

# Face detection in Go and Webassembly

Endre Simo

<https://esimov.com>

<https://github.com/esimov>

[https://twitter/simo\\_endre](https://twitter/simo_endre)

April 29, 2020

# Agenda

- ▶ What is Pigo?
- ▶ Key features
- ▶ Technical overview
- ▶ Pigo and GoCV (OpenCV) comparision
- ▶ Use cases and integrations
- ▶ Pupils/eyes localization
- ▶ Facial landmark points detection
- ▶ Pigo as a shared library
- ▶ Porting Pigo to Webassembly (WASM)
- ▶ Demo time

# What is Pigo?

## What is PiGo?



- ▶ Computer vision and machine learning library for face detection, pupils/eyes localization and facial landmark points detection
- ▶ The only face detection library in the Go ecosystem developed 100% in Go
- ▶ The implementation is based on *Pixel Intensity Comparison-based Object detection* paper

## Why it has been developed?

- ▶ Installing OpenCV sometimes can be daunting
- ▶ OpenCV is huge, impossible to deploy it on small platforms where space constraints are important

## Key features

What are the benefits of using Pigo over other existing solutions? Just to name a few of them:

- ▶ Very lightweight, no requirements for 3rd party modules and external libraries
- ▶ Platform independent, one single executable
- ▶ Simple and elegant API
- ▶ High processing speed
- ▶ There is no need for image preprocessing prior detection
- ▶ The face detection is based on pixel intensity comparison encoded in the binary file tree structure
- ▶ Fast detection of in-plane rotated faces
- ▶ Pupils/eyes localization
- ▶ Facial landmark points detection

# Technical overview

- ▶ **Pigo**, like the **Viola Jones** face detection algorithm is also constructed around cascade decision trees, but the cascade classifier is in binary format
- ▶ The role of a classifier is to tell if a face is present in the current region or not
- ▶ The classifier consists of a decision tree, where the results of pixel intensity comparison test are in binary format.
- ▶ Because the cascades are encoded into a binary tree structure they first need to be unpacked.

## Upacking steps

- ▶ Read the depth of each tree and write it into the buffer array.
- ▶ Retrieve the number of stages and write it into the buffer array.
- ▶ Obtain the scale multiplier (applied after each stage) and write it into the buffer array.
- ▶ Obtain the number of trees per stage and write it into the buffer array.
- ▶ Obtain the depth of each tree and write it into the buffer array.
- ▶ Traverse all the stages of the binary tree
- ▶ Read prediction from tree's leaf nodes.

## Unpacking the result

In the end we should get a cascade with the following structure:

```
return &PuplocCascade{  
    stages:    stages,  
    scales:    scales,  
    trees:     trees,  
    treeDepth: treeDepth,  
    treeCodes: treeCodes,  
    treePreds: treePreds,  
, nil
```

## Classify regions

Next we classify the regions based on the parsed binary data.

For classification we are using a simple pixel intensity comparision test in binary format.

```
bintest := func(px1, px2 uint8) int {
    if px1 <= px2 {
        return 1
    }
    return 0
}
idx = 2*idx + bintest(pixels[x1], pixels[x2])
```

The `idx` will be the index in the prediction tree.

Note: for in plane rotated faces we are applying the same formula only that we are calculating the rotation angle.

## Run cascade

- ▶ An image region is considered being face if it passes all the cascade members. Since this process is limited to a relatively small number of regions, this gains high computation speed.
- ▶ During the decision tree scanning each detection is flagged with a detection score.
- ▶ An image region is considered as face if the detection score is above a certain threshold ( $\sim 0.995$ )
- ▶ The detector function will return a struct with the following structure

```
// Detection struct contains the detection results composed of
// the row, column, scale factor and the detection score.
type Detection struct {
    Row    int
    Col    int
    Scale  int
    Q      float32
}
```

## Cluster detection

- ▶ Due to the noisiness of the underlying pixel data, the detector might produce overlaps in detections.



- ▶ The cascade regions are clustered together by applying an **IoU** (Intersection over Union) formula over the detection results.

## The intersection over union method

```
sort.Sort(det(detections))

calcIoU := func(det1, det2 Detection) float64 {
    // Unpack the position and size of each detection.
    r1, c1, s1 := float64(det1.Row), float64(det1.Col), float64(det1.Sc
    r2, c2, s2 := float64(det2.Row), float64(det2.Col), float64(det2.Sc

    overRow := math.Max(0, math.Min(r1+s1/2, r2+s2/2)-math.Max(r1-s1/2,
    overCol := math.Max(0, math.Min(c1+s1/2, c2+s2/2)-math.Max(c1-s1/2,

    // Return intersection over union.
    return overRow * overCol / (s1*s1 + s2*s2 - overRow*overCol)
}
```

## End result



## Pigo and GoCV (OpenCV) comparision

## Benchmark results

BenchmarkGoCV-4	3	414122553 ns/op	704 B/op
BenchmarkPIGO-4	10	173664832 ns/op	0 B/op
PASS			
ok	github.com/esimov/gocv-test	4.530s	

[github.com/esimov/pigo-gocv-benchmark](https://github.com/esimov/pigo-gocv-benchmark)

## Use cases and integrations

# OpenFaaS integration

OpenFaaS Portal

Deploy New Function

Search for Function

pigo-face-detector

Status	Replicas	Invocation count
Ready	1	1

Image: esimov/openfaas-pigo:latest  
Function process: /handler  
URL: http://127.0.0.1:8080/function/pigo-face-detector

Invoke function

INVOKE

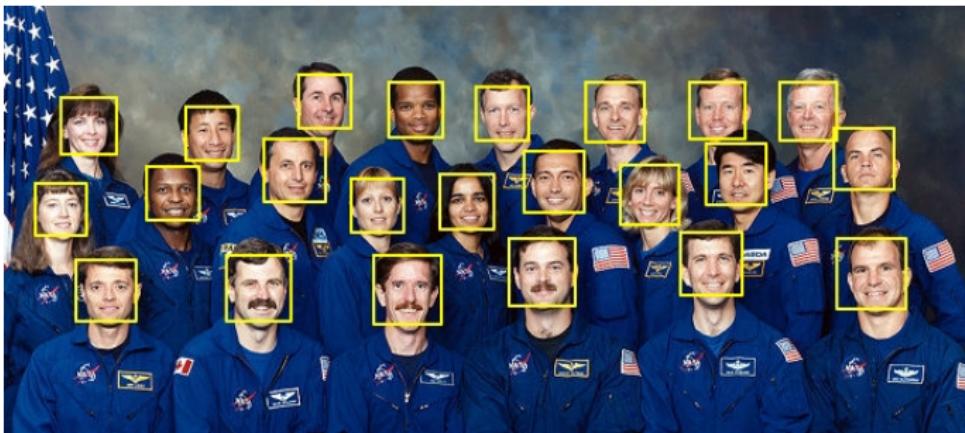
Text  JSON  Download

Request body:  
<http://www.balkaninsight.com/uploads/1/images/2016-06-25/e2066a86e25a978d5ab9f7a00b8db076.jpg>

Response status: 200 Round-trip (s): 3.188

Response body: 135882 byte(s) received

# OpenFaaS integration



[github.com/esimov/pigo-openfaas](https://github.com/esimov/pigo-openfaas)

# OpenFaaS intergation

OpenFaaS Portal

Deploy New Function

Search for Function

pigo-faceblur

Status: Ready    Replicas: 1    Invocation count: 19

Image: esimov/pigo-openfaas-faceblur:0.1    URL: http://127.0.0.1:8080/function/pigo-faceblur

Function process: /handler

---

Invoke function

**INVOKE**

Text  JSON  Download

Request body:  
<https://raw.githubusercontent.com/esimov/pigo-openfaas/master/pigo-openfaas/samples/nasa.jpg>

Response status: 200    Round-trip (s): 1.472

Response body: 526224 byte(s) received

---

pigo-face-detector

caire-image-resizer

face-detect-opencv

colorise

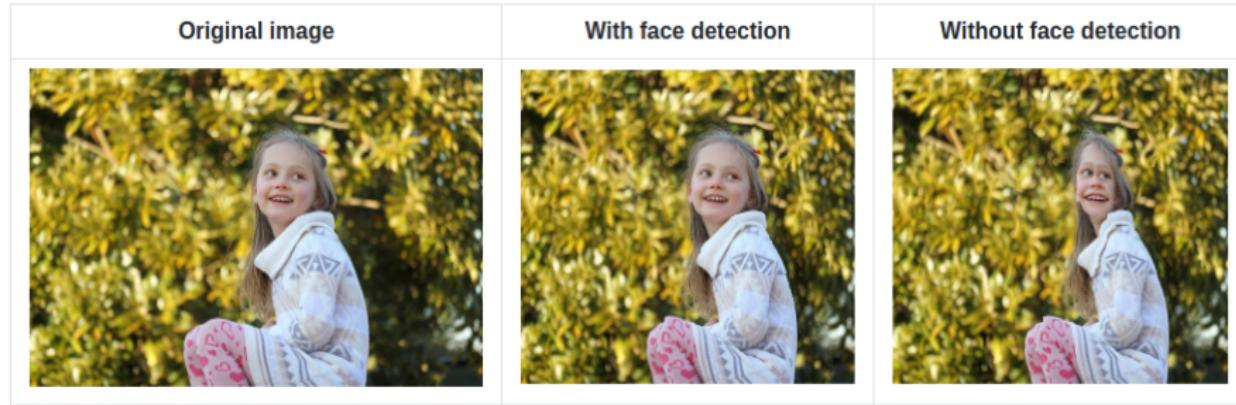
face-detect-pigo

# OpenFaaS integration



[github.com/esimov/pigo-openfaas-faceblur](https://github.com/esimov/pigo-openfaas-faceblur)  
[github.com/esimov/stackblur-go](https://github.com/esimov/stackblur-go)

## Other integration



Avoiding face deformation in Caire  
[github.com/esimov/caire](https://github.com/esimov/caire)

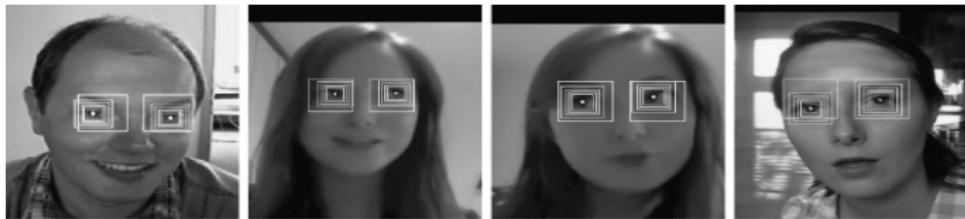
## Pupils/eyes localization

## Pupils/eyes localization



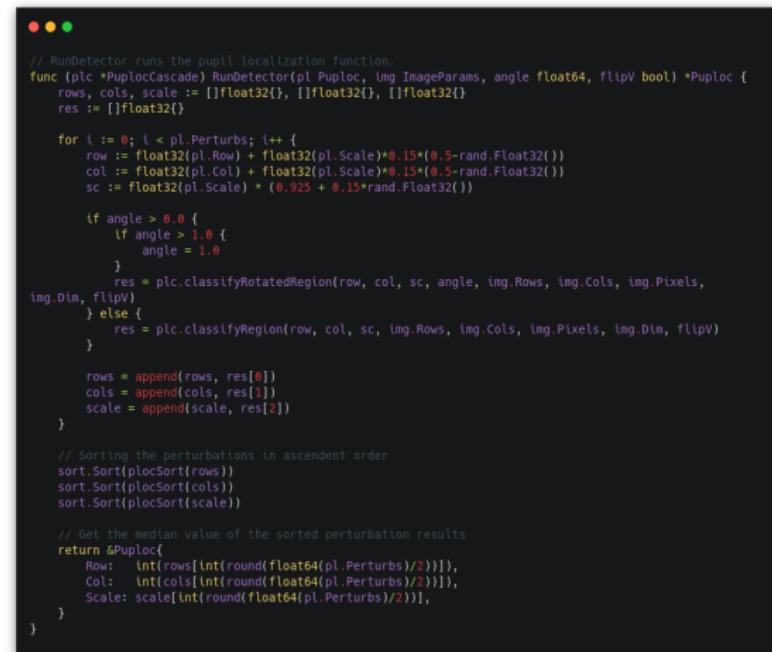
## Short overview

- ▶ The implementation pretty much resambles with the face detection method but with few remarkable differences.
- ▶ The detection method (as the face detection) is based on tree ensembles.
- ▶ As on Pigo face detection, the detection is based on the same pixel intensity binary test.
- ▶ The output of the regression trees might be noisy
- ▶ We introduce a random perturbation factor during runtime to outweigh the false positive rates on detection



## Sort the perturbations

After the face region has been classified we sort the perturbations in ascendant order.



```
// RunDetector runs the pupll localization function.
func (plc *PuplocCascade) RunDetector(pl Puploc, img ImageParams, angle float64, flipV bool) *Puploc {
    rows, cols, scale := []float32{}, []float32{}, []float32{}
    res := []float32{}

    for i := 0; i < pl.Perturbs; i++ {
        row := float32(pl.Row) + float32(pl.Scale)*0.15*(0.5-rand.Float32())
        col := float32(pl.Col) + float32(pl.Scale)*0.15*(0.5-rand.Float32())
        sc := float32(pl.Scale) * (0.925 + 0.15*rand.Float32())

        if angle > 0.0 {
            if angle > 1.0 {
                angle = 1.0
            }
            res = plc.classifyRotatedRegion(row, col, sc, angle, img.Rows, img.Cols, img.Pixels,
                img.Dim, flipV)
        } else {
            res = plc.classifyRegion(row, col, sc, img.Rows, img.Cols, img.Pixels, img.Dim, flipV)
        }

        rows = append(rows, res[0])
        cols = append(cols, res[1])
        scale = append(scale, res[2]))
    }

    // Sorting the perturbations in ascendant order
    sort.Sort(plocSort(rows))
    sort.Sort(plocSort(cols))
    sort.Sort(plocSort(scale))

    // Get the median value of the sorted perturbation results
    return &puploc{
        Row: int(rows[int(round(float64(pl.Perturbs)/2))]),
        Col: int(cols[int(round(float64(pl.Perturbs)/2))]),
        Scale: scale[int(round(float64(pl.Perturbs)/2))],
    }
}
```

## Left/right face detection

Same formula for left and right eye detection. The sign is flipped on the right eye.

```
// left eye
puploc = &pigo.Puploc{
    Row:      face.Row - int(0.075*float32(face.Scale)),
    Col:      face.Col - int(0.175*float32(face.Scale)),
    Scale:    float32(face.Scale) * 0.25,
    Perturbs: perturb,
}

// right eye
puploc = &pigo.Puploc{
    Row:      face.Row - int(0.075*float32(face.Scale)),
    Col:      face.Col + int(0.185*float32(face.Scale)),
    Scale:    float32(face.Scale) * 0.25,
    Perturbs: perturb,
}
```

## Facial landmark points detection

## Facial landmark points detection



## Detection method

- ▶ The landmark points are detected based on the results returned by the pupil localization function

```
dist1 := (leftEye.Row - rightEye.Row) * (leftEye.Row - rightEye.Row)
dist2 := (leftEye.Col - rightEye.Col) * (leftEye.Col - rightEye.Col)
dist := math.Sqrt(float64(dist1 + dist2))

row := float64(leftEye.Row+rightEye.Row)/2.0 + 0.25*dist
col := float64(leftEye.Col+rightEye.Col)/2.0 + 0.15*dist
scale := 3.0 * dist

flploc = &Puploc{
    Row:      int(row),
    Col:      int(col),
    Scale:    float32(scale),
    Perturbs: perturb,
}
```

## Compute the right landmark points

This can be achieved by:

- 1.) flipping the sign of the column coordinate in tree nodes

```
if flipV {  
    c1 = min(ncols-1, max(0, (256*int(c)+int(-plc.treeCodes[root+4*idx+  
        c2 = min(ncols-1, max(0, (256*int(c)+int(-plc.treeCodes[root+4*idx+  
    } else {  
        c1 = min(ncols-1, max(0, (256*int(c)+int(plc.treeCodes[root+4*idx+1  
        c2 = min(ncols-1, max(0, (256*int(c)+int(plc.treeCodes[root+4*idx+3  
    }
```

- 2.) flipping the sign in the column coordinate for each binary test

```
if flipV {  
    dc += -plc.treePreds[lutIdx+1]  
} else {  
    dc += plc.treePreds[lutIdx+1]  
}
```

# Pigo as a shared library

## Running the face detector from Python as shared library

- ▶ Go is missing a well founded and generally available webcam library
- ▶ The idea is to transfer the Go face detection results to Python as a shared object (.so) library
- ▶ A few things we need to take care:

The exported function should be annotated with the `//export` statement.  
The source must import the pseudo C package.  
An empty main function should be declared.  
The package must be a main package.

- ▶ In Python the **Ctype** library is used to interoperate with the Go code through `cgo`
- ▶ It provides C compatible data types, and allows calling functions in DLLs or shared libraries.

## Limitations

- ▶ In Go is not possible transfer a 2D array as an array pointer.
- ▶ The trick is to convert the 2D array to a 1D array.

| -> delimit each detection group  
| -> introduce as a first slice element the number of detected faces

- ▶ Using the **numpy** library we transform it to the desired shape.
- ▶ In the end we should obtain something like below:

```
[2 0 0 0 0 272 297 213 41 1 248 258 27 41 0 248 341 27 41 0 238 599 72]
```

where the first value represents the number of detected faces and the rest are the **x, y** position and the **scale** factor

## Transfer the detection result from Go to Python

```
go func() {
    // Since in Go we cannot transfer a 2d array through an array pointer
    // we have to transform it into 1d array.
    for _, v := range result {
        det = append(det, v...)
    }
    // Include as a first slice element the number of detected faces.
    // We need to transfer this value in order to define the Python array.
    det = append([]int{len(result), 0, 0}, det...)

    // Convert the slice into an array pointer.
    s := *(*[]uint8)(unsafe.Pointer(&det))
    p := uintptr(unsafe.Pointer(&s[0]))

    // Ensure 'det' is not freed up by GC prematurely.
    runtime.KeepAlive(det)

    // return the pointer address
    pointCh <- p
}()

return <-pointCh
```

## Some caveats

- ▶ The cost using cgo can be substantial
- ▶ C and Go exists in two different universe, they interoperate through cgo.  
This is not costs effective.
- ▶ The C compiler has to be invoked for every C file in the package.
- ▶ Slower build times
- ▶ CType introduce another latency factor

# Porting Pigo to Webassembly (WASM)

## Motivation

- ▶ Running the real time version of Pigo in Python as shared object does not showed the library pure performance
- ▶ Webassembly is an emerging technology with a lot of potential
- ▶ More and more projects are getting ported to Wasm
- ▶ Go already offer good Webassembly support trough the **syscall/js** package
- ▶ Go **v1.11** was the first version targeting Wasm
- ▶ The API has gone through some refactorings and improvements to be stable starting from **v1.13**

## Considerations to keep in mind

- ▶ To access a Javascript global variable in Go you have to call the **js.Global()** function.
- ▶ Use the **Call()** function in Go in order to call a JS method.
- ▶ To get or set an attribute of a JS object or Html element we can call the **Get()** or **Set()** functions.
- ▶ *The JavaScript callback functions should always be invoked inside a goroutine , otherwise you will encounter deadlock*
- ▶ The JS methods need to be used for fetching a file. The standard **io** package is not usable here.
- ▶ In order to compile for Webassembly we need to explicitly specify and set the **GOOS=js** and **GOARCH=wasm** environment variables on the building process

```
$ GOOS=js GOARCH=wasm go build -o lib.wasm wasm.go
```

## Building the wasm file

We need to use the following build constraint

```
// +build js,wasm*
```

The **Render()** method will call the underlying JS render method:

```
// +build js,wasm
```

```
package main

import (
    "github.com/esimov/pigo/wasm/canvas"
)

func main() {
    c := canvas.NewCanvas()
    webcam, err := c.StartWebcam()
    if err != nil {
        c.Alert("Webcam not detected!")
    } else {
        webcam.Render()
    }
}
```

## The Render() method:

```
func (c *Canvas) Render() {
    var data = make([]byte, c.windowSize.width*c.windowSize.height*4)
    c.done = make(chan struct{})

    if err := det.UnpackCascades(); err == nil {
        c.renderer = js.FuncOf(func(this js.Value, args []js.Value) int {
            go func() {
                width, height := c.windowSize.width, c.windowSize.height
                c.reqID = c.window.Call("requestAnimationFrame", c.renderer)
                c.ctx.Call("drawImage", c.video, 0, 0)
                rgba := c.ctx.Call("getImageData", 0, 0, width, height)

                uint8Arr := js.Global().Get("Uint8Array").New(rgba)
                js.CopyBytesToGo(data, uint8Arr)
                pixels := c.rgbatoGrayscale(data)
                res := det.DetectFaces(pixels, height, width)
                c.drawDetection(res)
            }()
            return nil
        })
        c.window.Call("requestAnimationFrame", c.renderer)
        <-c.done
    }
}
```

## Calling the wasm file

The wasm file can be referenced in the main html file.

```
<script type="text/javascript">
    function fetchAndInstantiate(url, importObject) {
        return fetch(url).then(response =>
            response.arrayBuffer()
        ).then(bytes =>
            WebAssembly.instantiate(bytes, importObject)
        ).then(results =>
            results.instance
        );
    }
    var go = new Go();
    var mod = fetchAndInstantiate("lib.wasm", go.importObject);
    window.onload = function () {
        mod.then(function (instance) {
            go.run(instance);
        });
    };
</script>
```

## wasm\_exec.js

- ▶ Incompatibilities between Go versions
- ▶ Different Go versions have different **wasm\_exec.js** file
- ▶ Copy the `wasm_exec.js` file from **\$GOROOT** on the fly.

wasm:

```
cp -f "$$(go env GOROOT)/misc/wasm/wasm_exec.js" ./js/
GOOS=js GOARCH=wasm go build -o lib.wasm wasm.go
```

## Demo time