GOLAB

Face detection in Go and Webassembly

(using the Pigo library)

Endre Simo

21/10/2020

What is Pigo?

https://github.com/esimov/pigo

What is Pigo?

- Computer vision 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
- High processing speed
- No need for image preprocessing prior detection
- Simple and elegant API
- 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 method will return in the following struct:

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

Regions classification

- We classify the regions based on the parsed binary data
- The classification is based on pixel intensity comparison test encoded 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


```
// 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

21/10/2020

Cluster detection

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

Detection result

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

Short overview

- The detection function is almost identical for the left and 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

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

Use cases and integrations

- OpenFaaS integration
https://github.com/esimov/pigo-openfaas
https://github.com/esimov/pigo-openfaas-faceblur

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

Pigo as a shared library

- Go is missing a well founded and generally available webcam library

- Transfer the Go face detection results to Python as a shared object library (**.so**)

Interoperability between Go and Python trough cgo

- 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.

Interoperability between Go and Python trough cgo

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 the 1D array to the desired shape (normally to a multidimensional array with **x, y, dim**) .
- 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 25 1 230 587 9 25 0 233 616 9 25 0]

```
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 buffer length.
        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
```

Interoperability between Go and Python trough cgo

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)

- Running Pigo as shared object does not proved the library pure performance in terms of real time capabilities
- Webassembly is a binary instruction format targeting the web browsers which brings almost native like performance
- Many low level languages are already offer support for WASM (C, C++, Rust, Go etc.)
- More and more projects are getting ported to WASM
- Go already offer good Webassembly support through the **syscall/js** package
- The API has gone through some refactorings and improvements to be stable starting from **v1.13**

Building

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

Build constraints:

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

Porting Pigo to Webassembly (WASM)

- Running Pigo as shared object does not proved the library pure performance in terms of real time detection
- Webassembly is a binary instruction format targeting the web browsers which brings almost native like performance
- Many low level languages are already offer support for WASM (C, C++, Rust, Go etc.)
- More and more projects are getting ported to WASM
- Go already offer good Webassembly support through the **syscall/js** package
- The API has gone through some refactorings and improvements to be stable starting from **v1.13**

WASM integration

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

- 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 integration


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

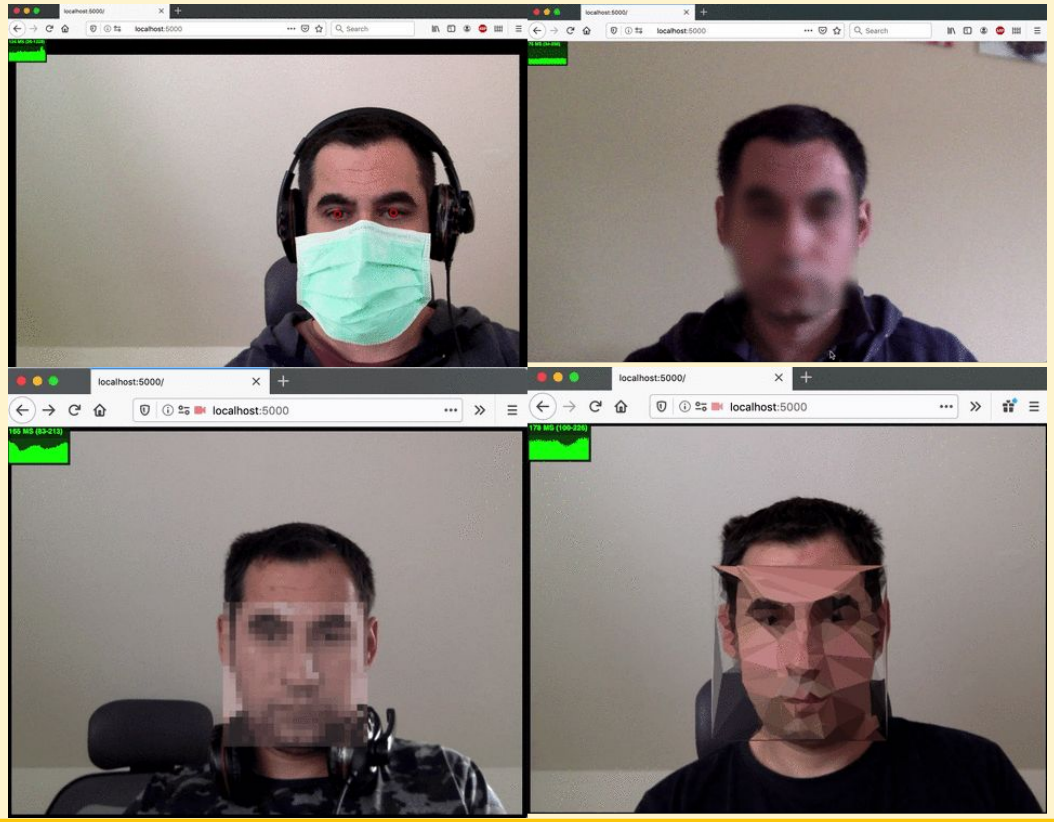
WASM integration

- 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>
```

DEMO

21/10/2020



What's next?

- Features detection and matching (to detect similarities between and inside images)
- Evolve Pigo into a fully featured Computer Vision and Machine Learning library

GOLAB

Endre Simo

@simo_endre
github.com/esimov
esimov.com

Powered by

develer