuracy > threshold {
                // Calculate bounding box
                boundingBox := calculateBoundingBox(frameWidth, frameHeight, row)

                // Append data to the lists
                accuracyList = append(accuracyList, accuracy)
                boundingBoxes = append(boundingBoxes, boundingBox)
                detectedObjects = append(detectedObjects, &DetectedObject{
                    Rect:      boundingBox,
                    ClassName: className,
                    ClassID:   classID,
                    Accuracy:  accuracy,
                })
            }
        }
    }

    return something
}
```

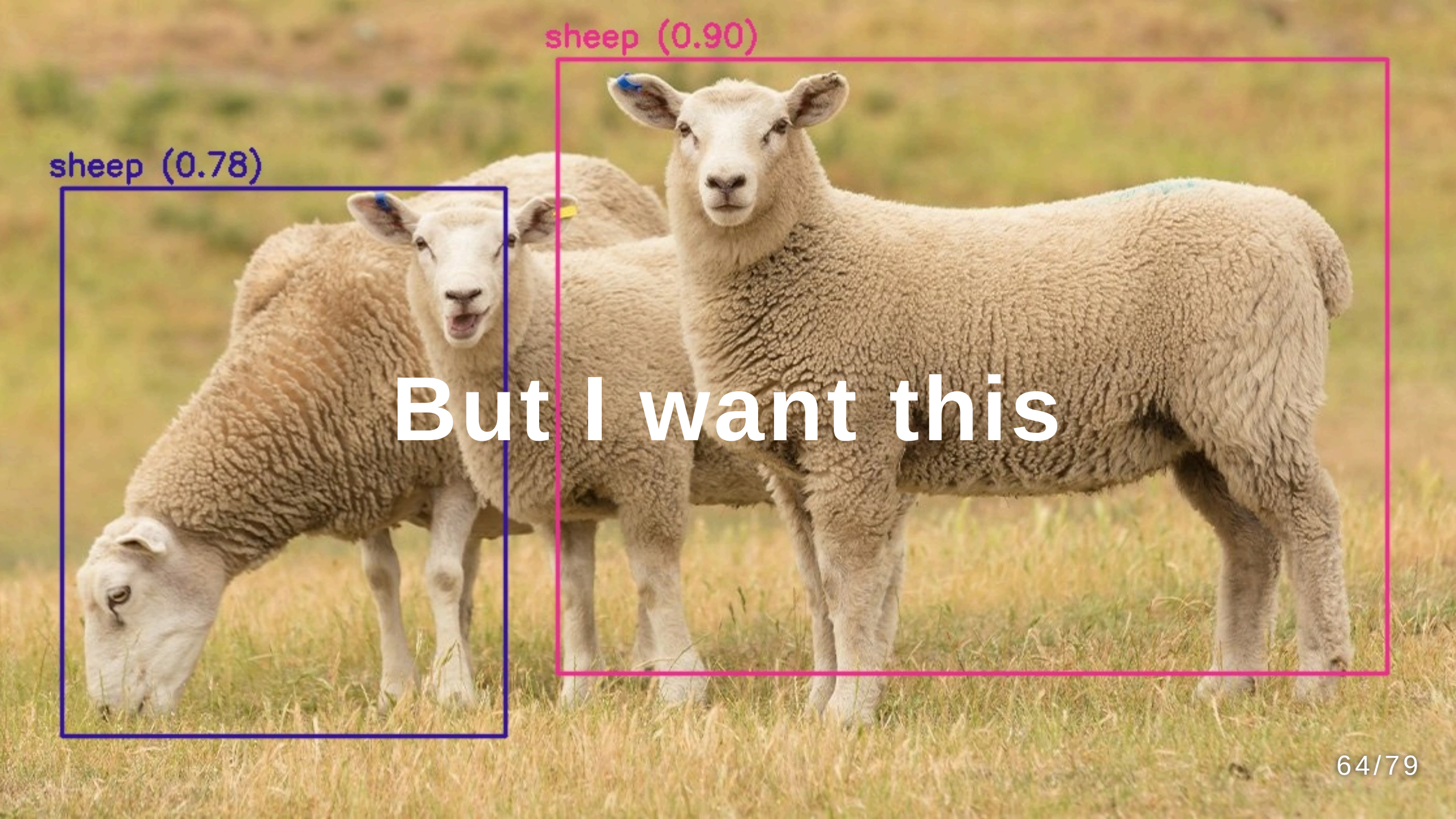
```go
// go
// ...
for i, yoloLayer := range detections {
    cols := yoloLayer.Cols()
    data, err := detections[i].DataPtrFloat32()
    if err != nil {
        panic(err)
    }
// ...
```

```
// go
// ...
// Iterate Yolo results
for j := 0; j < yoloLayer.Total(); j += cols {
    row := data[j : j+cols]
    scores := row[5:]

    classID, accuracy := getClassIDAndAccuracy(scores)
    className := netClasses[classID]
// ...
```

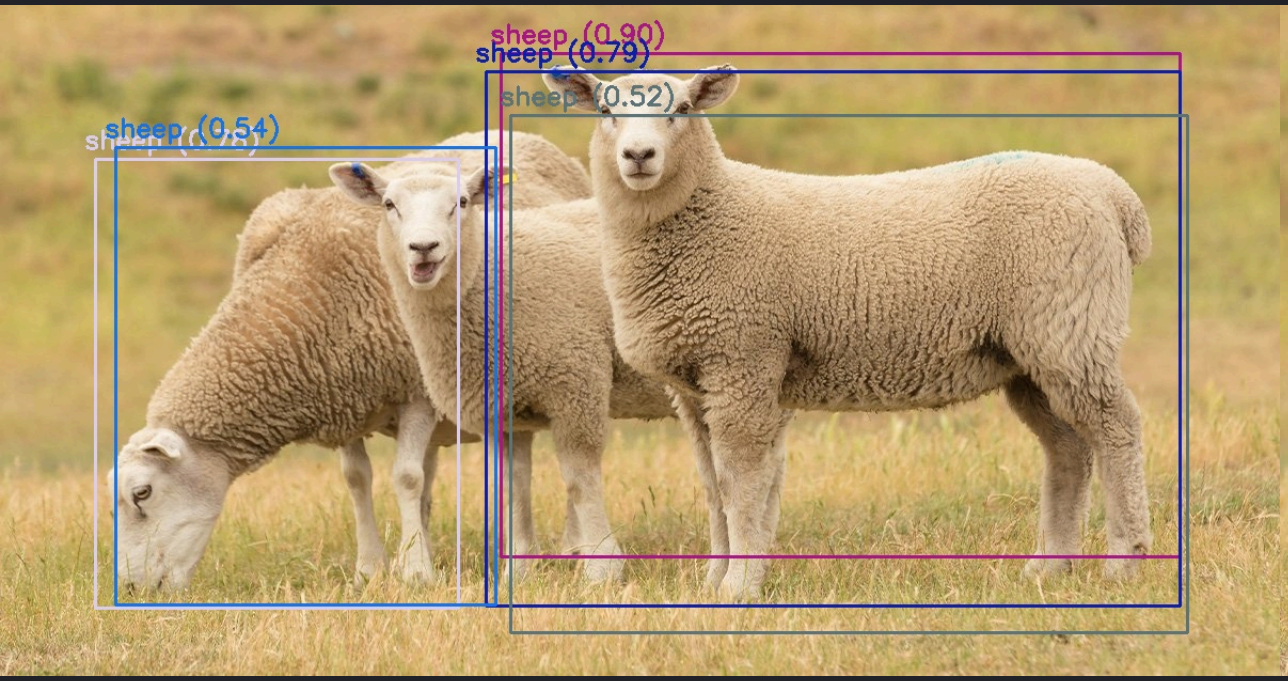
```go
// go
// Remove the bounding boxes with low accuracy
if accuracy > threshold {
    // Calculate bounding box
    boundingBox := calculateBoundingBox(frameWidth, frameHeight, row)

    // Append data to the lists
    accuracyList = append(accuracyList, accuracy)
    boundingBoxes = append(boundingBoxes, boundingBox)
    detectedObjects = append(detectedObjects, &DetectedObject{
        Rect:      boundingBox,
        ClassName: className,
        ClassID:   classID,
        Accuracy:  accuracy,
    })
}
```

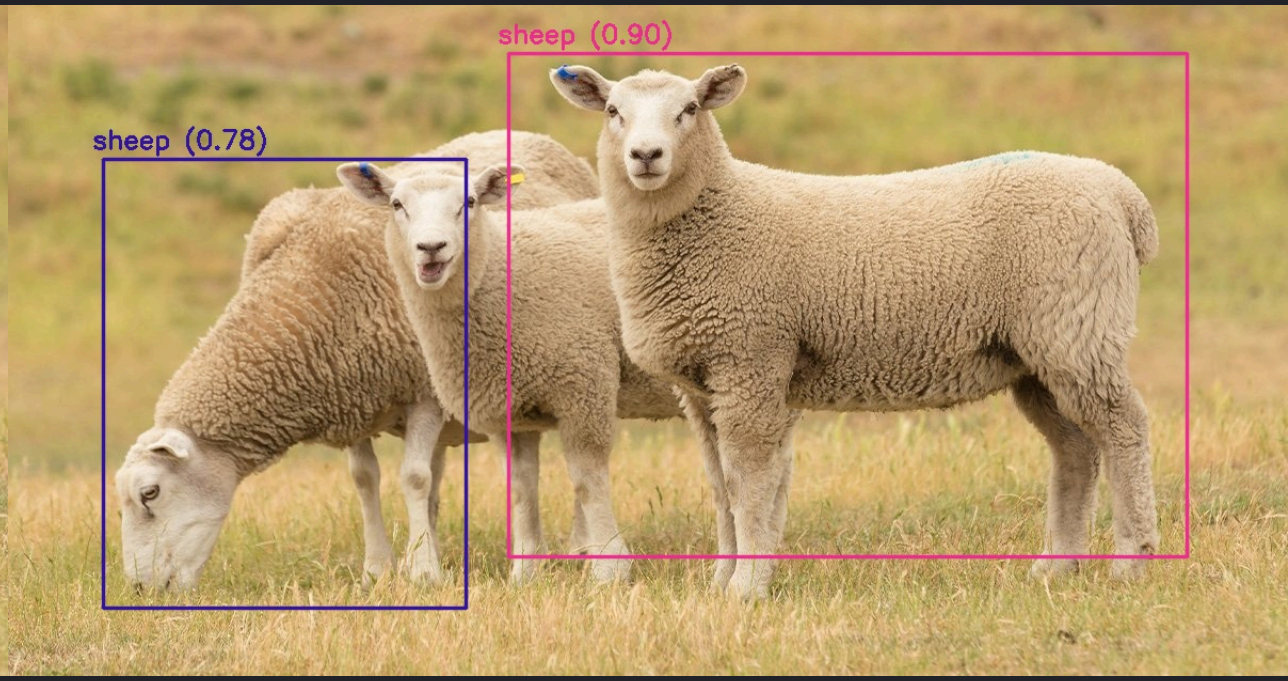
Working version


sheep (0.90)
sheep (0.79)
sheep (0.52)
sheep (0.54)
63/79


sheep (0.90)
sheep (0.78)
But I want this

Non-maximum Suppression (NMS)

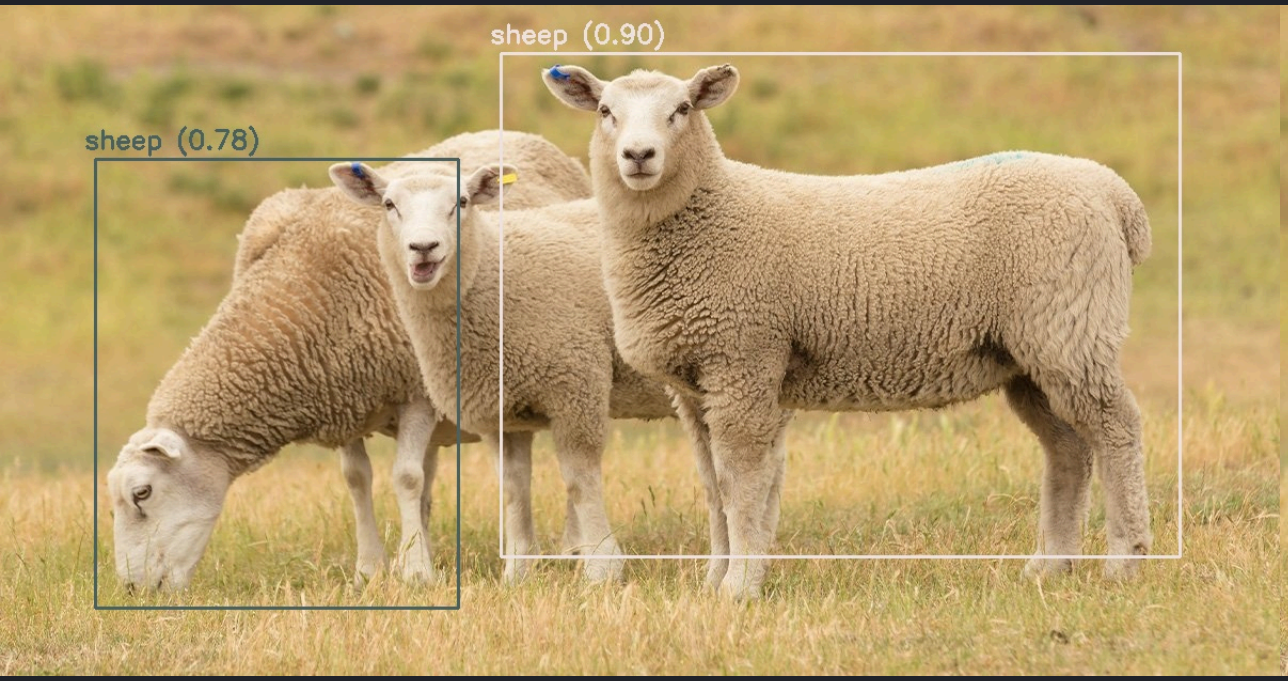

sheep (0.90)
sheep (0.79)
sheep (0.52)
sheep (0.54)

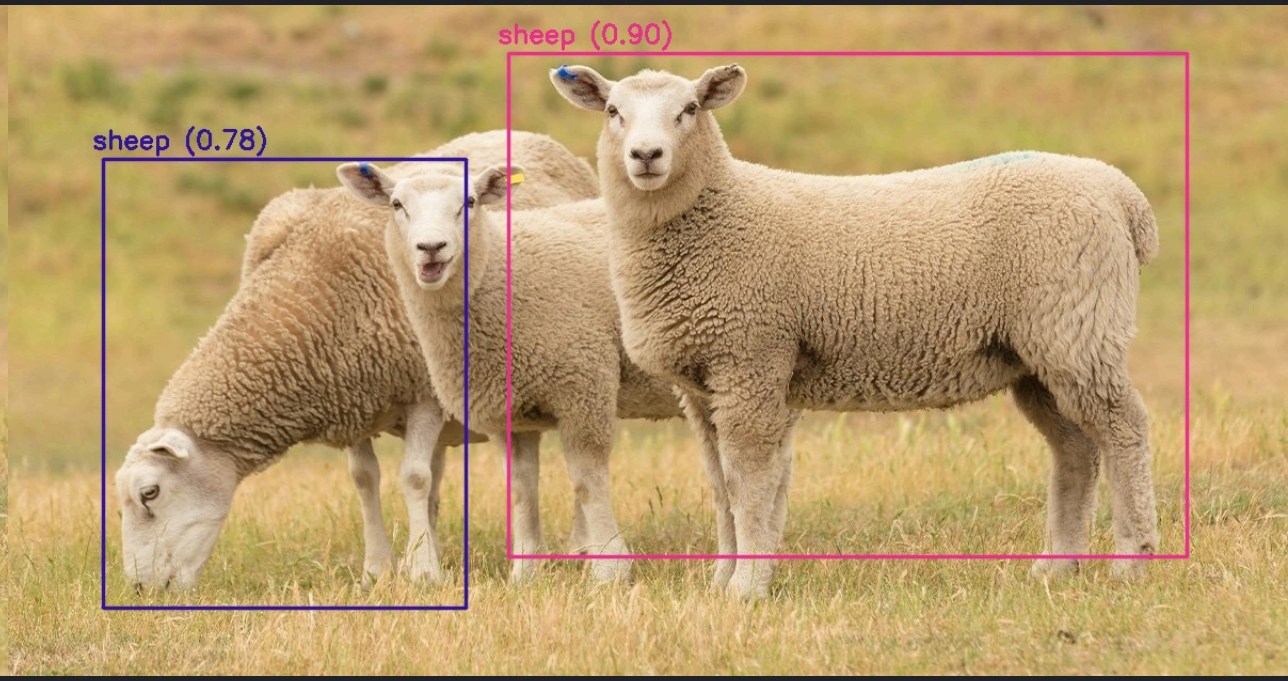

sheep (0.90)
sheep (0.78)

```python
# python
detected_classes, scores, boxes = net.detect(original_image, THRESHOLD, 0.25)
```

```python
# python
nmsThreshold: float = 0.25
```

```go
// go
indices := make([]int, len(boundingBoxes))
for i := range indices {
    indices[i] = -1
}

// Apply non-maximum suppression
indices = gocv.NMSBoxes(boundingBoxes, accuracyList, threshold, nmsThreshold)
filteredDetectedObjects := make([]*DetectedObject, 0, len(detectedObjects))
for i, idx := range indices {
    if idx < 0 || (i != 0 && idx == 0) {
        // Eliminate zeros, since they are filtered by NMS (except first element)
        // Also filter all '-1' which are undefined by default
        continue
    }
    filteredDetectedObjects = append(filteredDetectedObjects, detectedObjects[idx])
}
```

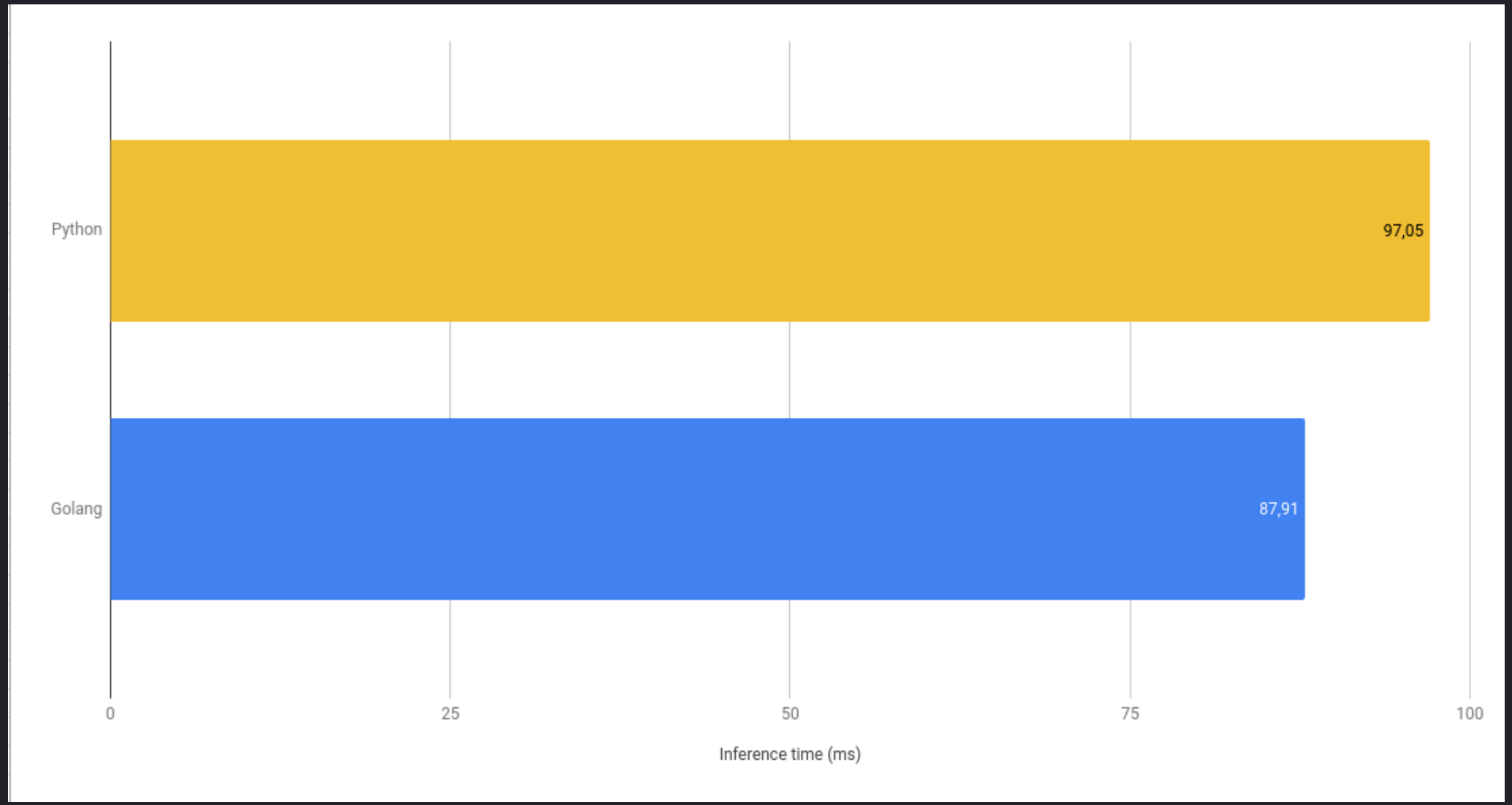

sheep (0.90)
sheep (0.78)


sheep (0.90)
sheep (0.78)

Results

Average inference time in Python: 97.05 ms

Average inference time in Golang: 87.91 ms


Python
97,05
Golang
87,91
0
25
50
75
100
Inference time (ms)

~11 % inference time increase

Key takeaways

Have fun with your code

Don't tunnel vision, for our use case, Golang was faster but **it is not** always the case

Thanks for listening

Questions?

Thank You!

👨‍💻 Armand Brière
https://www.linkedin.com/in/armand-briere/

/ OSEDEA – **Projects** – **Careers**
osedea.com