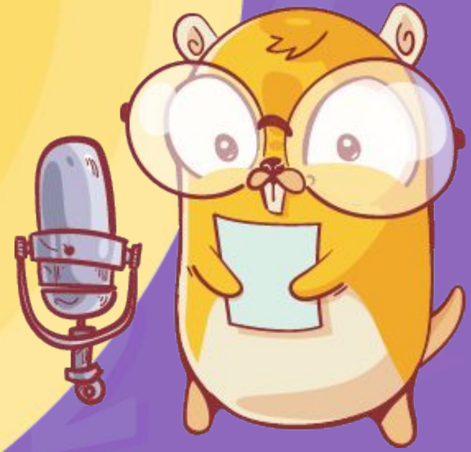
GOLAB

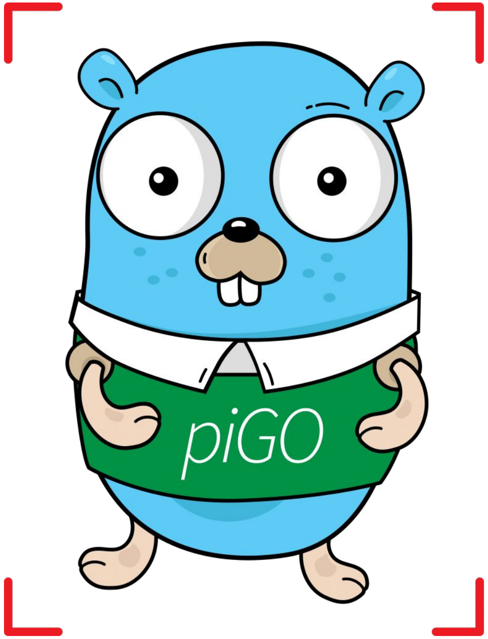
Face detection in Go and Webassembly

Endre Simo



21/10/2020

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

Why it has been developed?

- All of the existing face detection libraries developed in Go are actually bindings (wrappers) around some C/C++ libraries
- Bindings (using the cgo) most of the times are not cost effective
- Compiling a C library to Go results in slower build times

Why it has been developed?

- The desire of a single binary file is just a desire

- Installing OpenCV sometimes can be daunting

- OpenCV is huge, impossible to deploy it on small platforms where space constraints are important

What are the benefits of using Pigo?

- 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
- CLI application bundled into the library

What are the benefits of using Pigo?

- Fast detection of in-plane rotated faces
- Pupils/eyes localization
- Facial landmark points detection
- WASM (Webassembly) support

Technical overview

- **Pigo** is 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.

Unpacking the cascade files

- Because the cascades are encoded into a binary tree structure they first need to be unpacked.
- The unpacking step will result in the following struct:

```
return &Pigo{
        treeDepth,
        treeNum,
        treeCodes,
        treePred,
        treeThreshold,
 }, nil
```

Classify regions

- We classify the regions based on the parsed binary data
- The classification is based on pixel intensity comparison 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])
```

Run the cascades

- An image region is considered being face if it passes all the cascade members.
- 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 (**~0.995**)

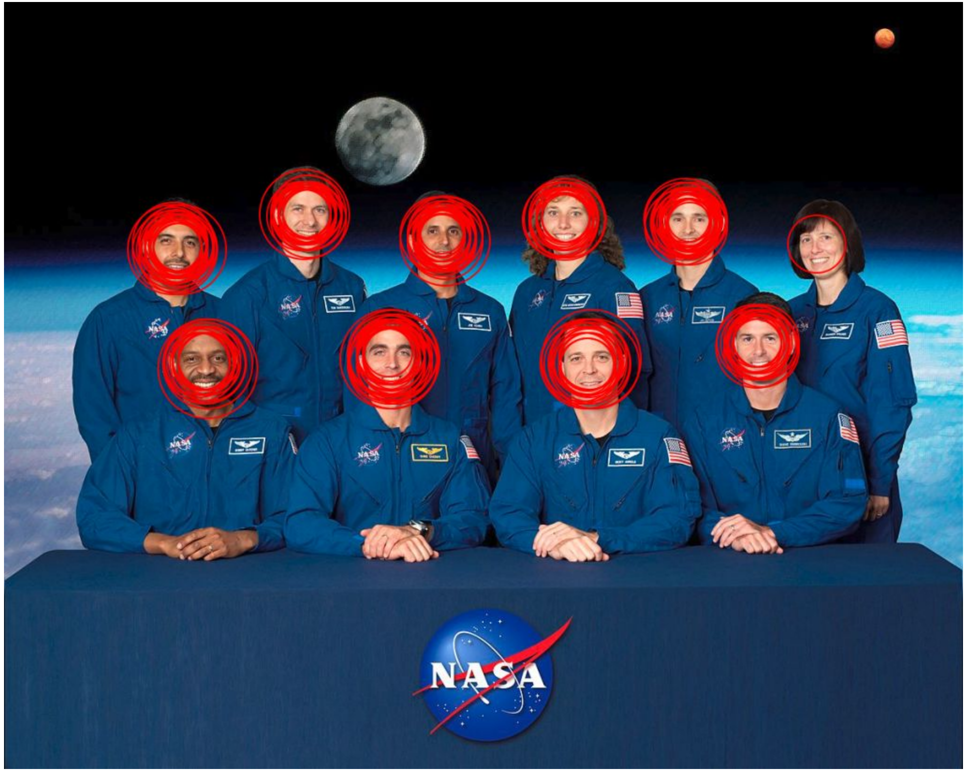
Run the cascades


```go
// 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.

Cluster detection

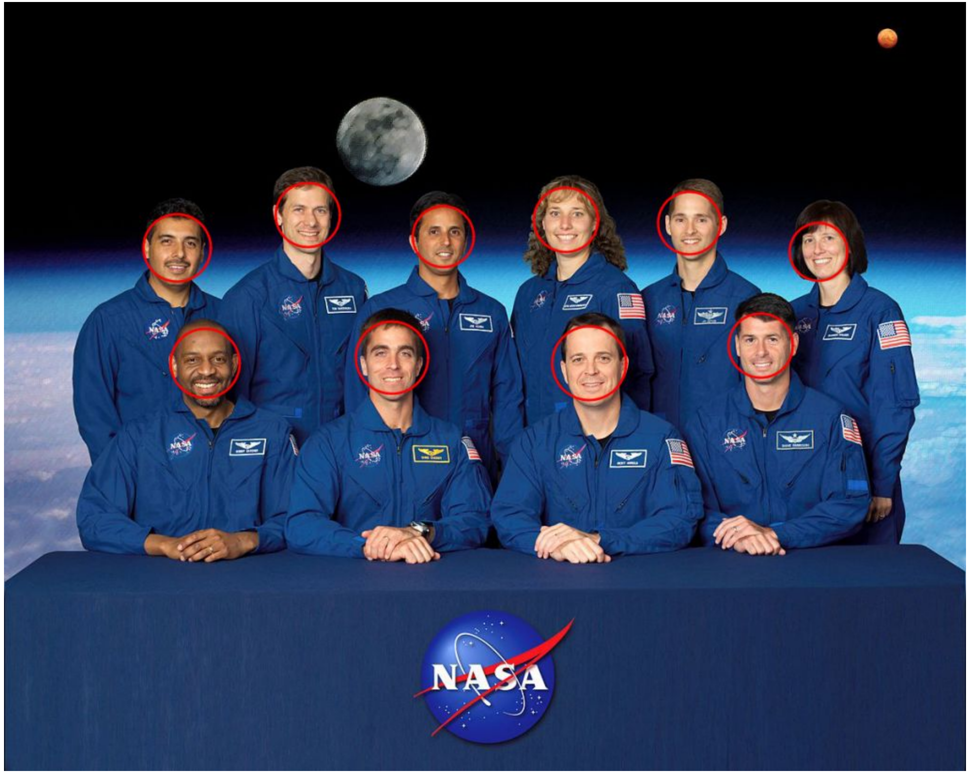


Cluster detection

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

Detection result

21/10/2020



Pupils/eyes localization

Short overview

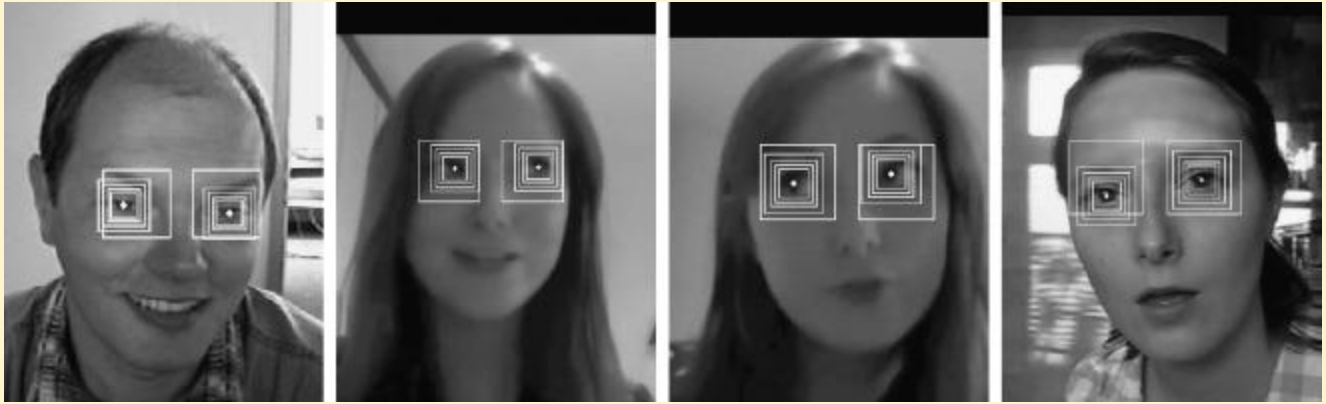
- The implementation resembles with the face detection method

- The output of the regression trees might be noisy

- Random perturbation factor to outweigh the false positive rates on detection

Pupils/eyes localization

21/10/2020



Short overview


```go
// 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

21/10/2020

Landmark points detection

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

Compute the landmark points

This can be achieved by:

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

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

GOLAB

Endre Simo

twitter.com/simo_endre
github.com/esimov
esimov.com

Powered by
develer