

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



Objectif of the presentation

Go
Programming language

Python
Programming language

+ Add comparison

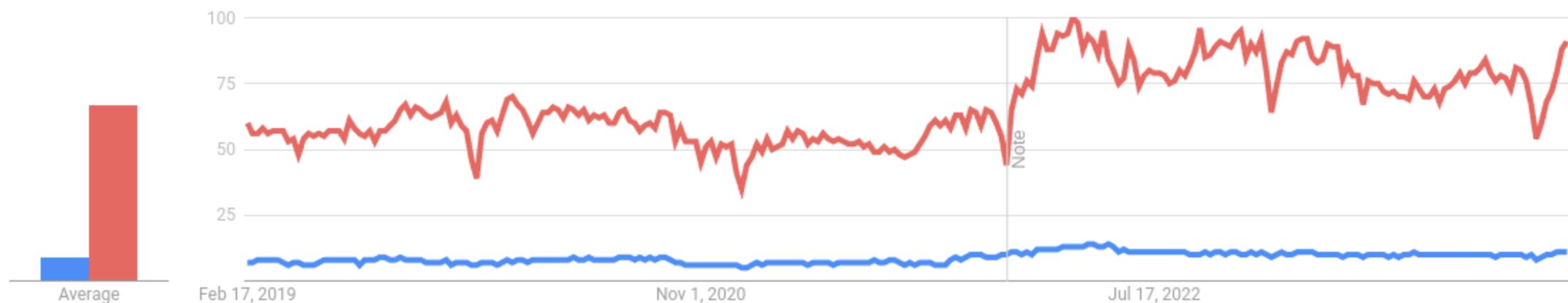
Worldwide ▾

Past 5 years ▾

All categories ▾

Web Search ▾

Interest over time ⓘ



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 Confoo

```
# python
print("Hello Confoo!")
```

```
// go
package main

import "fmt"

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

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

Functions

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

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

Bonus points

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

```
# confoo24.inference/playground
./playground.go:23:6: z declared and not used
```

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

Let's dive in

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

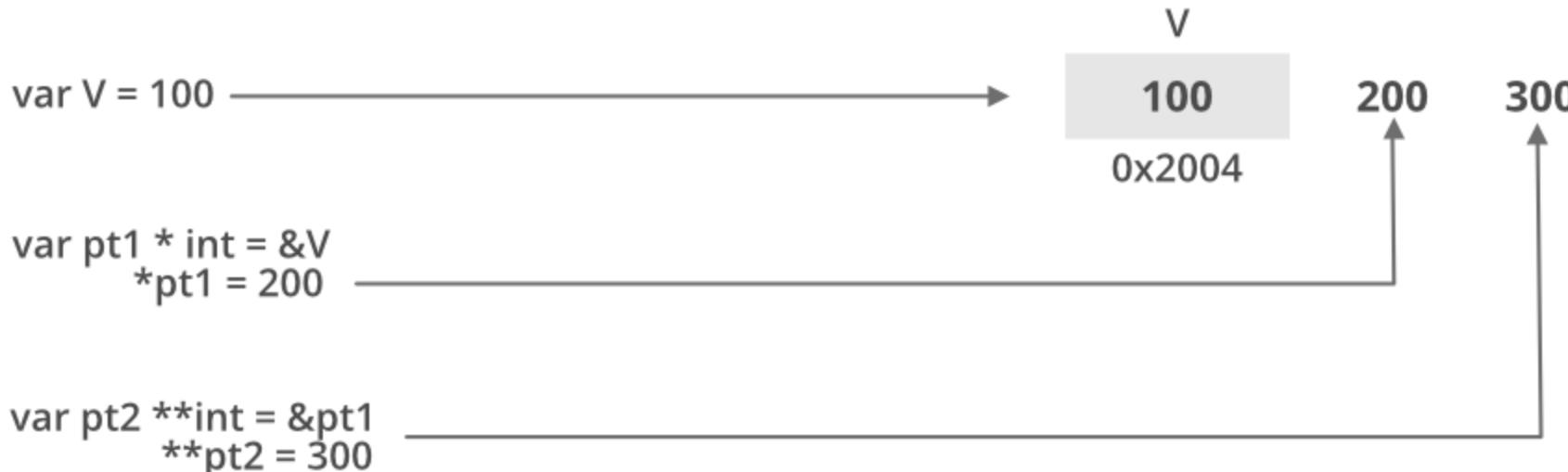
    // Load the model and classes
    netModel := loadModel("yolov4-tiny.weights", "yolov4-tiny.cfg")
    classes := loadYAML("classes.yaml")

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

What is `*img` ?

Oh that's a pointer...

How Pointers works in Go



DE

int *

int



```
// 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

What is *img ?

*img is just the file name

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

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

Load model

```
# 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(yolo modelName string, yolo ModelCfg string) gocv.Net {
    net := gocv.ReadNet(yolo modelName, yolo ModelCfg)

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

    return net
}
```



```
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





Load yaml

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

```
// 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](#)

A photograph of a person sitting by a campfire at night. The campfire is bright orange and yellow, casting light on the surrounding dark trees and grass. In the background, a city skyline is visible under a dark sky with some stars. The word "Checkpoint" is overlaid in white text.

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

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

```
// 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

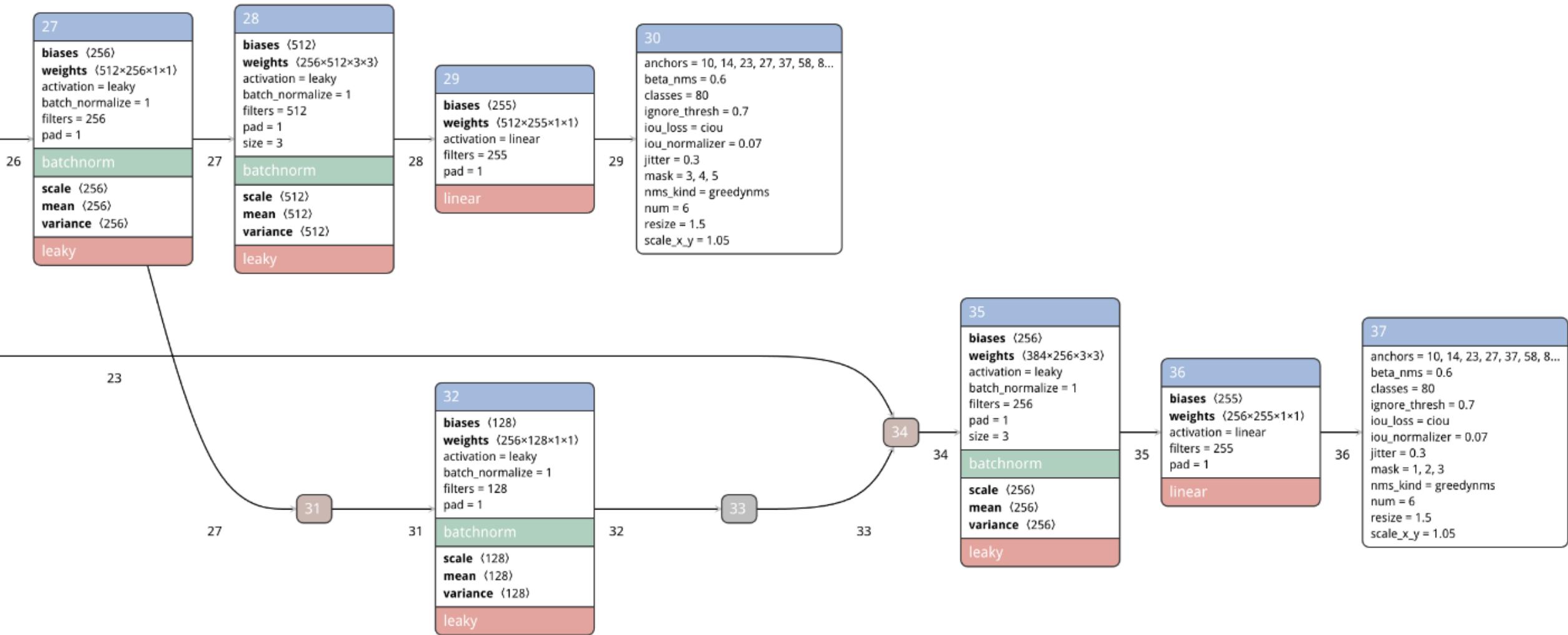
```
# python  
detected_classes, scores, boxes = net.detect(original_image, THRESHOLD, 0.25)
```

```
// 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(getOutputLa  
yerNames(&net))
```







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

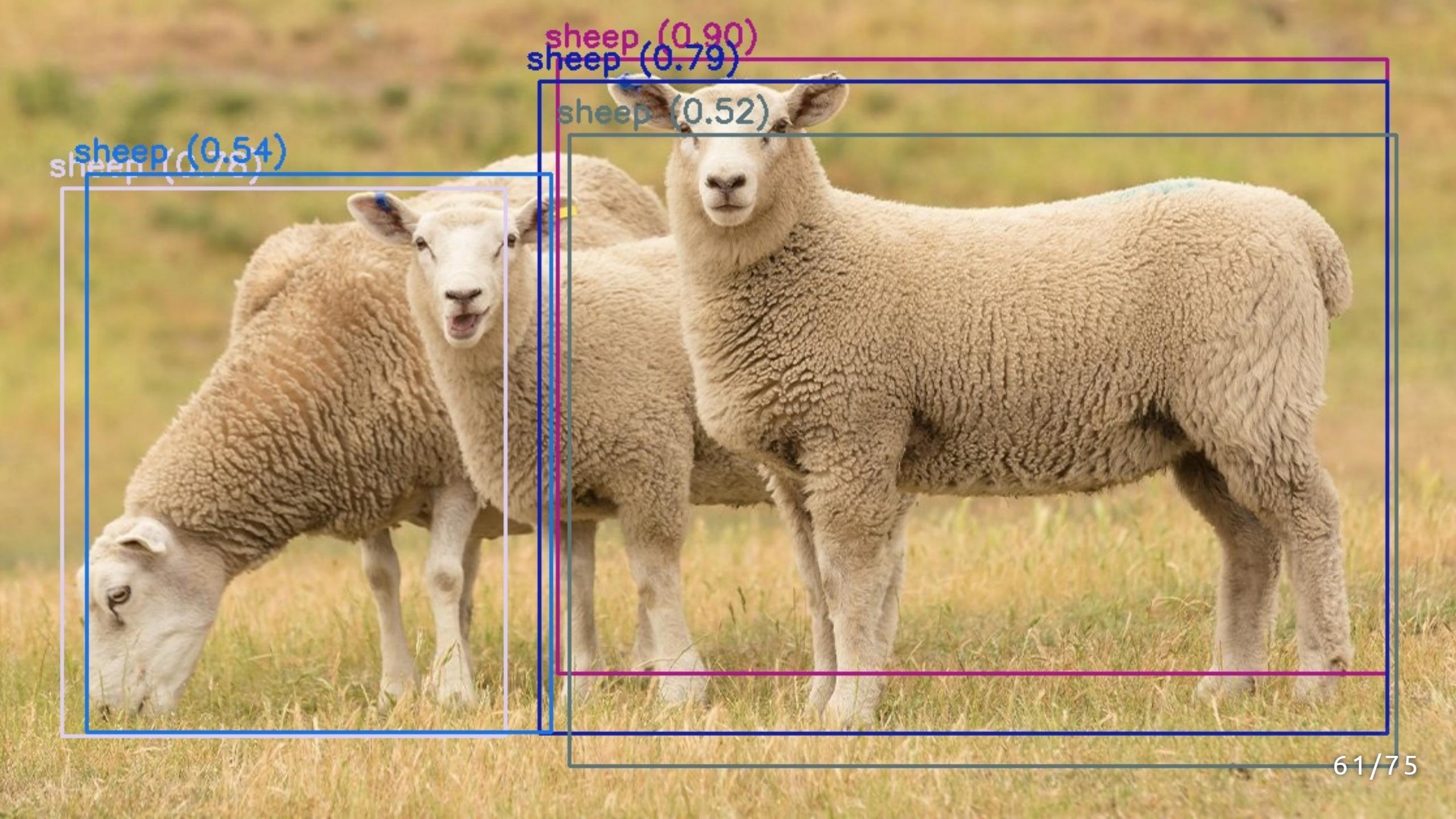
```
// go
// ...
// Iterate Yolo results
for j := 0; j < yoloLayer.Total(); j += cols {
    row := data[j : j+cols]
    scores := row[5:]

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

```
// go
// 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



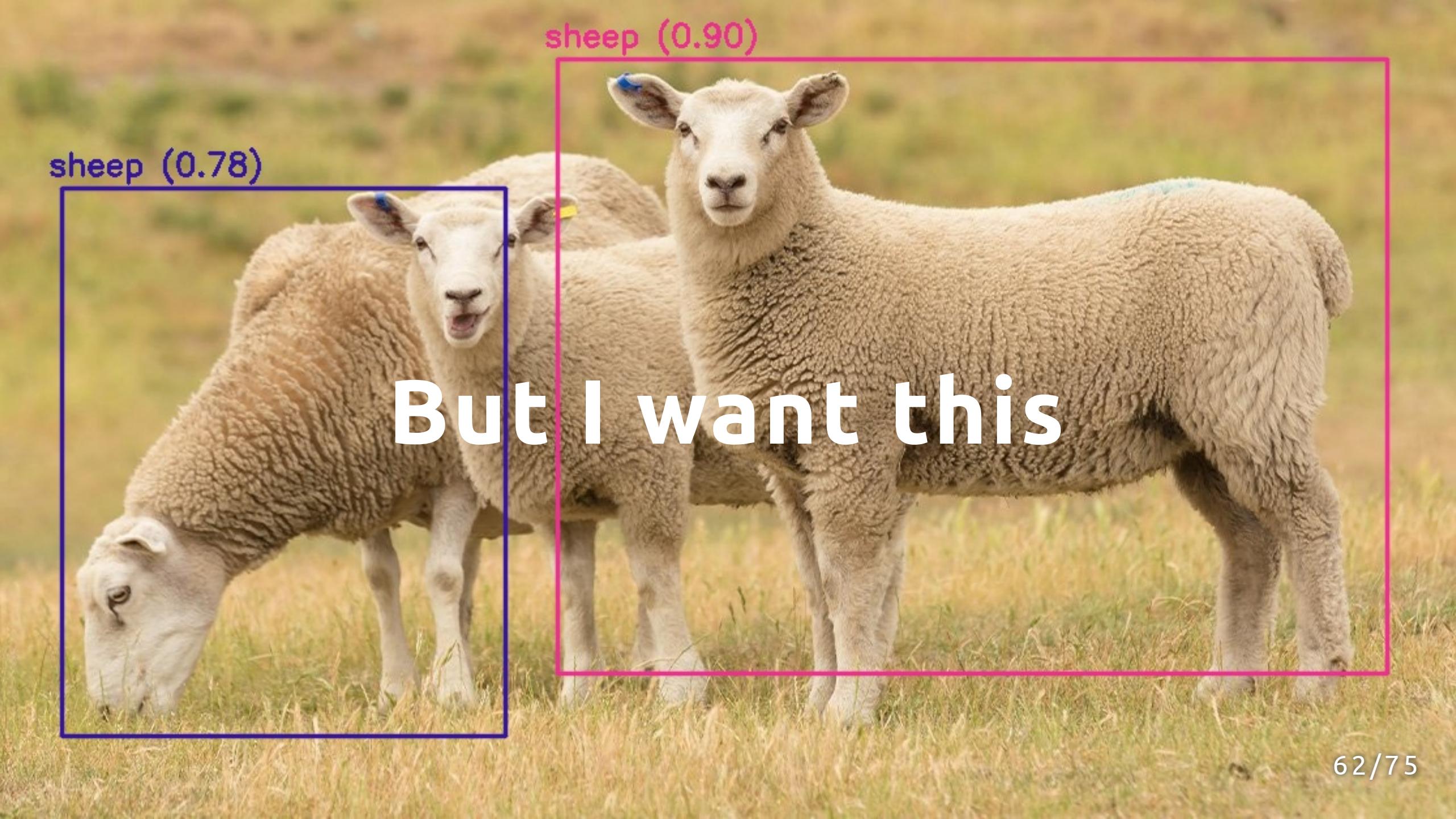
sheep (0.54)

sheep (0.78)

sheep (0.90)

sheep (0.79)

sheep (0.52)

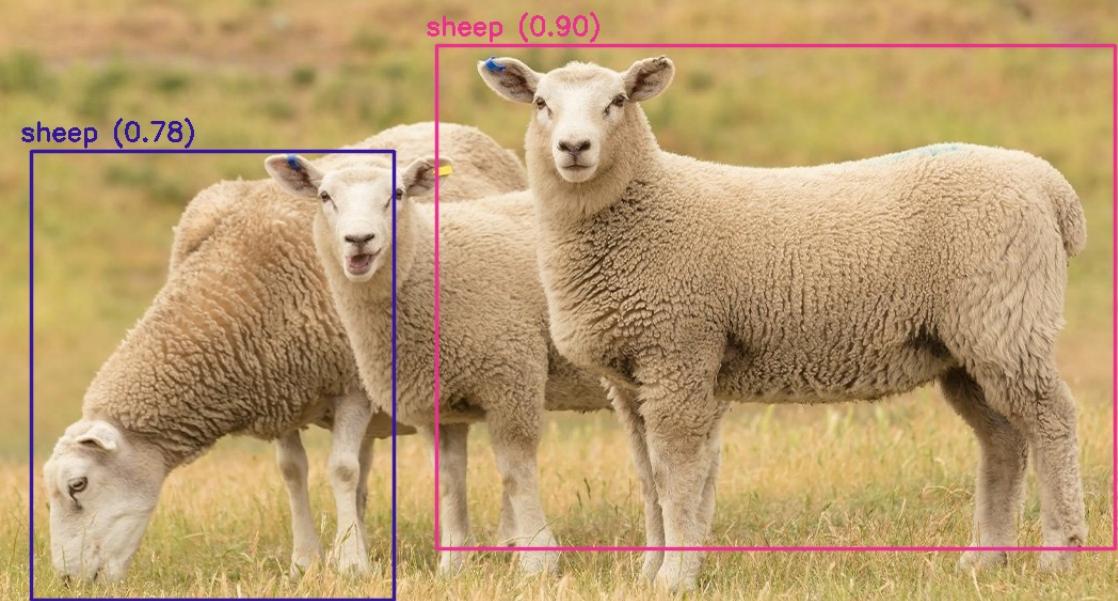
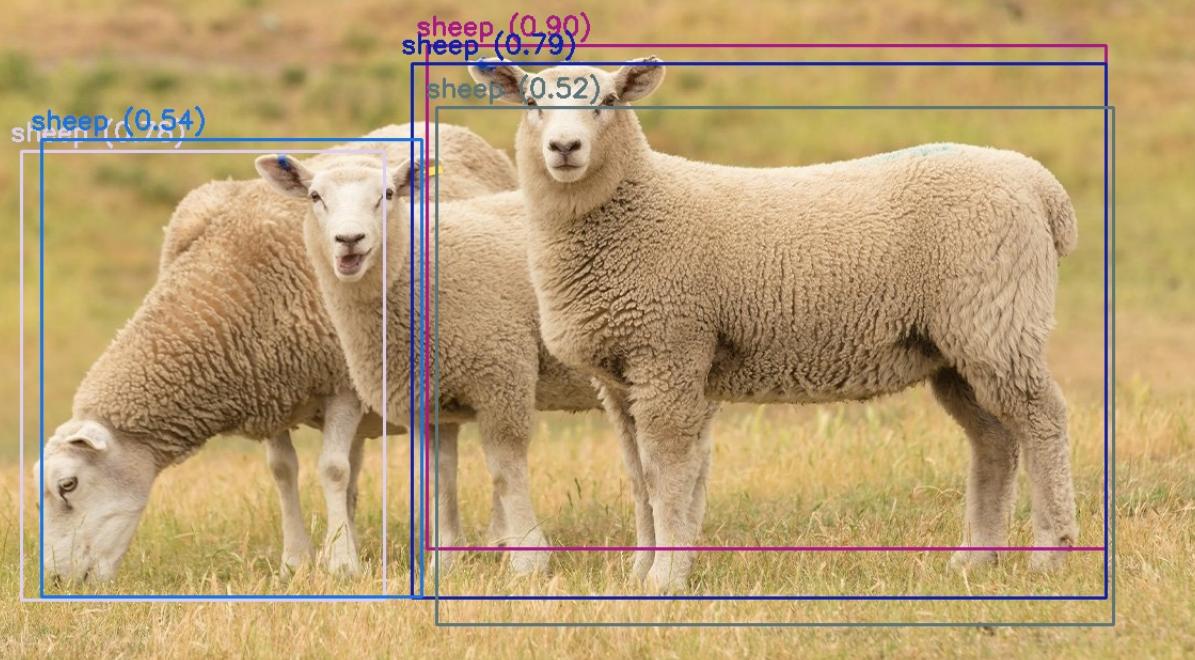


sheep (0.78)

sheep (0.90)

But I want this

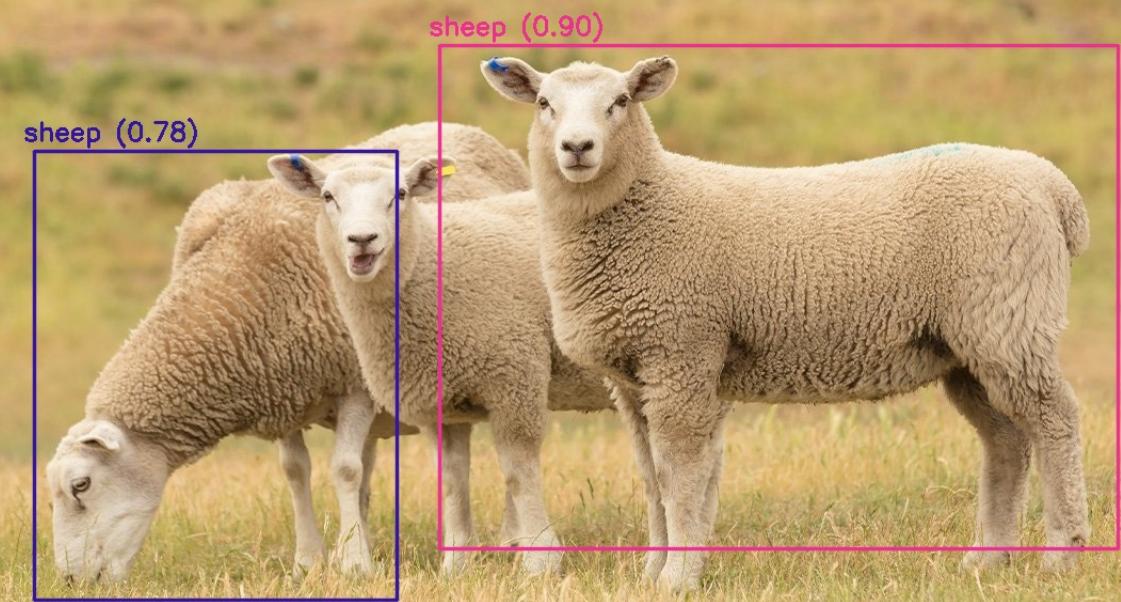
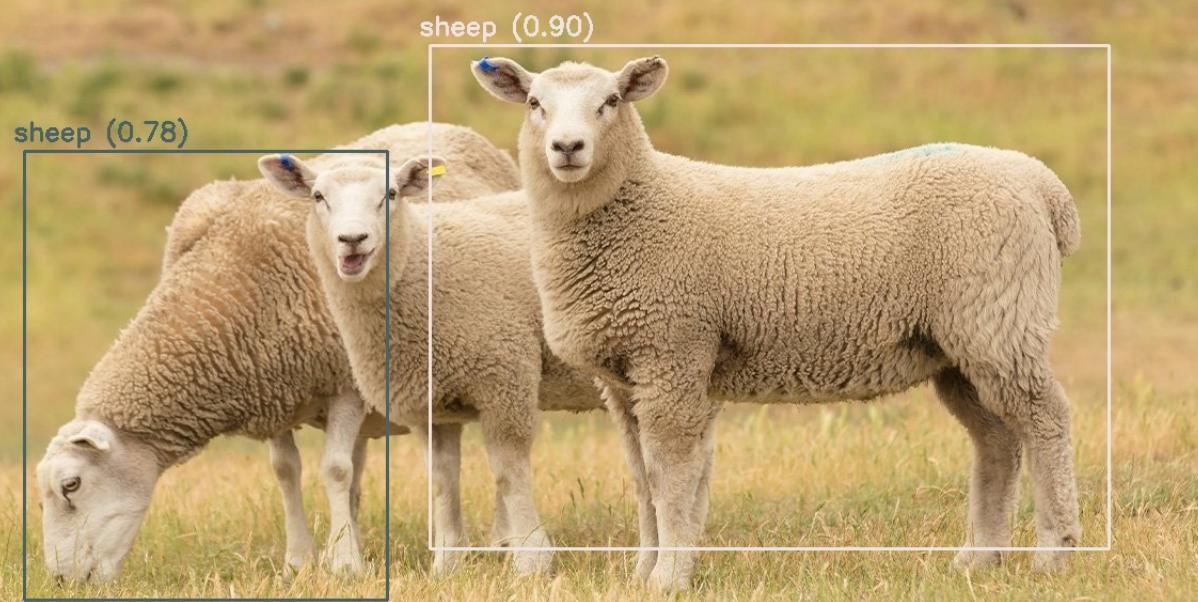
Non-maximum Suppression (NMS)



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

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

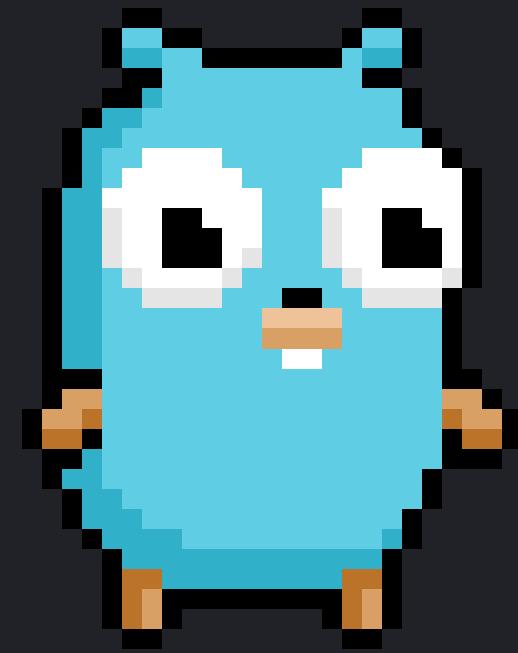
```
// go  
indices := make([]int, len(boundingBoxes))  
for i := range indices {  
    indices[i] = -1  
}  
  
// Apply non-maximum suppression  
indices = gocv.NMSBoxes(boundingBoxes, accuracyList, threshold, nmsThreshold)  
filteredDetectedObjects := make([]*DetectedObject, 0, len(detectedObjects))  
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])  
}
```



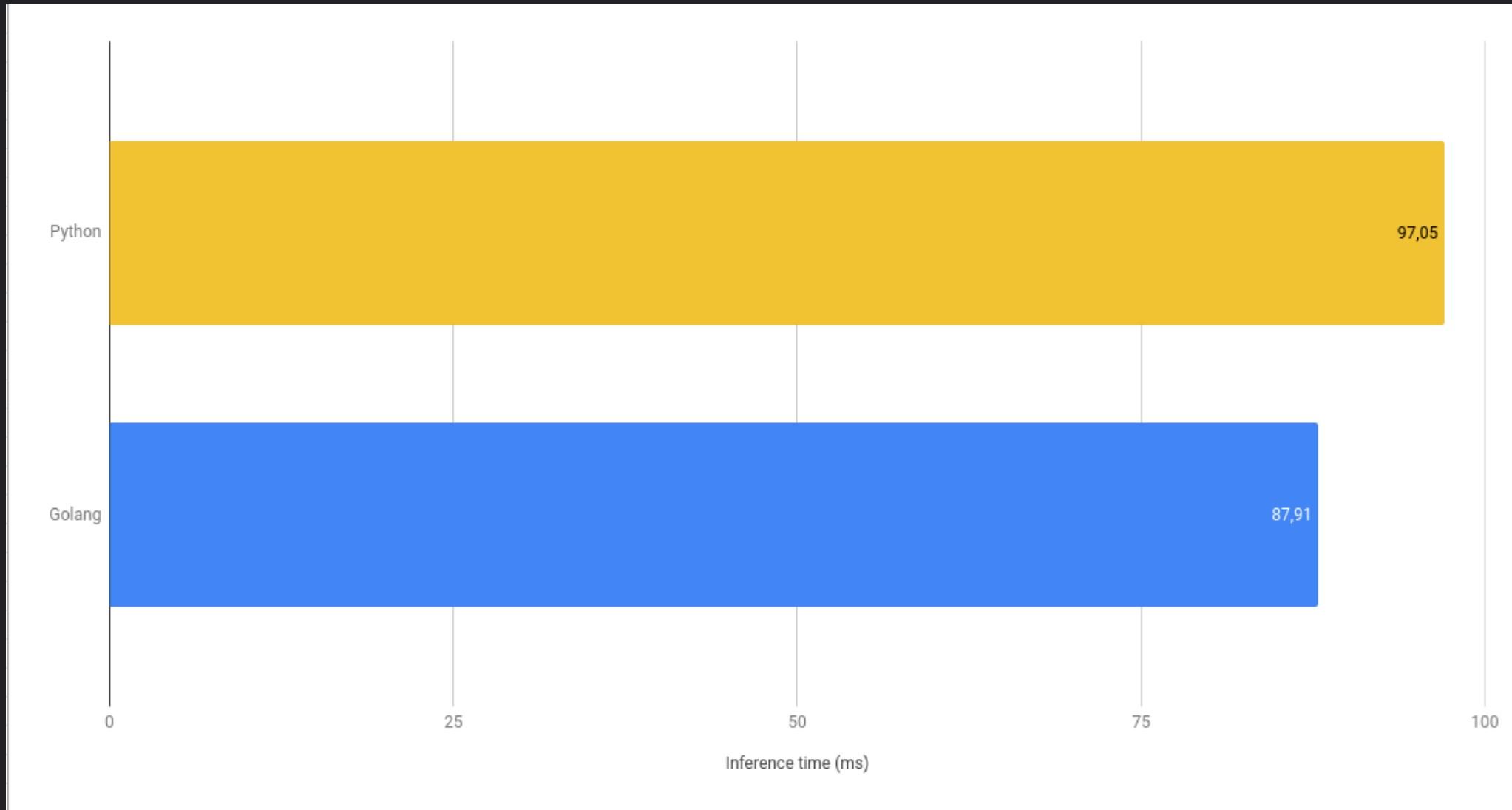
Results



Average inference time in Python: 97.05 ms

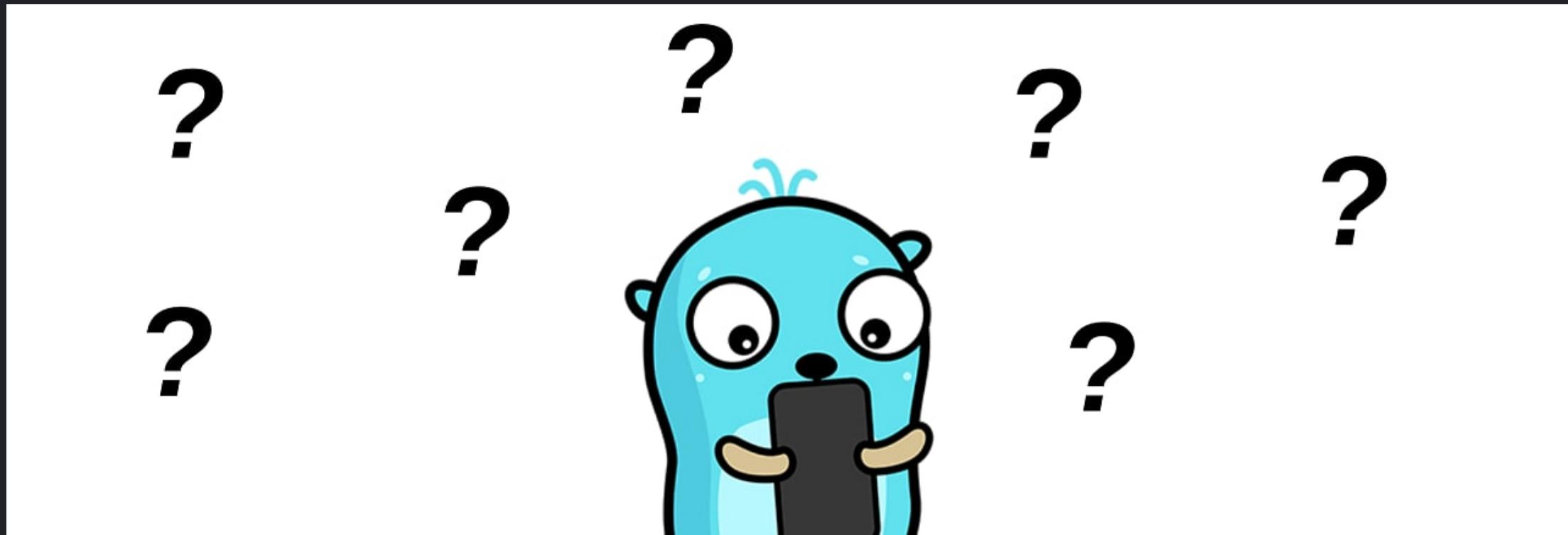


Average inference time in Golang: 87.91 ms



~11 % inference time increase

Questions?



<https://github.com/ArmandBriere/confoo2024>

Thank You!



Armand Brière
<https://www.linkedin.com/in/armand-briere/>

/ OSEDEA – Projects – Careers
osedea.com

