View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Open Archive Toulouse Archive Ouverte

Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in: http://oatao.univ-toulouse.fr/22612

Official URL

DOI : https://doi.org/10.1145/3240508.3243656

To cite this version: Pizenberg, Matthieu and Carlier, Axel and Faure, Emmanuel and Charvillat, Vincent *Web-Based Configurable Image Annotations.* (2018) In: 26th ACM Multimedia Conference (MM 2018), 22 October 2018 - 26 October 2018 (Seoul, Korea, Republic Of).

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

Web-Based Configurable Image Annotations

Matthieu Pizenberg
University of Toulouse
matthieu@pizenberg.fr

Axel Carlier
University of Toulouse
axel.carlier@irit.fr

Emmanuel Faure
CNRS - IRIT
emmanuel.faure@irit.fr

Vincent Charvillat
University of Toulouse
vincent.charvillat@irit.fr

Figure 1: Screenshot of the interface of our image annotation Web application.

ABSTRACT

We introduce a new application for annotating images, with the purpose of constituting training datasets for machine learning algorithms. Our open-source software is meant to be easily used and deployed, configured to meet the annotation needs of any use case, and embeddable in crowdsourcing campaigns using the Amazon Mechanical Turk service.

KEYWORDS

annotation; open source software; dataset

Reference Format:
Matthieu Pizenberg, Axel Carlier, Emmanuel Faure, and Vincent Charvillat. 2018. Web-Based Configurable Image Annotations. In *2018 ACM Multimedia Conference (MM '18), October 22–26, 2018, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3240508.3243656

1 INTRODUCTION

Image annotations are required in a wide range of applications including image classification (which requires textual labels), object detection (bounding boxes), or image segmentation (pixel-wise classification). The rise and successes of deep learning lead to an increasing need for annotations, as training sets should be of a large size for these algorithms to be efficient. Yet, researchers still spend time and resources to create ad hoc tools to prepare those datasets. The application we present in this paper aims at providing a customizable tool to fulfill most image annotation needs.

Many image annotation applications already exist (Table 1). LabelMe [10], one of the most popular, provides an interface for drawing bounding boxes and polygons around objects in an image. It has been used extensively to create datasets for image segmentation. Some more recent softwares share the same goals, with their own specificities. For example, Labelbox [5] and Dataturks [3] provide annotation tasks management, particularly useful when crowdsourcing the annotations; these softwares are proprietary. The VGG Image Annotator (VIA [8]) is an open-source client application like ours, with the specificity of providing annotation attributes, editable in a spreadsheet format.

We release an open-source application [2], entirely client side, meaning that no data is uploaded to any server. Images are loaded from files and annotated locally, in the browser. The simplest tool, from a user perspective, should be immediately available i.e. should not require any additional installation to be fully functional. Our image annotation software is thus a Web-based application, easily configurable to fit users needs, as well as embeddable in the Mechanical Turk platform to design crowdsourcing campaigns.

We first present the features of our application, then describe its architecture. Finally, we explain how it can be used to start crowdsourcing experiments.

2 PRESENTATION OF THE APPLICATION

A screenshot of the application can be seen in Figure 1. The image to be annotated occupies the central part of the screen; a toolbar is located on top, object classes are available on the left and images to be annotated on the right.

Images. Multiple images can be loaded at the same time using the image icon on the top-right corner of the application. These images are not uploaded on the server, and can either be loaded locally from the client's machine, or from a distant server.

Tools. Our application includes several tools to annotate images. Icons for these tools are depicted in Figure 2. From left to right, the first available annotation is the point, that can be useful to

Application	Year	Tools	Configurable interface	Tasks management	Type	License
LabelMe	2008	bbox, polygon, iterative semi-automatic segmentation	no	Mturk integration	server	OSS
VIA	2016	bbox, polygon, point, circle, ellipse	no	no	client	OSS
Labelbox	2018	bbox, polygon, point, line	yes	yes	server	private
Dataturks	2018	bbox, polygon	no	yes	server	private
Ours	2018	bbox, polygon, point, stroke, outline	yes	Mturk integration	client	OSS

Table 1: Most relevant image annotation Web applications.

designate objects in the image. It can also be used as a seed in region-growing image segmentation methods. The second annotation we included is the bounding box, which provides the localization of objects in the image, and is used in object detection problems. The information we acquire are the left, right, top and bottom coordinates of the bounding box. The third annotation we chose to implement is the stroke, or scribble, which is a popular interaction in image segmentation. It consists in a sequence of points, interpreted as a continuous line. The outline, fourth type of annotation, is a closed shape, typically drawn around objects. It is comparable to a bounding box in essence, but provides a more precise location of objects. Finally, polygons can also be drawn (as in LabelMe, for instance), by successively clicking new points as vertices.

All these tools are available both with a mouse or a touch interaction. As a matter of fact, some tools are better suited to touch devices (for example, outlines) than others (polygons).

Figure 2: Annotation tools icons

Object classes. For most annotation tasks, we also need to differentiate objects in the images. Typically each annotated area is attributed a class, or label. The PASCAL VOC dataset [9], for example, is composed of 20 classes, grouped by categories:

- *Person*: person
- *Animal*: bird, cat, cow, dog, horse, sheep
- *Vehicle*: aeroplane, bicycle, boat, bus, car, motorbike, train
- *Indoor*: bottle, chair, dining table, potted plant, sofa, tv/monitor

In our application, classes are specified in a JSON configuration file. A strict corresponding config for PASCAL VOC classes is presented in Listing 1.

To attribute a class to an annotation, a user should first select the class in the left sidebar, then use a tool to create an annotation. Selecting a class in the left sidebar also highlights the annotations corresponding to this class.

Configuration file. The five annotation tools are optionally made available by the configuration file. In Listing 1, the last line of the depicted configuration file contains an `annotations` field, listing the tools that should be available. In this case, they all are.

In addition to the five fundamental annotation types, each type can be derived in virtually any number of variations. For example,

```
{ "classes":
  [ { "category": "Person"
    , "classes": [ "person" ]
    }
  , { "category": "Animal"
    , "classes": [ "bird", "cat", "cow", "dog", "horse",
        "sheep" ]
    }
  , { "category": "Vehicle"
    , "classes": [ "aeroplane", "bicycle", "boat", "bus",
         "car", "motorbike", "train" ]
    }
  , { "category": "Indoor"
    , "classes": [ "bottle", "chair", "dining table", "
        potted plant", "sofa", "tv/monitor" ]
    }
  ]
, "annotations": [ "point", "bbox", "stroke", "outline",
     "polygon" ]
}
```

Listing 1: A configuration file to annotate the PASCAL dataset.

```
{ "classes": []
, "annotations":
    [ "bbox"
    , { "type": "stroke", "variations": [ "fg", "bg" ] }
    ]
}
```

Listing 2: A configuration file to include two types of strokes.

interactive segmentation algorithms often require *foreground* and *background* scribbles. In our application, this would mean the user would need to draw two types of strokes. This can be achieved using the configuration file, as in Listing 2. Such configuration would result in two stroke icons in the toolbar, of different colors, just as in Figure 1.

3 TECHNICAL CHOICES

The application code is organized in two parts:

- A minimalist Node.js server, located in the `server/` directory. It is statically serving the content of `server/dist/` with compression.
- A complete Elm client application, located in the `client/` directory. Elm [6, 7] isn't a JavaScript framework, it is a functional programming language, compiling to JavaScript to run in browsers. Its syntax is inherited from Haskell but far simpler. The compiled application is 150 KB gzipped, which is great for low bandwidth connections.

3.1 The application architecture

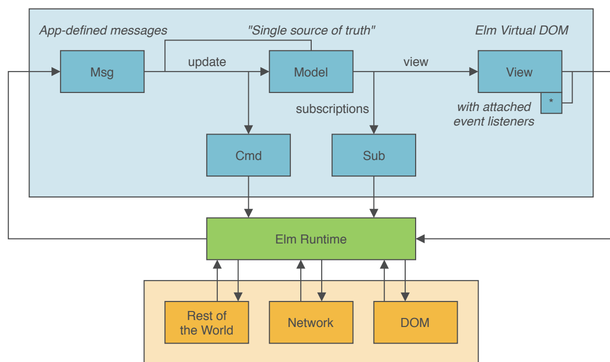

Figure 3: The application architecture.

The application architecture enforces a unidirectional data transformation flow, visualized in Figure 3. The central entity is the `Model`. It contains all and every information about our application state. The visual aspect of our application is called the `View` (basically an HTML rendered document) which is generated by the `view` function, from the `Model`. Finally, all events generate messages, of type `Msg`. The `update` function, updates the model by reacting to those messages, closing the loop.

All functions are pure, meaning there is no side effect, outputs of functions are entirely defined by inputs. There cannot be global variables mutations, real world events, network interaction etc. Basically such a program would be running in a predestined way from its start to its end, preventing us from loading images and interacting with them. This is why the application is attached to the Elm runtime, provided by the language, transforming all real world events ("side effects") into our defined set of messages, of type `Msg`.

The main challenge with pure functions is to describe side effects without performing them. Those are described in three locations:

(1) View attributes as DOM event listeners for pointer events.
(2) Commands (`Cmd`) generated by the update function, like loading of images.
(3) Subscriptions (`Sub`) to outside world events like the window resizing.

The Elm runtime takes those side effect descriptions, perform them, and, whenever there is a result / an answer, transforms it into one of our defined messages (`Msg`) and routes it to our update function.

3.2 The model states

The `state` is the main component of the `Model`. It contains the images and configuration loaded as well as the annotations performed. Its type is defined as in Listing 3 and can be modeled as a finite state machine, visualized in Figure 4.

The application available online starts in state 0 (`NothingProvided`) and enables you to reach state 2 (`AllProvided`) with buttons to load images and configuration. Two messages called `LoadImages` and `ConfigLoaded` produce transitions in the state machine.

```
1  type State
2      = NothingProvided
3      | ConfigProvided Config Classes (Zipper Tool)
4      | ImagesProvided (Zipper RawImage)
5      | AllProvided Config Classes (Zipper Tool) (Zipper
          AnnotatedImage)
```

Listing 3: State type definition.

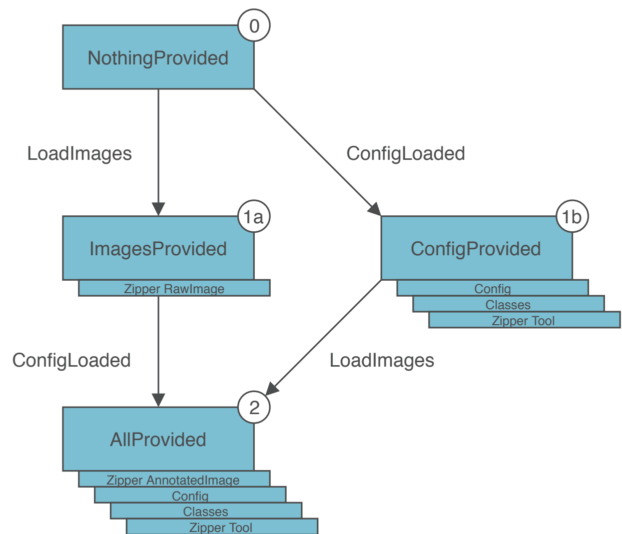

Figure 4: The application states.

```
1  type Msg
2      = WindowResizes Device.Size
3        -- pointer events
4      | PointerMsg Pointer.Msg
5        -- select things
6      | SelectImage Int
7      | SelectTool Int
8      | SelectClass Int
9        -- files
10     | LoadImages (List { name : String, file : Value })
11     | ImageLoaded { id : Int, url : String, width : Int,
          height : Int }
12     | LoadConfig Value
13     | ConfigLoaded String
14     | Export
15       -- other actions
16     | ZoomMsg ZoomMsg
17     | RemoveLatestAnnotation
```

Listing 4: Msg type definition.

3.3 The messages

All modifications of the model are understood by looking at the `Msg` type definition (Listing 4). The `update` function then performs the modifications described by those messages.

- The `WindowResizes` message is triggered when the application is resized. In the update function, it takes the new size and recomputes some view parameters.
- A `PointerMsg` message is triggered by pointer events (mouse, touch, etc.). In the update function, this is the message activating all the annotations logic code of our application.

- The messages `SelectImage`, `SelectTool` and `SelectClass` are generated when clicking on images, tools and classes.
- Files are handled by five messages:
  - When loading images from the file explorer, a `LoadImages` message is generated with a list of the images files and their names as identifiers. For each image correctly loaded an `ImageLoaded` message is generated, providing a local url, corresponding to the image in memory.
  - The messages `LoadConfig` and `ConfigLoaded` behave similarly.
  - The `Export` message causes the application to serialize into JSON all the annotations, and asks the user to save the generated file. It is triggered by clicking on the export button of the top action bar.
- Whenever an event should change the zooming level of the drawing area, a `ZoomMsg` message is generated.
- Finally, the `RemoveLatestAnnotation` message is also explicit.

3.4 The view

The view of this application is based on four components, each implemented in its own module, with potentially different versions depending on the current state of the application.

- The top action bar (`src/View/ActionBar.elm`).
- The center annotations viewer area (`src/View/AnnotationsArea.elm`).
- The right images sidebar (`src/View/DatasetSideBar.elm`).
- The left classes sidebar (`src/View/ClassesSideBar.elm`).

3.5 Library and application duality

In order to offer a turnkey solution to image annotations, we created a configurable application solving most needs. But we also thought of cases where advanced modifications are required. Consequently, the foundation of this application has been extracted in the independent package elm-image-annotation [4]. It is designed as an API to create, modify and visualize geometric shapes, useful in the context of image annotation.

Modules for manipulation and serialization (in JSON) of annotations are under the `Annotation.Geometry` namespace. It already contains one module for each tool presented earlier. If you want to introduce a new tool, this is where you can create a new module.

This package also contains the following important modules, under the `Annotation` namespace:

- `Annotation.Style`: defines types describing appearance of points, lines and fillings of annotations.
- `Annotation.Svg`: exposes functions rendering SVG elements for each annotation kind.
- `Annotation.Viewer`: manages the central visualization area, supporting zooming and translations, relative to an image frame.

If you are interested in creating another rendering target than SVG, like canvas, WebGL, ..., it would require alternative modules to `Annotation.Svg` and `Annotation.Viewer`. The rest of the code can stay unchanged.

4 CROWDSOURCING ANNOTATIONS

Image annotation interfaces are often used in the context of large datasets of images to annotate. As such, tasks management for crowdsourcing campaigns is an important feature. Labelbox and Dataturks are all-in-one services providing tasks management directly in their applications. Just like LabelMe, we choose instead to provide a configuration, ready to use with Amazon Mechanical Turk (Mturk).

Mturk comes in two sides. A "requester" is defining a set of tasks while a "worker" is performing them. Workers are payed by requesters through Mturk service. The concept of a "HIT" (Human Intelligence Task) characterizes the task unit. In our case, one HIT means one image to be annotated. We describe in details how to setup a campaign with our template in the application documentation.

5 CONCLUSION

In this paper we have introduced our web-based image annotation application. More information is available in the online documentation [1]. The application is still actively developed, we welcome all feedback and contributions.

ACKNOWLEDGMENTS

We would like to thank:

- @tforgione and @GarciaDelMolino for your wise feedbacks.
- @dncg for your Windows tests.
- The online Elm community for their help along the road: @evancz for the delightful Elm language, @ianmackenzie for your fantastic geometry library, @mdgriffith for your very refreshing layout library, @luke for the amazing tool Ellie, @norpan, @jessta, @loganmac, @antew, for your invaluable help on slack.

REFERENCES

[1] 2018. Application documentation. https://reva-n7.gitbook.io/annotation-app/. (2018). Accessed: 2018-05-20.
[2] 2018. Application source code. https://github.com/mpizenberg/annotation-app. (2018). Accessed: 2018-05-20.
[3] 2018. Dataturks. https://dataturks.com/. (2018). Accessed: 2018-05-20.
[4] 2018. Image annotation package. https://github.com/mpizenberg/elm-image-annotation. (2018). Accessed: 2018-05-20.
[5] 2018. Labelbox. https://www.labelbox.com/. (2018). Accessed: 2018-05-20.
[6] Evan Czaplicki. 2017. Elm. http://elm-lang.org/. (2017). Accessed: 2018-05-20.
[7] Evan Czaplicki and Stephen Chong. 2013. Asynchronous functional reactive programming for GUIs. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 411–422.
[8] A. Dutta, A. Gupta, and A. Zissermann. 2016. VGG Image Annotator (VIA). http://www.robots.ox.ac.uk/ vgg/software/via/. (2016). Accessed: 2018-05-20.
[9] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88, 2 (2010), 303–338.
[10] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. 2008. LabelMe: a database and web-based tool for image annotation. *International journal of computer vision* 77, 1 (2008), 157–173.