PROJECT REPORT

Mia Swery
Eva Radu

I. Introduction

In the first part of this project, our purpose is to build a program that enables a user to annotate images. Indeed, for every picture of the dataset, the user should be able to select boxes and associate a category for each of these boxes.

The aim of this program is to detect whether a human being in a picture is wearing a mask or not, in order to create an image classifier. It will be the second part of this project.

II. Image preprocessing

We selected a hundred pictures of people wearing a mask or not from a Kaggle database. Then we converted and saved them into a PNG format.
We chose to resize the pictures that would be too big to fit into the window of the program. Their new size will be a maximum of 500x500. After being resized, the images are stored in the “resized” directory of the “img” directory.
We also made sure our program is also able to deal with JPG images.
We pre-processed the images only once through our PreProcessed program.

III. The features of our interface

1) Creation and features of the boxes

If a user wants to select an area to annotate, he must first click one end of the box and release this click at another end of the box. Then a pop-up will be shown containing all the information about the box that has just been created, that is to say: the coordinates of the four ends, the height, the width, the area, and finally the category to which the box belongs.

A user can create multiple boxes as long as they are not too small, contained within another box, or containing another box.

Furthermore, whenever a box has a total intersection with the other boxes higher than 20%, this box is deleted and it will not be saved in the *annotations.json* file.

All the information on the different boxes is written within a json file entitled *annotations.json* (which is generated when all the pictures have been annotated). The category of a box is set by default to “No category”.

But a user is able to add other categories !

2) Import/Add of the categories

We decided to create a file named "*categories.json*" to save all the categories we will work with.

At first, we only have one category to annotate the pictures, which is a default category (oddly enough, it is called "No Category"). The user is then able to add new categories either from a file or manually. This file can be a json or a csv one. We considered that a json file should be written as following :

```
{
   "categories": [ "c1", "c2",”c3”]
}
```

And the csv files can be written in two different ways :

```
“c1”,
“c2”,
“c3”
```

or

```
“c1”,”c2”,”c3”
```

The categories from a file are imported thanks to the path the user will type (if the path does exist and the file is written accordingly to the rules mentioned above). These categories will be saved in our *categories.json* file.

3) Replacement of a category

A user may want to replace the name of a category with another one. It is possible thanks to our "Replace a category" button. There are two text entries. The one at the top is where the user should type the name of the category he wants to replace and the entry below it is reserved for the new name of the category.
Internally, the categories.json will be modified. For consistency's sake, all the boxes previously annotated and belonging to the replaced category will have their category's value replaced by the new name of the category.

The user might want to replace a category with another one that already exists. This action might be seen as a replacement followed by a deletion.

Don't worry : deleting a category is also possible !

4) Deletion of a category

If a user wants to delete a category, it is not a problem. This category will be safely deleted within our *categories.json* file and all the boxes belonging to this category will now be considered as “No Category” boxes.
All the user needs to do is to type the name of the category he wants to delete within the reserved entry and then press the “Delete a category” button.

5) The help button

The different steps to create a box and visualize all the related information are summarized within the help window. If a user feels lost, all he needs is to click on our “Help me !” button.

6) Next Page

Once all the boxes have been successfully created, the user can move on to the next image by clicking on the “next” button.

When a user clicks on the “next” button while the window is showing the last picture to be annotated, the *annotations.json* file where all the boxes and their information are stored will be generated !

IV. How things were conceived internally

a) Saving of the boxes

Since none of us ever used objects in python, we had to work with multiple functions and both global and local variables.

For instance, all the boxes are saved in a dictionary within the code. They have some “features” : [first_coords, second_coords, height, width, area, category, current_image_number, rect_id]. The “first_coords” attribute is a tuple of the first point selected by the user and “second_coords” is a tuple of the second point released by the user while drawing a box.
If a box is updated, any change will be saved within this dictionary.

For a later use of all the annotations made by the user, the information about the boxes is written within our *annotation.json* file.
It could be filled like this for instance :

```
{
"data": [
{
"image": {
        "path": "../img/resized/with_mask/image31.png",
        "boxes": [
        {
        "point top left": [
                170,
                262
        ],
        "point top right": [
                322,
                262
        ],
        "point bottom left": [
                170,
                141
        ],
        "point bottom right": [
                322,
                141
        ],
        "height": 121,
        "width": 152,
        "area": 18392,
        "category": "No Category",
        "path": "../img/annotated_images/image0.png"
        }
        ]
}
},
{
"image": {
        "path": "../img/resized/with_mask/image15.png",
        "boxes": "null"
}
},
…
]
}
```

b) Windows

We have a root window from which all the other windows will be created. The root window is never destroyed until the program stop running (contrary to the other windows). It is simply modified.

However, please be careful since there is a minor bug ! Our pop-up windows tend to appear behind the current window and not before it. We tried to find some way to fix that, but we didn't find any solution.

V. Our team

We tried to work together all the time so that we could keep track of the evolution of the project. Hence, it is a bit difficult to say who did what, since we did everything together.

VI. Conclusion and possible improvements

To conclude, we can say that our interface seems to be working properly and to be doing everything it is supposed to do.

Of course, some improvements could be done with more time. For instance, we could add a button that helps to retrieve the image shown before the current image. We didn't achieve to do it since we didn't know how to put back all boxes on the pictures.