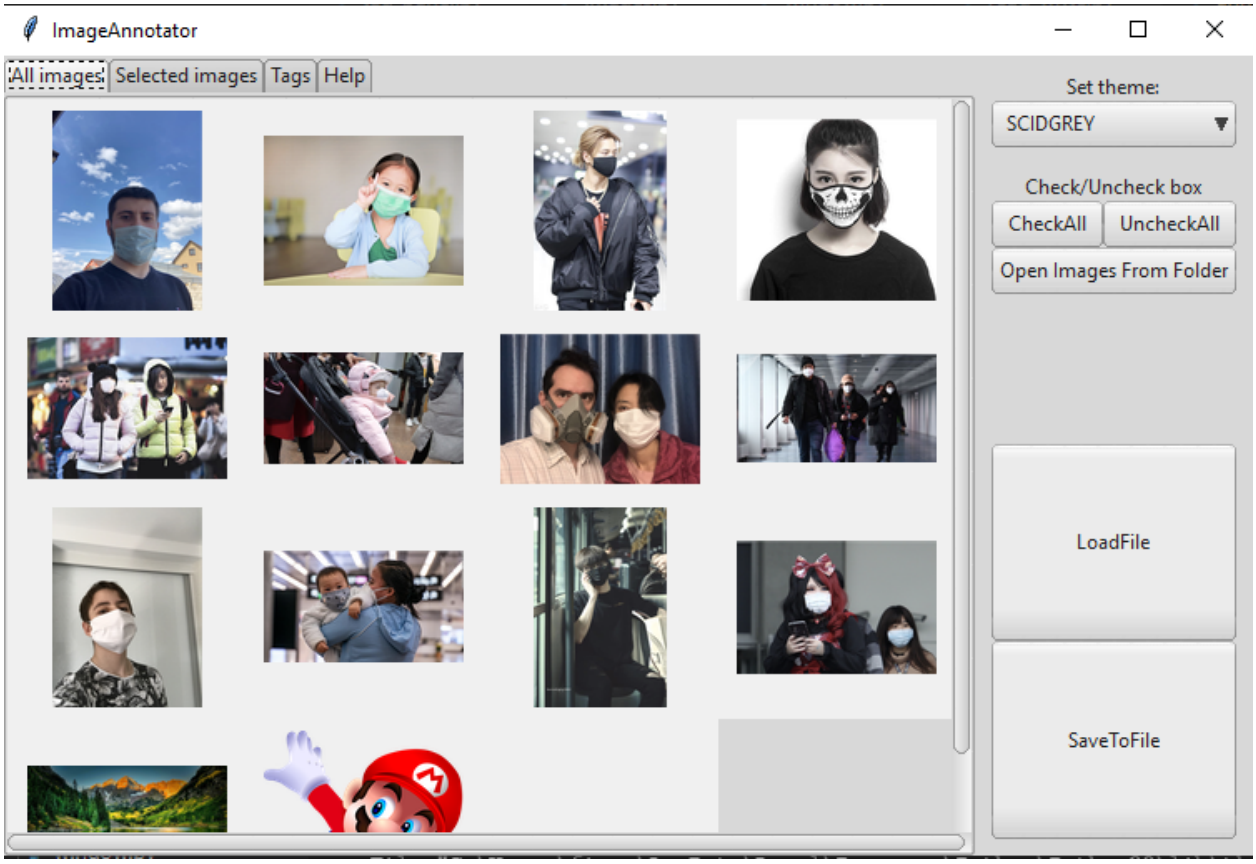


Image 1

Projet NNL - Part 1

Fissore Davide, Federica Galbiati & Antoine Venturelli

M1 Informatique, year 2021-2022

—

Main goal :

the main objective of this first part of project is to realize a graphic interface to make dynamic annotation on images loaded from a specific folder

Overview

Programming language Python version 3.9.7 with following libraries :

- PIL to load and treat images
- tkinter to charge the graphic interface
- json to save and load json encoded files
- ttkthemes to get a larger library of theme for our interface
- tkhtmlview to display simple HTML and CSS text
- shapely to work on shapes and get coverage methods

How to run the project

The project can be launched from the src folder (this is mandatory so that imports of local files go correctly) opening the *window.py* file. It will take few seconds to load all libraries, for example the ttkthemes import is a bit slow and also the interface will be less reactive, to avoid this you can pass the *"-fast"* optional parameter to open the classic tk.Tk() window (exemple *python3 ./window.py -fast,* note that the python3 command may not work on some OS for example in certain windows distributions you should use py).

Task repartition

We have worked the most part of the project in the university available room and we have collaborated together to create every aspect of the project.

In particular Antoine has worked on the graphic interface adding different options such as scrolling menu, notebook or tooltip, Davide has created the class for adding, renaming tags and images and annotation treating, Federica has dealt with the help panel using HTML and CSS and the parse and save of JSON files.

Project structure

The project is characterized by four main folders :

- src → containing the python file source
- img → containing a sample of images that can be used in the program (these images are taken from kaggle)
- pre → containing pretreatment scripts (it is still empty)
- rep → containing the report for the project

Short script description

In the src folder there are 8 scripts and a folder text in which there are some simple HTML files used for the help menu.

The interface is created via the *window.py* file and it integrates the *tag_panel.py* and the *helppanel.py* files.

The *annotator.py* file aims to create a separate window in which it is possible to open an image and add some annotation on it.

Finally, *images.py* and *tags.py* helps to store data in images whereas *read_write.py* and *scrollableframe.py* are useful files to respectively read and write JSON files and create scrollable windows within the interface.

Main features (code behind the scene)

The code is based on two classes that hold the main logic to create annotation on image and tag manipulation.

The Img class

The Img class takes in parameter the image path and from it, it creates the base attributes needed to store other information such as the *set* of tags ST and the *dict* of association DA {tag → list of coordinates associated to this tag}.

We can create a new association in the DA via the *add_tag* method taking as parameter the name of a tag (if it doesn't exist in our ST it will be added) and the top-left couple and the bottom-right coordinates of the newly-created annotation.

In this method we do some checks to verify if these coordinates are valid : a rectangle is not valid if its "*surface is less than 40 pixels in total or spatial dimensions (height and width) are too small, for example less than 5 pixels. Moreover, if a box has an intersection with other boxes for more than 20% of its surface, it should be discarded. One should also discard a box if it completely covers a pre-existing box or it is completely contained in a pre-existing one*. (indication from project subject)".

In this class we have other tools to handle annotation removal, tag rename and some methods to create a tkinter *Label* containing the desired image for the interface.

Finally the *__str__* and *__repr__* methods (similar to the toString method in java class) are used to return the string representation of an Img object to be printed in a JSON file.

The Tag class

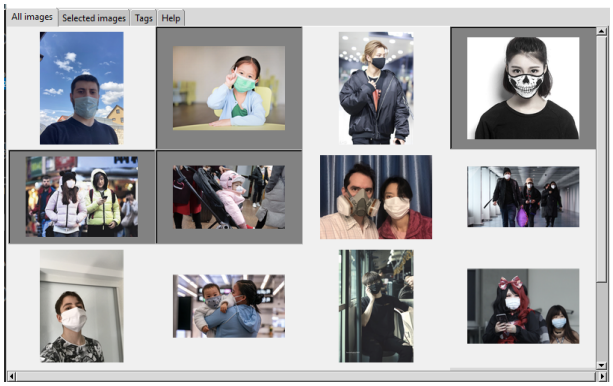
The tag class extends the set type of Python since we do not want any repetition of a pre-existing tag, moreover a tag object owns a list of *imgs* to know every image of our application and perform *side effect* operation if needed.

For example, when we rename tag A in tag B, we take every Img from the *imgs* list of our current tag object and if the image owns A then it will rename it in B. Similar operations are done for removal (every image will pop the tag key from its *dict* of annotation) and insertion.

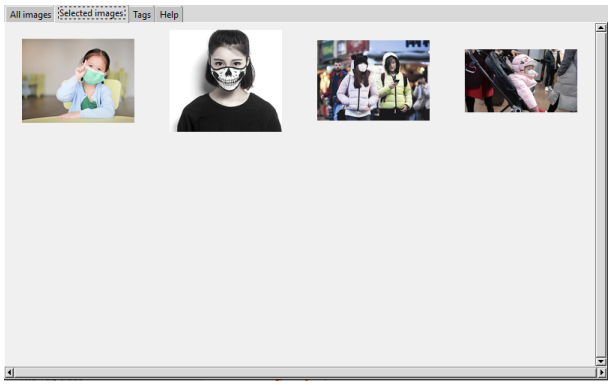
Added features (code for the interface)

"All images" & "Selected images" panels

Our interface is conceived to cover the wanted functionalities. When we run the project, we can see the first panel shown in Image 1 listing all the images of the *img* folder. Here we can select the sample of images (by clicking on it) that will be available in the *"selected images"* panel for annotation phase.



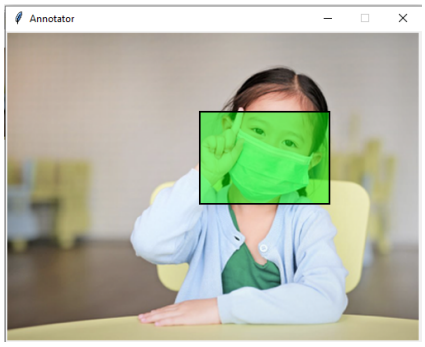
All images panel



Selected images panel

In the selected images panel you can click on the image you want to annotate and a new window will pop up.

The annotator



Annotator panel

Here, you can click a first time to draw the rectangle of the annotation and a second time to draw it (in case of error you can click the *ESC* (échap) button of the keyboard to undo the operation).

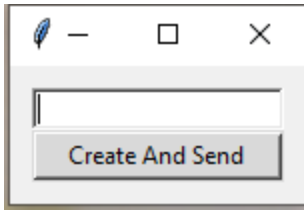


Image 2

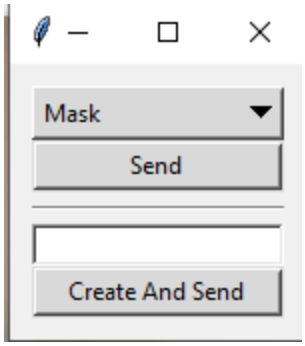


Image 3

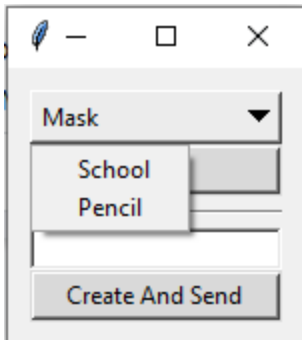


Image 4

When the user has confirmed the annotation you will see either the Image 2 if you still have no pre-existing tag where you can enter your new tag and save it in the tag set, or the Image 3 which is similar to the Image 2 with the possibility to choose one among the existing tags (Image 4) thanks to an OptionMenu.

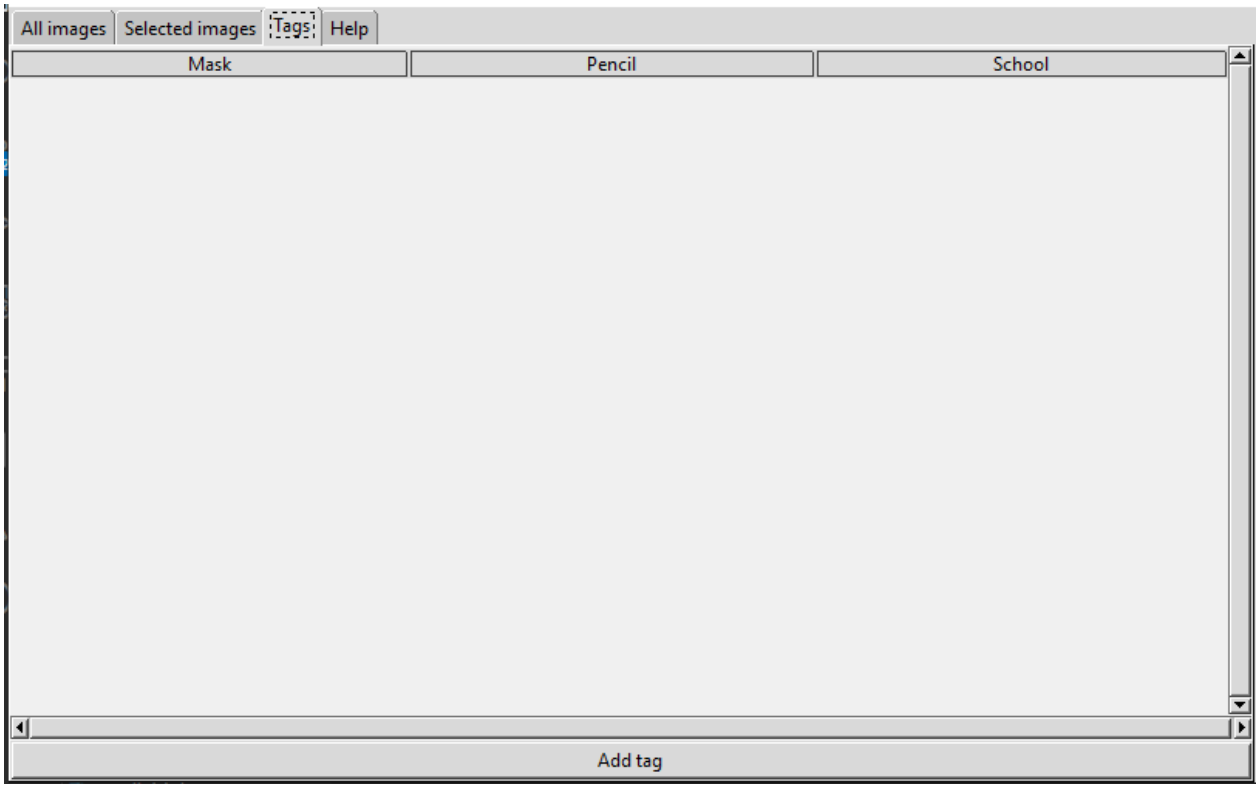
We are also able to rename or delete an annotation in case of miss-click by clicking on the wanted annotation with the right key of the mouse.

This will perform a simple rename of the tag for the current image and not a whole rename of a tag A in tag B (you can do it in Tag Panel, see continuation).

To know the current annotation name, move the mouse on it and a little tooltip will display in the bottom-left corner of the annotator window.

Finally if you create a box that does not respect the parameter imposed (area size, sides length or covering constraints) a message of error will be prompt to warn you about the impossibility of performing this task.

The tag panel



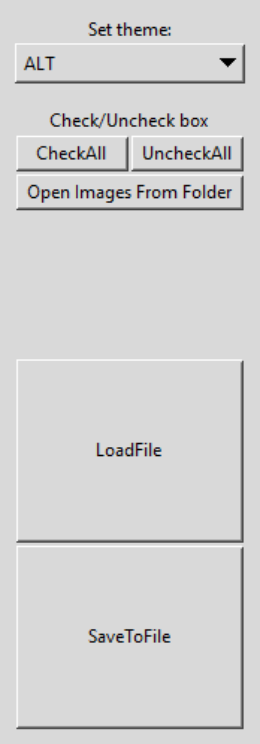
This panel is composed of a grid of existing tags and an add tag button.

You can add a tag via the corresponding button and rename and delete an existing tag by clicking with the right-key of the mouse (in this case the rename will impact every image's annotation).

The help panel

The help panel contains a brief description of how our interface works, it is composed of four clickable buttons each of them showing its related text.

Other features : the right panel



The right panel of our interface contains some buttons performing different tasks.

You can modify the theme of the window via the OptionMenu under the *"Set theme"* label (they are the default theme in case of *"-fast"* execution of your OS or some customized themes in case of the *"normal"* launch).

The other buttons :

- checkAll → allow you to select every image of the *"All images"* and pass them in the *"Selected images"* panel.
- uncheckAll → that perform the inverse operation
- Open Image From Folder → allowing you to load the images of a custom folder.
- loadFile (resp. saveToFile) → allow you to save (resp. load) a JSON file with the annotations of current (resp. previous) session.

Conclusion

WIth this first part of the project we have created a functional interface to load and annotate images. We have learnt how to use the tkinter library to correctly place tkinter objects and the shapely library to easily check area and coverage of different shapes such as rectangular boxes