Báo cáo kết quả nghiên cứu giai đoạn 2

1. Server

- Hàm main sẽ nhận URL tại cổng 8080 và route là /facedetection sẽ thực hiện chạy hàm tương ứng, ở đây là hàm Action.

```
func main() {
    http.HandleFunc("/facedetection", Action)
    http.ListenAndServe(":8080", nil)
}
```

- Hàm Action sẽ đọc file với key là "File" và lưu vào thư mục Uploads trên server và đọc loại dữ liệu trả về bằng key "Type". Sau đó gọi làm "pigo …" để chạy xuất ra file ảnh đã đánh dấu khuôn mặt và file json chứa vị trí khuôn mặt trong bức ảnh. Sau đó sẽ dựa vào Type mà trả dữ liệu về cho client (Type 1: Image, Type 2: Json).

```
func Action(w http.ResponseWriter, r *http.Request) {
    r.ParseMultipartForm(10 * 1024 * 1024)
    _, a, _ := r.FormFile("File")
    f, _ := a.Open()
    tempFile, _ := ioutil.TempFile("Uploads", "input-*.jpg")
    defer tempFile.Close()
    fileBytes, _ := ioutil.ReadAll(f)
    tempFile.Write(file Bytes)
    cmd := exec.Command("pigo", "-in", tempFile.Name(), "-json", "-
out", "out.jpg", "-cf", "github.com/esimov/pigo/cascade/facefinder")
    _ = cmd.Run()
    fmt.Println(tempFile.Name())
    //-----------------------------
    var FileName string
    if r.FormValue("Type") == "1" {
        FileName = "out.jpg"
        fmt.Println(FileName)
        file, err := os.Open(FileName)
        if err != nil {
            log.Fatalln(err)
        }
        defer file.Close()
        io.Copy(w, file)
    } else if r.FormValue("Type") == "2" {
        plan, _ := ioutil.ReadFile("output.json")
        var data []OutputJson
        err := json.Unmarshal(plan, &data)
        if err != nil {
            fmt.Print("Cannot unmarshal the json ", err)
        }
        json.NewEncoder(w).Encode(data)
    }

}
```

❖ Dockerfile

```
#Cài đặt ubuntu và các gói cần thiết
FROM ubuntu

RUN apt-get update
RUN apt-get install -y curl
RUN rm -rf /var/lib/apt/lists/*

#Cài đặt golang
ENV GOLANG_VERSION 1.4.2
RUN curl -sSL https://dl.google.com/go/go1.14.linux-amd64.tar.gz \
        | tar -v -C /usr/local -xz
ENV PATH /usr/local/go/bin:$PATH
RUN mkdir -p /go/src /go/bin && chmod -R 777 /go
ENV GOROOT /usr/local/go
ENV GOPATH /go
ENV PATH /go/bin:$PATH

#Cài đặt Git & Pigo
RUN apt update
RUN apt-get install -y git
RUN cd go && go get -u -f github.com/esimov/pigo/cmd/pigo
RUN cd go/src/github.com/esimov/pigo/cmd/pigo && go install

#Lấy những file cần thiết
COPY Server.go /go/src/server.go
RUN mkdir /go/src/Uploads

#Chạy chương trình
WORKDIR /go/src
EXPOSE 8080
RUN go build -o ./go/src .
CMD ["./go/src"]
```

2. Client

- Hàm main sẽ chạy hàm input, sau đó gọi hàm MakeRequest để gửi dữ liệu lên server.

```
func main() {
    //--------------------------------------
    var Path, Type string
    input(&Path, &Type)
    //--------------------------------------
    MakeRequest(Path, Type)
}
```

- Hàm input với 2 thông số đầu vào là:
  - Path: Đường dẫn file ảnh.
  - Type 1: trả về ảnh đã đánh dấu vị trí khuôn mặt, Type 2: trả về file Json chứa vị trí khuôn mặt.

```
func input(Path *string, Type *string) {
    cin := bufio.NewScanner(os.Stdin)
    fmt.Print("Path(vd: C:/Users/ASUS/go/src/Final/image/in.jpg ) :")
```

```
    cin.Scan()
    *Path = cin.Text()
    fmt.Print("Type(1:Image || 2:Json ): ")
    cin.Scan()
    *Type = cin.Text()
}
```

- Hàm MackRequest mở theo Path đã nhập, và Post lên server cùng với Type, và xử lý dữ liệu nhận về
  - Type 1: Lưu lại ảnh với tên output.jpg vào thư mục Image.
  - Type 2: Xuất kết quả Json nhận được.

```
func MakeRequest(PathFile string, Type string) {
    url := "http://localhost:8080/facedetection"
    method := "POST"
    payload := &bytes.Buffer{}
    writer := multipart.NewWriter(payload)
    file, errFile1 := os.Open(PathFile)
    defer file.Close()
    part1,
        errFile1 := writer.CreateFormFile("File", "input.png")
    _, errFile1 = io.Copy(part1, file)
    if errFile1 != nil {
        fmt.Println(errFile1)
    }
    _ = writer.WriteField("Type", Type)
    err := writer.Close()
    if err != nil {
        fmt.Println(err)
    }
    client := &http.Client{}
    req, err := http.NewRequest(method, url, payload)
    if err != nil {
        fmt.Println(err)
    }
    req.Header.Add("Content-Type", "application/x-www-form-urlencoded")
    req.Header.Set("Content-Type", writer.FormDataContentType())
    res, err := client.Do(req)
    if Type == "1" {
        f, _ := os.Create("Image/output.jpg")
        _, _ = io.Copy(f, res.Body)
    } else if Type == "2" {
        body, _ := ioutil.ReadAll(res.Body)
        fmt.Println(string(body))
    }
}
```

❖ Dockerfile

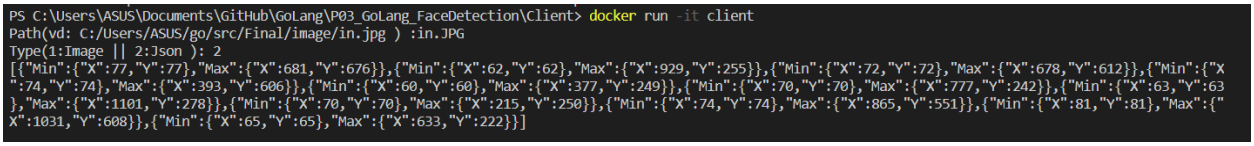
```
#Cài đặt ubuntu và các gói cần thiết
FROM ubuntu

RUN apt-get update
RUN apt-get install -y curl
RUN rm -rf /var/lib/apt/lists/*

#Cài đặt golang
ENV GOLANG_VERSION 1.4.2
```

```
RUN curl -sSL https://dl.google.com/go/go1.14.linux-amd64.tar.gz \
        | tar -v -C /usr/local -xz
ENV PATH /usr/local/go/bin:$PATH
RUN mkdir -p /go/src /go/bin && chmod -R 777 /go
ENV GOROOT /usr/local/go
ENV GOPATH /go
ENV PATH /go/bin:$PATH

#Lấy những file cần thiết
COPY Client.go /go/src/Client.go
COPY /Test /go/src
RUN mkdir /go/src/Image
RUN mkdir /go/src/Json

#Chạy chương trình
WORKDIR /go/src
EXPOSE 8080
RUN go build -o ./go/src .
CMD ["./go/src"]
```

3. Kết quả thực nghiệm

- Build dockerfile server và chạy container để khởi chạy server tại port 8080

- Build dockerfile client và chạy container để khởi chạy chương trình test.
- Nhập các thông số Path và Type, dưới đây là 1 ví dụ: Ta đã nhập Path và chọn Type 2, kết quả trả từ server về sẽ là Json các khuôn mặt được tìm thấy.



Hết