ELEKTRONICA-ICT

Bachelorproef 2023-2024

Object Detection in Automated Software Testing

Author *Brent Gerets*
Intern *Bart Stukken*
Extern *David Vandingenen*
Company *Brightest*

Abstract

This application note presents ‘BrightSight’, an application to facilitate the entire process of setting up an object detection model to test the presence of certain elements on specific web pages. The conventional method to test the presence of elements on a page in automated software testing involves parsing the page’s HTML. An element being present in the HTML is no guarantee that it is indeed visible and displayed correctly to the user, however. Because of the increased interest in machine learning over the past few years, object detection models are used to verify whether an element is visible. A comparison of popular and state-of-the-art object detection models is made to decide which to focus on, considering the large number available. Before a model can be used it must be trained on a custom dataset specifically created for the website to be tested. The presented application allows for the creation of custom datasets in a streamlined manner. These can then be used to train the models (YOLOv9 and RT-DETR) inside the application. Furthermore, the models can be tested after training to verify their performance meets expectations. The application considers the limited knowledge of machine learning most users have, by simplifying the required inputs to the bare necessities and providing a convenient user experience. The goal of creating a proof-of-concept involving object detection of web elements has been completed. However, it remains open to discussion and future research whether machine learning is the best solution to this problem. The application provides a starting point for and encourages the continued research into the use of machine learning in automated software testing. In addition, it serves as a tangible way to determine the effectiveness of object detection in the field of automated software testing.

Content

1 Introduction

In recent years, extensive research has resulted in substantial leaps in the field of machine learning. Many different machine learning models are available to be used by anyone, in a wide variety of applications. One such application that has not received as much attention as others is automated software testing. A possible reason might be that machine learning is a great solution to automate problem solving that previously could only be done efficiently by humans (e.g. detecting cats in images). Meanwhile, many solutions already exist for automated software testing. For example, Selenium is a library available in many languages that facilitates automated testing in the browser [1]. The library allows for interaction with web pages by retrieving the desired elements using XPath, id, class name, etc. [2]. In most cases this works fine. However, being able to retrieve an element using the library is not an assurance that it is visible to the user. Thus, the goal of the project is to discover whether machine learning is a viable solution to this problem. More specifically, object detection is the subcategory of machine learning that applies in this case. The research question can then be formulated as: ‘How can object detection be used in the automated software testing process?’.

In more practical terms, the goal of the project is to develop a proof-of-concept application of the use of object detection in automated testing. This includes deciding which object detection model(s) to use and developing a user interface to facilitate the training and testing of the object detection models on different web pages. It should also be possible to perform object detection inference in an automated testing environment, through code rather than the GUI. Developing an object detection model is not included in the scope since there are many pre-trained models available that can be fine-tuned for this project instead. The input of the model consists of screenshots of different pages of the website with bounding boxes around the elements that are to be detected. The model should be able to detect the elements at different positions on the screen and different scales, most other variations should not be detected since this should make the test fail (e.g. an element in the wrong colour should not be detected).

A few papers on the topic of user interface element detection have been published [3, 4, 5, 6, 7, 8]. However, all except one share a common issue that prevents their solutions from being applied to this problem. Namely, they attempt to detect as many UI elements as possible and classify them in certain categories (e.g. text, image, button, slider…). However, the aforementioned use case requires the detection of specific elements to confirm whether they are being displayed and subsequently perform actions on them. For instance, it is not the desired result to classify all buttons on a web page as buttons, but rather to detect all instances of a specific button. *Yeh et al.* [8] did propose a solution for detecting UI elements by providing screenshots of the specific elements. Their solution looks promising but unfortunately the paper was published in 2009, making the technology used significantly outdated.

First, section 2 discusses the materials and methods that were employed in the project. More specifically, the programming language and frameworks (2.1), object detection models (2.2), and data augmentation (2.3). Next, section 3 presents the results that were achieved. These are the implementation of the object detection models (3.1), the application itself (3.2), predicting without the GUI (3.3), the primary flow diagram (3.5), and deployment (3.4). Section 4 then proceeds to examine and discuss these same results. Finally, a conclusion and reflection are formulated in section 5.

2 Material and methods

This section discusses the materials and methods that were employed to achieve the results presented in section 3. It also covers the reasoning behind the decision to use these materials and methods over other alternatives.

2.1 Programming language, UI framework, environment

Often one of the first decisions to make is which programming language and/or frameworks will be used. Python (object-oriented) is used as the programming language for the application for several reasons. Firstly, it is the most popular language when it comes to machine learning and data science as a whole, thanks to its ease of use and rich collection of libraries and frameworks [9]. Moreover, Python is and has been the most popular programming language in general in recent years, according to the TIOBE index [10].

NiceGUI [11] is used to create the user interface of the application. It uses the browser as the frontend of the Python code and uses Vue and Quasar, though the application will exclusively be run locally and is not actually a website. NiceGUI also allows Tailwind classes to be used for styling. The decision for NiceGUI can be attributed to its modern look compared to native frameworks such as Tkinter [12]. It also allows for hot reloads when making changes (though they take a few seconds) and building pages is very straight-forward. NiceGUI is a new framework but is maintained and frequently updated by the open-source community.

Google Colab [13] was used to speed up training by making use of its cloud GPUs, specifically the T4 GPU with 15GB of VRAM. Around 300 'compute units' were purchased over the course of the project to increase the GPU usage limits (€11.19 per 100 units at 1.76 units per hour). These compute units also unlocked better GPUs with more VRAM (A100 with 16GB, L4 with 22.5GB) but also cost over two and a half times as much and are significantly harder to get access to without a Colab Pro+ or Enterprise subscription. The PXL was also contacted to determine whether it was possible to provide GPUs, which it unfortunately was not.

Outside of training, IntelliJ's PyCharm was used as IDE during development.

2.2 Object detection model comparison

To perform object detection, pretrained models are used and fine-tuned on a dataset containing images of the webpage elements to be tested. A large variety of pretrained models is available, so to avoid spending unnecessary time testing all of them, a selection is made based on the most important qualifiers. These were deemed to be accuracy, speed, and availability. Accuracy is the most important metric since it shows how good a model is at detecting objects. Speed is less important but faster models potentially allow for multiple checks with different models to corroborate the detection results. Availability is determined by the expected difficulty of settings up the model. It is the least important but high availability means a higher probability of the model working as intended on the new dataset.

2.2.1 Metrics and models

This section discusses which models will be compared and which metrics will be used to compare them. Firstly, accuracy determines how good a model is at performing object detection on a specific dataset. In particular, the popular Common Objects in Context dataset (COCO) [14]. Every model that was researched performed a benchmark on the COCO dataset. The COCO validation set was the most common, a few used the COCO test-dev set, and a few others didn't specify. The accuracies between these different COCO sets are generally very similar. The accuracy is expressed as the average precision (AP) with an Intersection over Union (IoU) threshold ranging from 0.5 to 0.95 with steps of 0.05 [15]. This threshold determines when a bounding box is accepted as correct. The IoU is calculated as follows [16]:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

This means that if the bounding box produced by the model and the true bounding box align perfectly, the IoU will be 1. If they don't overlap at all, the IoU will be 0. The AP is usually represented as a percentage and is sometimes also called mean average precision (mAP).

Secondly, speed refers to how much time a model spends to detect objects in an image. This is dependent on the quality of hardware that is used. Consequently, it is difficult to compare the speed of different models without performing benchmarking on all of them on the same machine. However, the number of Floating-Point Operations (FLOPs[1]) that the model must perform on an image can function as an estimate of efficiency and complexity [17]. Therefore, it can also function as an estimate of how much time

[1] Don't confuse FLOPs (Floating-Point Operations) with FLOPS (Floating-Point Operations per Second) [17].

a model needs to process an image. Some models unfortunately did not mention the number of FLOPs so the Frames Per Second (FPS) were used as an alternative.

Lastly, the availability of a model determines how easy it is to get the model up and running. The availability was split into three categories from low to high availability: ‘download’, ‘library’, and ‘library without the need for data loading’. A model that is only available through download is the hardest to get up and running since the entire setup must be figured out and performed by the developer. A model that is available directly through a library or framework is easier to use because their documentation tends to be relatively thorough. However, the data must still be loaded in and modified appropriately by the developer. Some libraries don’t require this and simply use a configuration file, these models are easiest to use and set up. The reason some degree of importance is attached to availability is that high availability ensures the model is functioning correctly and at its best performance.

There is a large collection of object detection models available. Comparing every one of them however would be a waste of time. A considerable portion of models are already outdated and easily surpassed by newer state-of-the-art models. To determine which models to compare, three different avenues where explored. First, a brief literature study of object detection model review papers was conducted to ascertain the most popular models. From this literature study can be concluded that YOLO, Faster R-CNN, and SSD are three of the most popular models [18, 19, 20, 21, 22]. Another avenue trough which to find popular models is to check which are available directly from machine learning frameworks such as PyTorch [23], TensorFlow [24] and Ultralytics [25]. These models are Faster-CNN, SSD, FCOS, RetinaNet, YOLO, RT-DETR, and more. Lastly, another way to find models is to study the COCO leaderboard [26]. Most of the models that appear in a published paper and have benchmarked on the COCO test-dev set appear on this leaderboard. Currently, Co-DETR is the best performing model in terms of accuracy. Co-DETR and some other state-of-the-art models were found on the leaderboard and compared.

2.2.2 Comparison

Taking inspiration from the work of *Ouchra and Belangour* [27], a Weighted Scoring Model (WSM) was used to determine which models should be the focus of the project. Table 1 contains all the models that were compared together with their accuracy score, speed score, availability score, and total weighted score. The scores were derived from their respective specifications. As mentioned in the previous section, the accuracy was determined using the reported accuracy of the model on the COCO dataset. The accuracy score was then calculated as a value between 0 and 100, where the model with the lowest accuracy has a score of 0, and the highest a score of 100, the rest in between. The speed score was calculated in the same way but reversed: lowest number of FLOPs gets a score of 100, highest a score of 0. If the FLOPs were not reported, the FPS was used as a fallback. As mentioned in the previous chapter, the availability was split into three categories. The lowest availability category receives a score of 0, the middle category 50, and the highest availability category gets a score of 100. Attachment 7.1 contains the complete table with all the accuracy, FLOPs and FPS metrics, as well as the sources of the data. Finally, a weight between 0 and 1 was assigned to the accuracy, speed, and availability. This weight represents the expected significance of each metric in the context of the project. Since speed is not imperative, it was assigned a weight of 0.3, while accuracy was assigned a weight of 0.6. The availability receives a very small weight of 0.1 since it is the least important factor that determines the quality of the model. These weights are by no means set in stone, but changing the weights slightly results in the same models receiving the highest score. The weighted total score is then once again a score between 0 and 100. The top 3 models are highlighted in the table below: YOLOv9-E, RT-DETR-X, and Co-DINO. The accuracy of YOLOv9-E and RT-DETR-X is around 10% lower than Co-DINO, which is significant. However, the combination of superior speed and availability places these models right behind Co-DINO. Even if the weights are changed slightly (while still reflecting their importance in the project), YOLOv9, RT-DETR and Co-DINO always end up in the top 5. In the case of YOLOv9 and RT-DETR a variant other than the highlighted one could be used since the scores are very similar and the slight increase in accuracy might not be worth the decreased speed. Usually, variants of a model perform similarly. However, Co-DETR is a special case since it is more of a technique that can be applied to existing DETR models. Therefore, the two Co-DETR variants in the table perform very differently due to Co-DINO and Co-Deformable-DETR being completely different DETR models. From this weighted scoring model comparison can be concluded that YOLOv9, RT-DETR, and Co-DINO should be the focus of the project. Unfortunately, a lot of problems were encountered with Co-DINO and it ended up not being implemented in the application (see section 3.1.2).

Table 1: Weighted Score Model comparison of popular and/or state-of-the-art object detection models (full version: 7.1).

Model name	Model variant	Accuracy score	Speed score	Availability score	Weighted total score
YOLOX	s	42.30	98.98	0.00	55.07
	m	57.95	96.33	0.00	63.67
	l	64.79	91.72	0.00	66.39
	x	68.22	84.59	0.00	66.31
YOLOv7		68.46	94.59	0.00	69.45
	X	72.62	89.78	0.00	70.50
	W6	76.77	80.19	0.00	70.12
	E6	79.95	71.44	0.00	69.40
	D6	80.93	54.99	0.00	65.06
	E6E	82.15	52.94	0.00	65.17
YOLOv8	n	34.47	100.00	100.00	60.68
	s	53.06	98.88	100.00	71.50
	m	66.01	96.06	100.00	78.43
	l	72.62	91.17	100.00	80.92
	x	75.06	85.95	100.00	80.82
YOLOv9	S	57.70	98.98	100.00	74.32
	M	68.95	96.16	100.00	80.22
	C	72.86	94.69	100.00	82.12
	E	**79.22**	**89.64**	**100.00**	**84.42**
RT-DETR	L	72.86	94.29	100.00	82.00
	X	**77.26**	**87.29**	**100.00**	**82.55**
Co-DETR	**Co-DINO**	**100.00**	**84.76**	**0.00**	**85.43**
	Co-Deformable-DETR	86.31	51.99	0.00	67.38
Faster R-CNN	R50-FPN	33.74	75.28	50.00	47.83
	R50-FPN	41.56	13.17	50.00	33.89
	R101-FPN	45.97	8.85	50.00	35.23
	X101-FPN	48.41	2.79	50.00	34.88
R-FCN		20.29	0.00	0.00	12.18
SSD	300 VGG16	0.00	34.24	50.00	15.27
FCOS	R50-FPN	40.59	89.80	50.00	56.29
RetinaNet	R50-FPN	32.27	70.77	50.00	45.60
	R50	37.90	11.93	50.00	31.32

	R101	42.05	11.93	50.00	33.81
InternImage	T	63.33	85.26	0.00	63.57
	S	64.79	81.32	0.00	63.27
	B	66.26	72.24	0.00	61.43
	L	80.44	21.60	0.00	54.74
	XL	80.68	0.00	0.00	48.41
Weight		**0.60**	**0.30**	**0.10**	

2.3 Data augmentation

Data augmentation is the practice of expanding a dataset by performing certain transformations on the elements of the dataset. This allows the model to learn invariant features and prevents overfitting [28]. It also increases the size of the dataset, which is important in this project since the dataset only consists of a few screenshots of the website. There are numerous augmentations that can be performed such as resizing, rotating, translating, changing colour, flipping, adding noise, etc. [29]. However, a lot of the augmentations are not desired in this project. For example, changing the colour of the images in the dataset is not desired since the model should only detect the elements in the correct colour. The same goes for flipping, rotating, and shearing. Nonetheless, there are a few data augmentations that can be applied in the context of this project. Firstly, the images can be resized to allow the model to detect the elements at different sizes. This is desired since changing the size of the browser window could change the dimensions of the elements within. It is important to resize only the part of the image within the bounding box. Resizing the entire image would be pointless since the image must be resized again before entering the model. Secondly, the part of the image withing the bounding box can be translated to a different position inside the image to prevent the model from overfitting to a specific position. Lastly, even though the model shouldn't be invariant to colour, noise can still be added to prevent overfitting and to allow the model to still detect the element when there are other elements in the vicinity. These are the most important data augmentations that can be applied. It should be noted that these augmentations should be applied to each other as well. For example, the original images are randomly resized. Then all the images, including the resized ones, are randomly translated. Finally, noise is added to all the images. Depending on the number of times an augmentation is applied, this can exponentially increase the size of the dataset.

3 Results

3.1 Implementation of models

3.1.1 YOLOv9 and RT-DETR

Both YOLOv9 and RT-DETR make use of the Ultralytics library. Training and predicting is very straightforward using this library. It simply requires a YAML file containing the paths to the train, test, and validation sets, and the labels. The train, test and validation folders contain the images as well as the annotations (text file containing all the bounding box data of an image). The annotations must be in the YOLO format which is one box per line consisting of a label (index) followed by the coordinates of the centre of the box, and the width and height of the box (all normalized). This is all done automatically by the dataset creator in the application.

3.1.2 Co-DINO

After learning how to use the MMDetection toolbox that was used to create the Co-DETR models and installing the correct Python version and required packages, Co-DINO was successfully tested on the COCO validation set. A new model configuration file was created that inherits the Co-DINO configuration [30], which in turn inherits the 5-scale ResNet-50 configuration [31]. The new configuration loads the checkpoint of the *5-scale Swin-L 16 epochs* Co-DINO model (provided through Google Drive [32]) that was also pre-trained on the Objects365 dataset and has an mAP of 64.1 on COCO-val [33]. The configuration was set up to use the COCO dataset standard, but with different classes. The parent configurations also had to be slightly modified to allow for a different number of classes. This configuration was functional but required too much VRAM and was therefore failing before the first epoch could start. To prevent this issue, the backbone was frozen (no new weights calculated) by increasing the number of frozen stages from 1 to 4 in the original Co-DINO config file. The maximum image size was also decreased to a lower resolution. These solutions combined reduced the peak VRAM usage to 12GB, within the limit of the Colab T4 GPU.

The model was trained on several datasets (of the same website) created by using different parameters as input for the data augmentation script that was developed beforehand. With every dataset, the model would get to 90+% accuracy suspiciously quickly, often already right after the first epoch. After a few epochs the accuracy was 99+%. The greatest number of epochs that has been trained on a single dataset is 10 epochs. The last epoch had an accuracy of 99.82%. Testing the model on new screenshots after training confirmed that something was wrong, since nothing was being detected apart from an incorrect label with less than 1% confidence every few screenshots. Work on Co-DINO was halted due to not much time being left until the end of the semester, and the much longer training time compared to other models (1 epoch per hour vs. 1 epoch every few minutes).

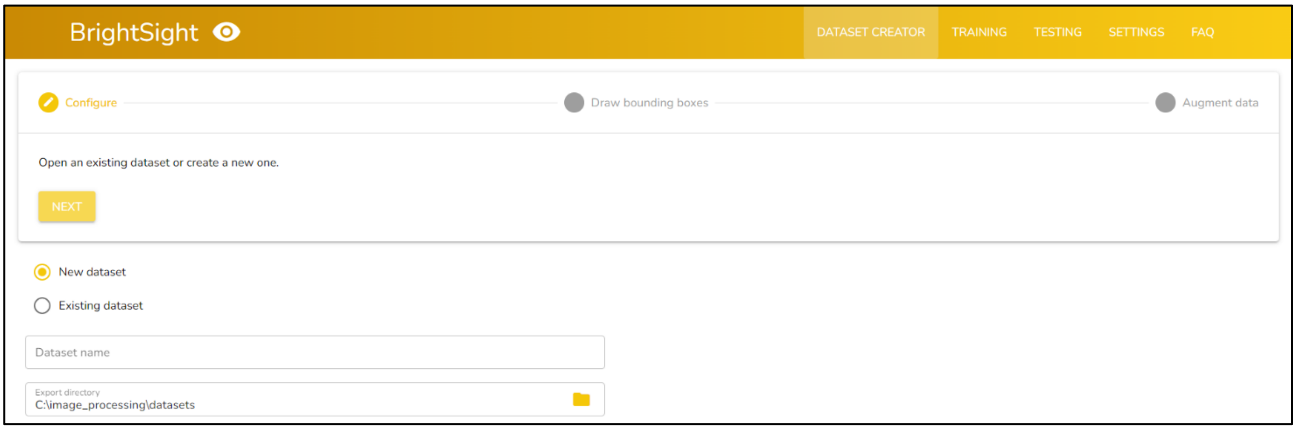
3.2 Application

An application dubbed 'BrightSight' was created to facilitate the training of the models on the websites of choice. The application also hosts a dataset creator which is used to create a dataset that can be used to train a model on a specific website. In addition, the prediction capabilities of the trained models can be manually tested using the application. Also, a settings page allows users to change default paths. Finally, there is a frequently asked questions page with usage information. The following sections discuss the individual pages of the app in further detail.

3.2.1 Dataset creator

Training a model requires a dataset as input. Creating a dataset is not straightforward, especially when the user is unfamiliar with the model. To make this process simpler, a dataset creator was developed. Creating a dataset using the dataset creator requires three steps: configuration, drawing bounding boxes, and augmentation. Each of these steps is represented by a separate page which can be navigated between using a 'stepper'.

The configuration step contains a radio menu with two options: 'new dataset' (Figure 1) and 'existing dataset' (Figure 2). The former prompts the user for a dataset name and save location. The latter requires the user to select a dataset save file (generated by the dataset creator). All file prompts in the application make use of a modified version of the local file picker example found in the NiceGUI Github repository [34] (Figure 3). A custom file picker is necessary due to the limitations of the native file picker. Mainly the fact that it does not allow the selection of a folder. Once a name and save location has been picked, or a save file has been successfully imported, the button to advance to the next step becomes available.



Open an existing dataset or create a new one.

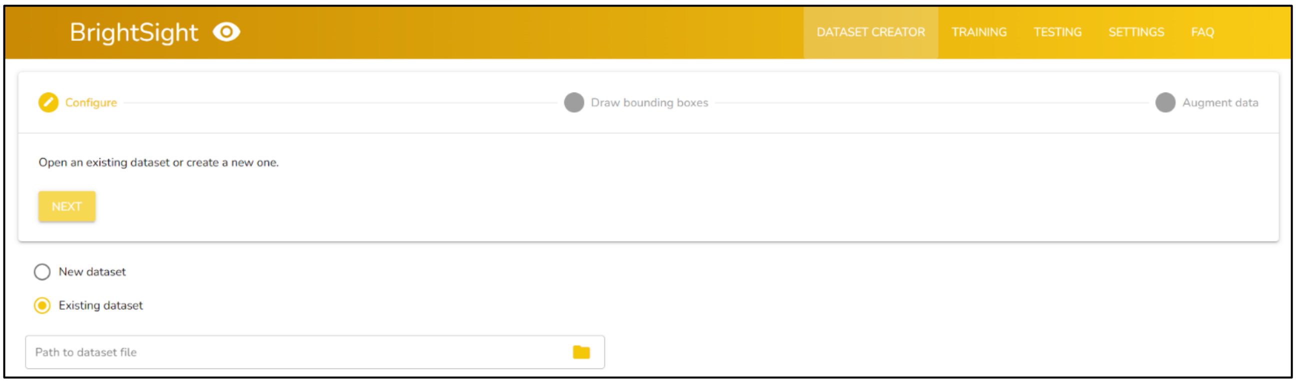
● New dataset

○ Existing dataset

Dataset name

Export directory
C:\image_processing\datasets

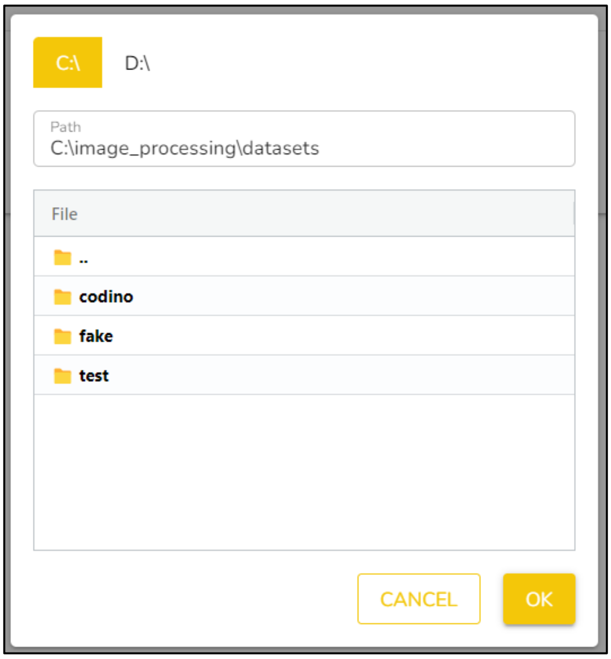
Figure 1: Dataset creator configuration step, new dataset.



Open an existing dataset or create a new one.

○ New dataset
● Existing dataset

Figure 2: Dataset creator configuration step, existing dataset.

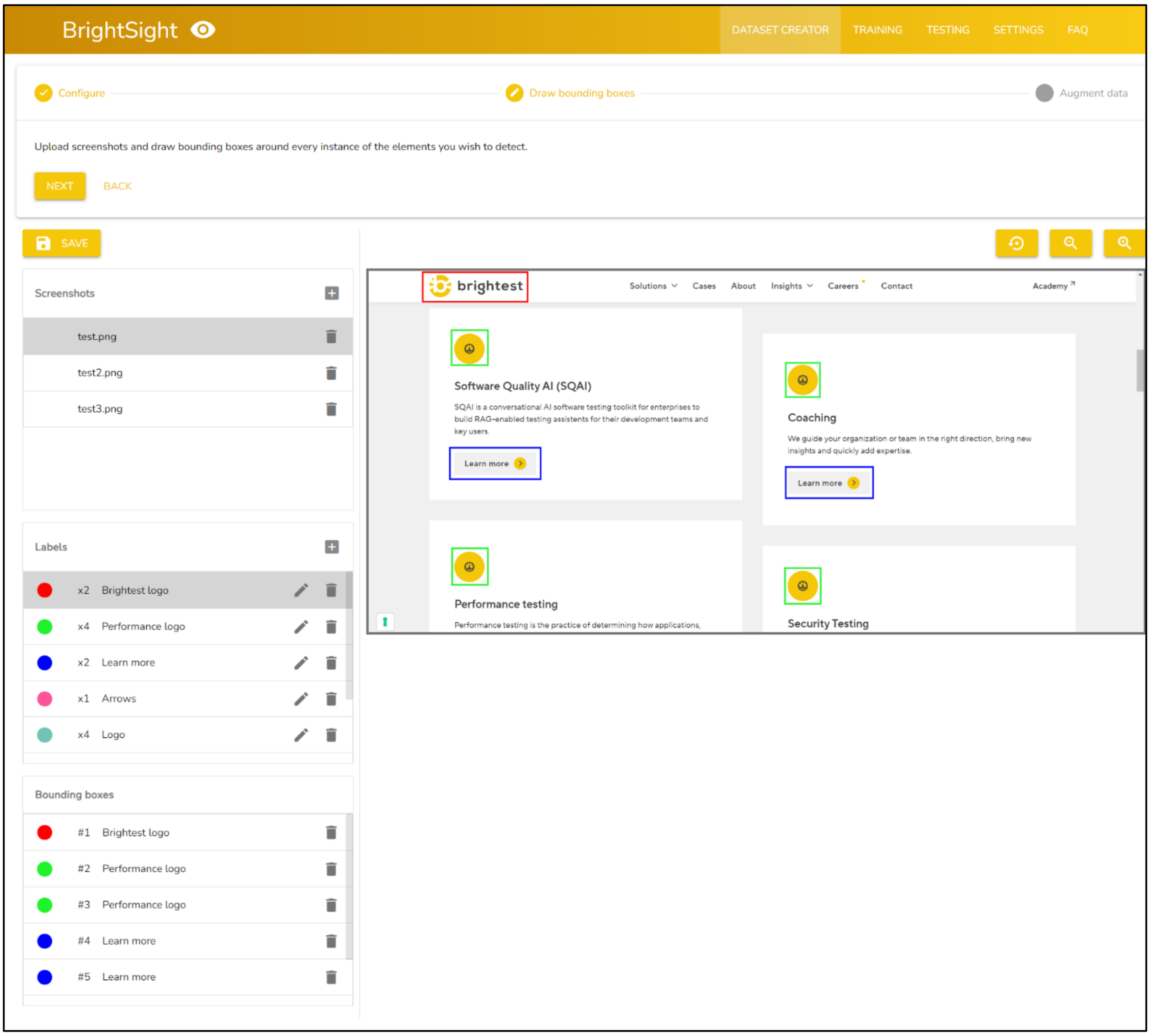


Path
C:\image_processing\datasets

File
..
codino
fake
test

Figure 3: Custom local file picker.

In the second step (Figure 4), screenshots of the website must be uploaded. On each screenshot, bounding boxes can be drawn around every instance of each element that must be detected by clicking and dragging on the screenshot. These bounding boxes are also given a label to represent which element they encompass. When drawing a new bounding box, the selected label is assigned to it. Only one screenshot is displayed at a time on the canvas, which also allows for panning and zooming using respectively the right mouse button and the scroll wheel. The canvas can be returned to its original position using the button above it on the right. It is possible to switch between screenshots, edit labels, edit bounding boxes, and remove screenshots, labels, and bounding boxes. Labels can be edited by clicking the edit icon, the name and colour of the label can then be modified. The changes take effect immediately. Bounding boxes can be edited by clicking on them inside the canvas or selecting them in the list. Once selected, a bounding box can be moved, resized, and deleted. Finally, every time a destructive action is about to be taken (such as removing a label, and thus also all the bounding boxes with that label) a dialog is displayed to confirm the action (Figure 5). The vertical divider between the lists and the canvas can be moved to increase the size of either. When the user is satisfied with their work, they can continue to the final step.



Upload screenshots and draw bounding boxes around every instance of the elements you wish to detect.

Screenshots

test.png

test2.png

test3.png

Labels

🟥	x2	Brightest logo
🟩	x4	Performance logo
🟦	x2	Learn more
●	x1	Arrows
●	x4	Logo

Bounding boxes

🟥	#1	Brightest logo
🟩	#2	Performance logo
🟩	#3	Performance logo
🟦	#4	Learn more
🟦	#5	Learn more

Software Quality AI (SQAI)

SQAI is a conversational AI software testing toolkit for enterprises to build RAG-enabled testing assistents for their development teams and key users.

Coaching

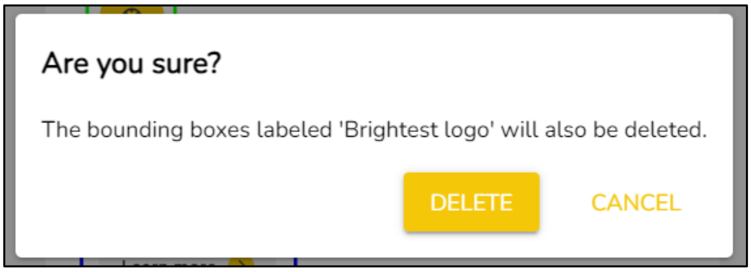
We guide your organization or team in the right direction, bring new insights and quickly add expertise.

Performance testing

Performance testing is the practice of determining how applications,

Security Testing

Figure 4: Dataset creator, drawing bounding boxes step.

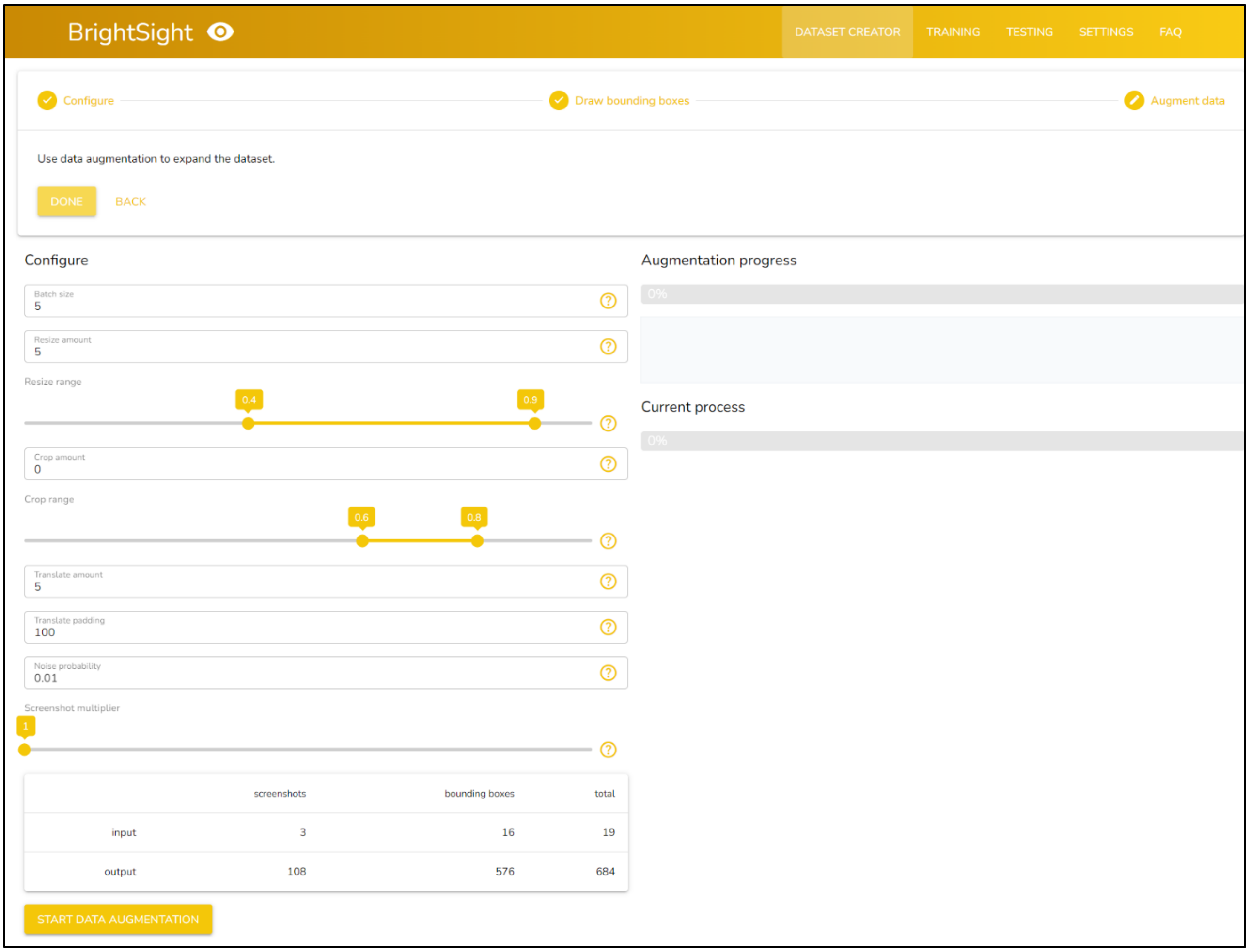


Are you sure?

The bounding boxes labeled 'Brightest logo' will also be deleted.

Figure 5: Confirmation dialog.

The data augmentation step (Figure 6) extracts the bounding boxes from the screenshots which, together with the screenshots, undergo a series of augmentations to increase the size of the dataset and the variety within it. The augmentation page allows the user to select how many times each of the three augmentations (resizing, cropping, translating) will be applied to each image, and what the range of these operations will be. For example, a resize amount of 5 and resize range of 0.4 to 0.9 means every image is resized 5 times, with a random factor between 0.4 and 0.9. Pixels also have a random chance to be turned into noise, which can be configured. Lastly, there is a screenshot multiplier which multiplies the number of times each of the three augmentations is executed, but only for the full screenshots. The goal of this is to increase the ratio of screenshots to bounding box images. A table containing the amount of input images and output images is displayed and dynamically updated. Once the configuration is completed, the data augmentation can be started. This step can take a while depending on the number of screenshots and bounding boxes, as well as the configured amounts for each of the augmentations. Two progress bars are displayed, one for the overall progress of the data augmentation, and one for the current step that the algorithm is performing. There is also a textbox which provides more feedback on the progress. After the data augmentation is completed, the provided export path will contain the dataset images and the required dataset files in different formats for training and testing.



Use data augmentation to expand the dataset.

Configure

Batch size: 5

Resize amount: 5

Resize range: 0.4 – 0.9

Crop amount: 0

Crop range: 0.6 – 0.8

Translate amount: 5

Translate padding: 100

Noise probability: 0.01

Screenshot multiplier: 1

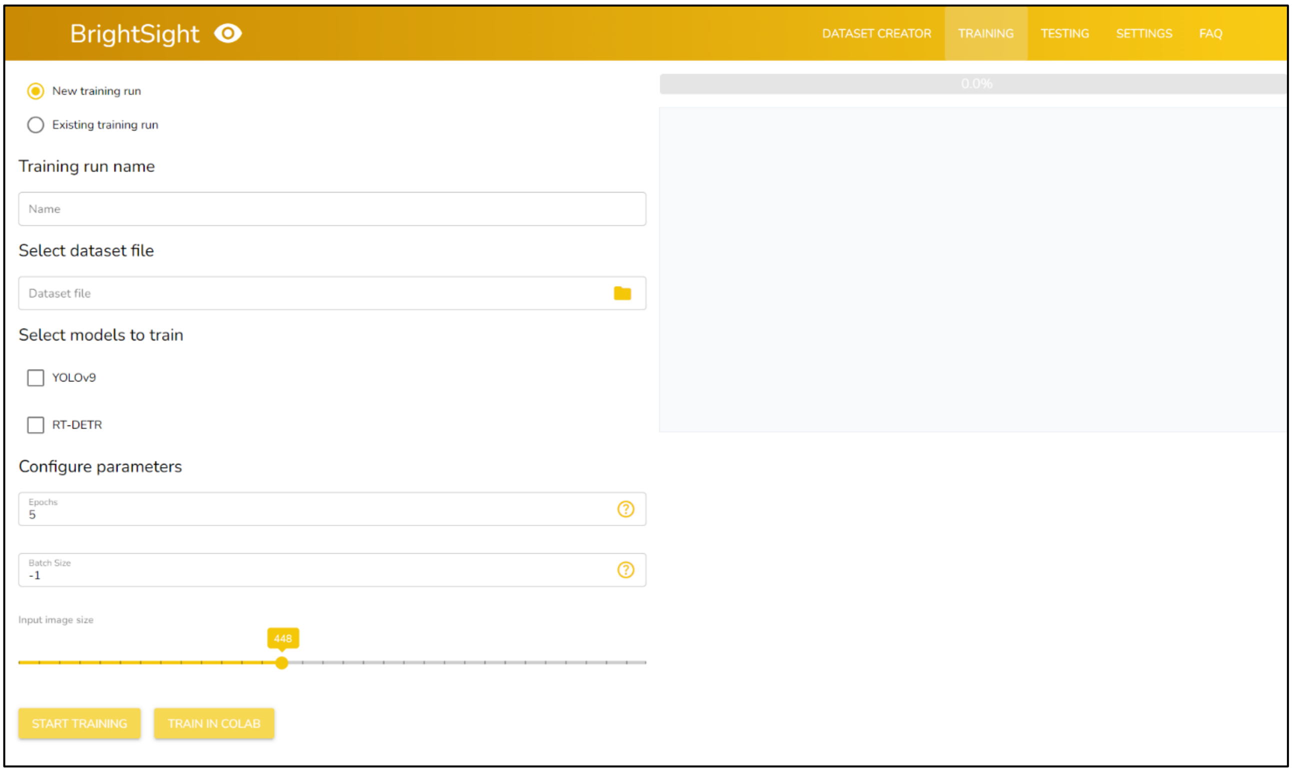
	screenshots	bounding boxes	total
input	3	16	19
output	108	576	684

Figure 6: Dataset creator, data augmentation step.

The dataset creator automatically saves the current dataset when switching between steps, and a manual save button is available in the second step (Figure 4). To stop the user from accidentally leaving without saving during the second step, a few lines of JavaScript code are used to show the browser's built-in confirmation notification when leaving the page. Some inputs inside the application have a question mark icon displayed on the right side. When hovering over this icon, a tooltip is displayed with information about the value that the input expects. These hints can be turned off in the settings. The dataset creator can also be used on its own without using the rest of the application, since it outputs config files that are compatible with Ultralytics as well as COCO-style annotation files.

3.2.2 Training

It is possible to either start a new training run (Figure 7) or continue an existing training run (Figure 8). Making it straight-forward to train a few epochs and then continue with more epochs later. Before starting a new training run, a name must be given to the training run and models to train must be selected (either YOLOv9 and/or RT-DETR). Then a dataset file (the one generated by the dataset creator) must be selected and the batch size, number of epochs, and input image size must be configured. The model automatically resizes the dataset images to this size, so it is not necessary for all dataset images to be this size. Increasing the input image size can increase accuracy but it also increases training time. During training the progress bar shows the training progress, and the console below shows some more detailed training output. The trained model is saved in the directory that can be configured in the settings. When continuing a training run, it is possible to change all the parameters except for the name of the training run (after clicking the 'import' button).



● New training run

○ Existing training run

Training run name

Name

Select dataset file

Dataset file

Select models to train

□ YOLOv9

□ RT-DETR

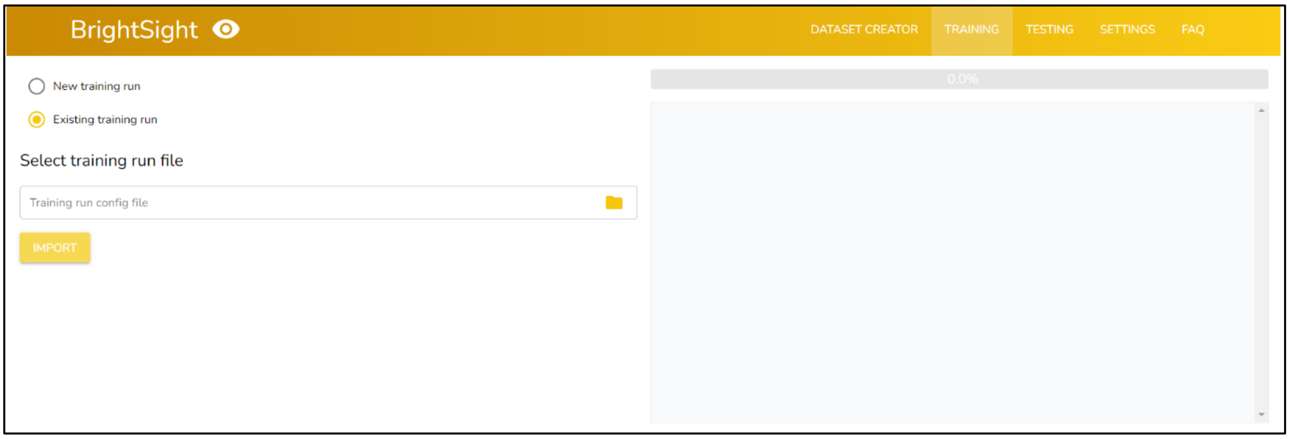
Configure parameters

Epochs
5

Batch Size
-1

Input image size
448

Figure 7: Training page, new training run.



○ New training run

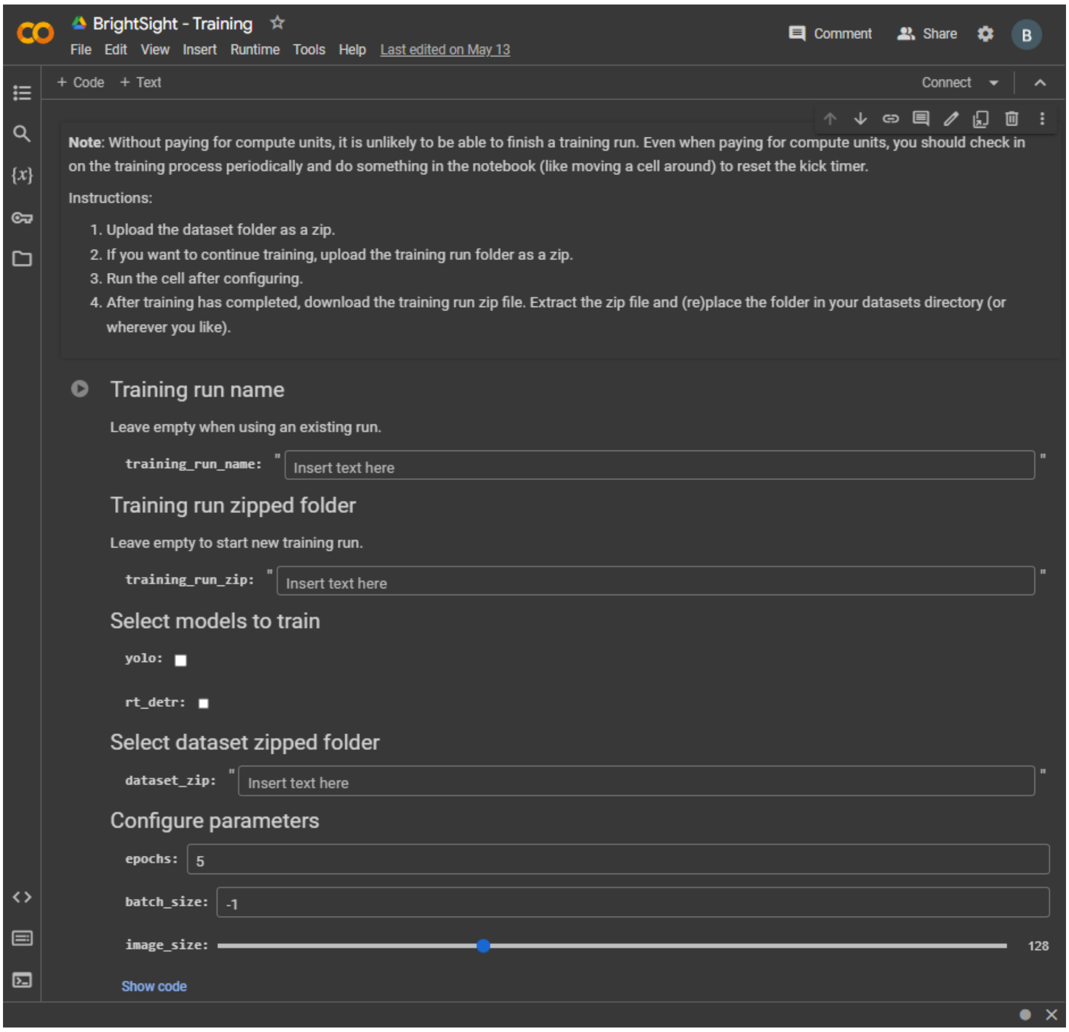
● Existing training run

Select training run file

Training run config file

Figure 8: Training page, existing training run.

Both options also have a 'train in Colab' button. If this button is clicked a file picker appears and after selecting an output directory, a zip file of the training run and a zip file of its dataset are made. Both zip files are placed in the specified output directory together with a Python notebook that can be opened in Google Colab. The two zip files must then be uploaded to the Colab runtime environment. The notebook (Figure 9) contains a recreation of the interface in the application, as well as instructions. After performing the configuration, all the user must do is run the only cell in the notebook to start training.



Note: Without paying for compute units, it is unlikely to be able to finish a training run. Even when paying for compute units, you should check in on the training process periodically and do something in the notebook (like moving a cell around) to reset the kick timer.

Instructions:

1. Upload the dataset folder as a zip.
2. If you want to continue training, upload the training run folder as a zip.
3. Run the cell after configuring.
4. After training has completed, download the training run zip file. Extract the zip file and (re)place the folder in your datasets directory (or wherever you like).

Training run name

Leave empty when using an existing run.

training_run_name: " Insert text here "

Training run zipped folder

Leave empty to start new training run.

training_run_zip: " Insert text here "

Select models to train

yolo: □

rt_detr: □

Select dataset zipped folder

dataset_zip: " Insert text here "

Configure parameters

epochs: 5

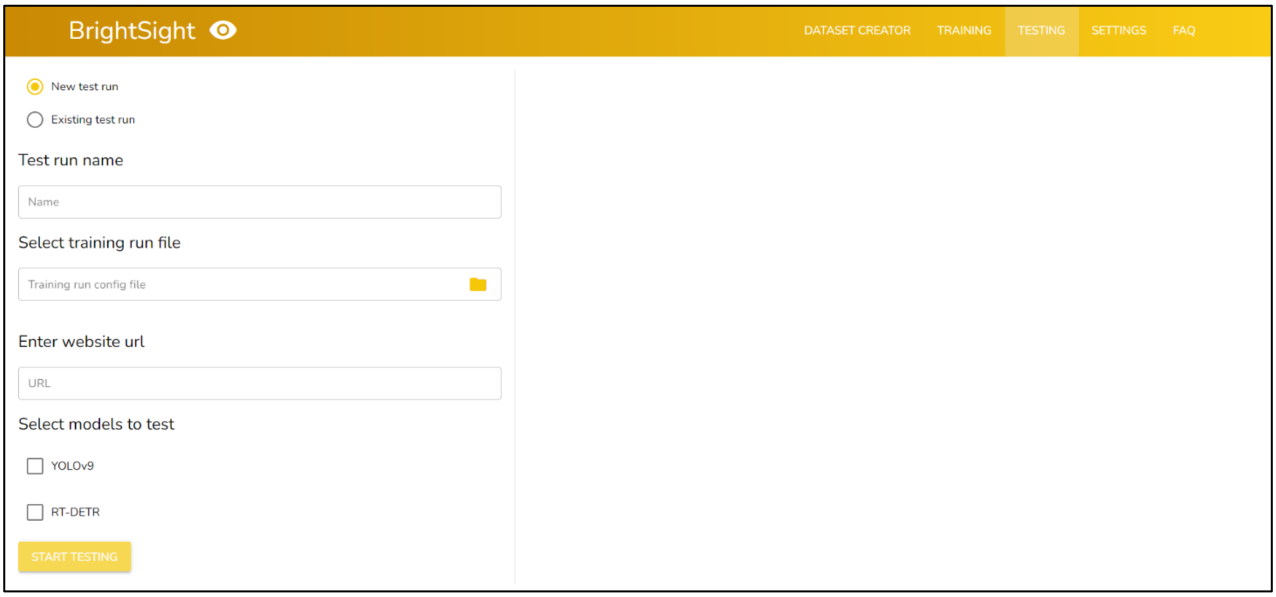
batch_size: -1

image_size: 128

Figure 9: Notebook for training in Google Colab.

3.2.3 Testing

The testing page requires the user to either create a new test run (Figure 10) or open an existing saved test run (Figure 11). A test run consists of a name, a training run that contains trained models, and a URL to the website to be tested.



● New test run

○ Existing test run

Test run name

Name

Select training run file

Training run config file

Enter website url

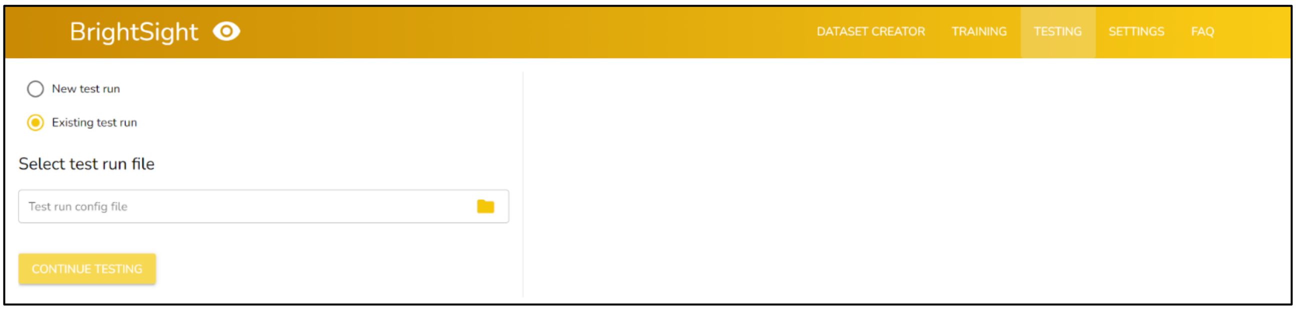
URL

Select models to test

□ YOLOv9

□ RT-DETR

Figure 10: Testing page, new test run.



○ New test run

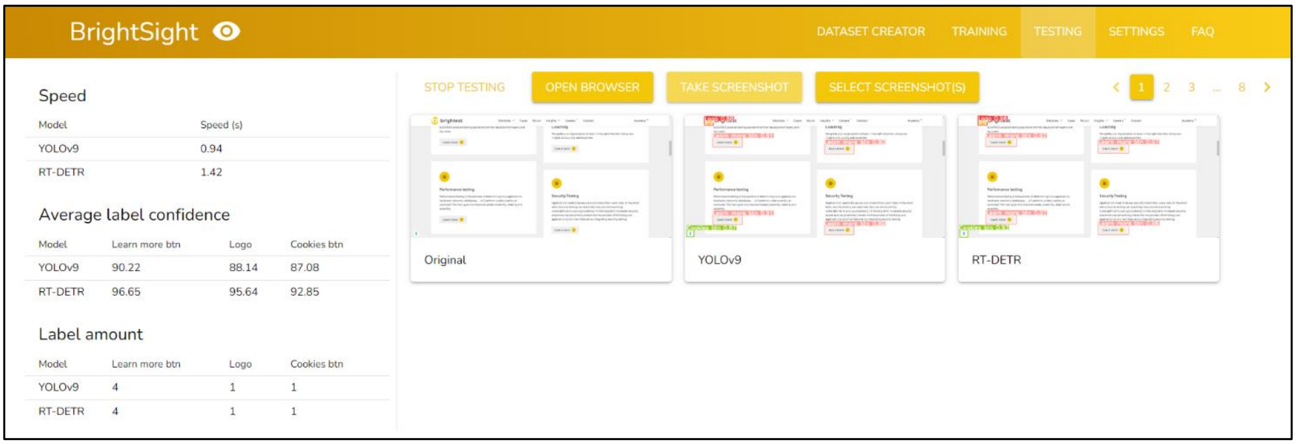
● Existing test run

Select test run file

Test run config file

Figure 11: Testing page, existing test run.

Once a test run is created or opened, three buttons appear: one to open the browser, one to take a screenshot, and one to select screenshots using the file picker. Screenshots can only be taken when the browser is opened. The browser is opened using Selenium. In here the user can use the website and at a certain point press the ‘take screenshot’ button. The screenshot will then be used as input for the models. Once completed (usually a couple seconds per model for YOLOv9 and RT-DETR) the original screenshot and the model outputs are shown (Figure 12). The output images contain bounding boxes with a label and confidence percentage. Selecting screenshots using the file picker works in the same way, except the browser isn’t necessary. An image can be enlarged by clicking it. In this view, navigation between the outputs is still possible using the menu on the bottom (Figure 13). Out of the enlarged image view, it is possible to navigate between screenshots using the menu in the top right corner. The page also shows tables containing the speed of each model, the average confidence and amount detected per label per model.



Speed

Model	Speed (s)
YOLOv9	0.94
RT-DETR	1.42

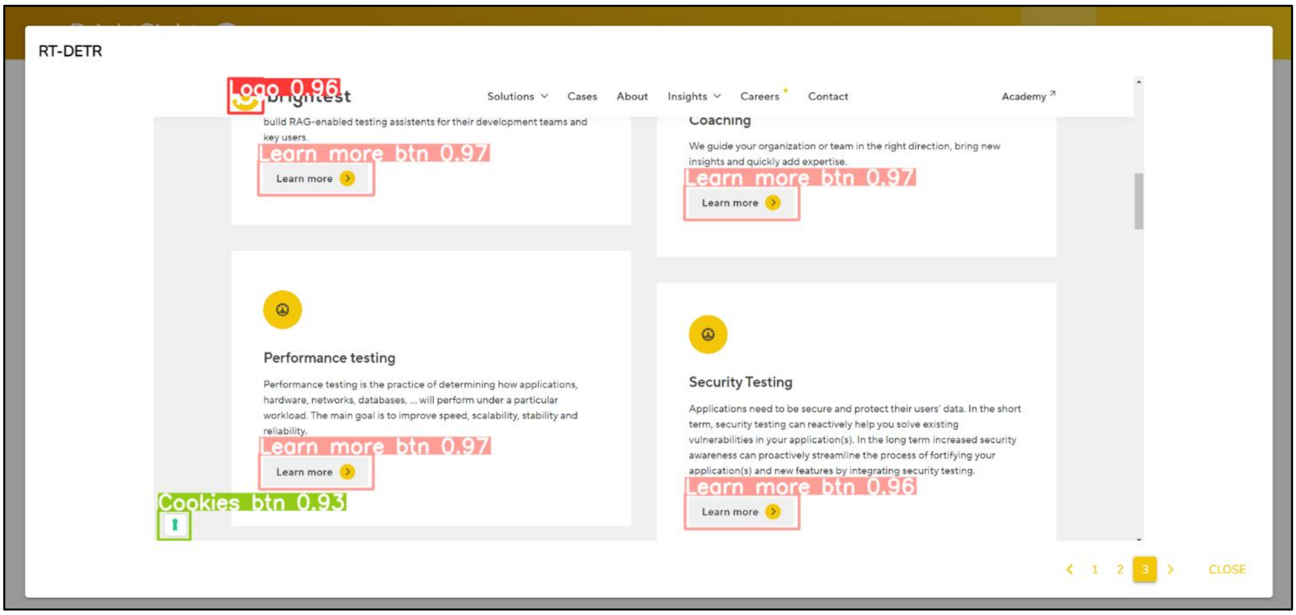
Average label confidence

Model	Learn more btn	Logo	Cookies btn
YOLOv9	90.22	88.14	87.08
RT-DETR	96.65	95.64	92.85

Label amount

Model	Learn more btn	Logo	Cookies btn
YOLOv9	4	1	1
RT-DETR	4	1	1

Figure 12: Testing page, after taking a screenshot.



build RAG-enabled testing assistents for their development teams and key users.

We guide your organization or team in the right direction, bring new insights and quickly add expertise.

Performance testing

Performance testing is the practice of determining how applications, hardware, networks, databases, ... will perform under a particular workload. The main goal is to improve speed, scalability, stability and reliability.

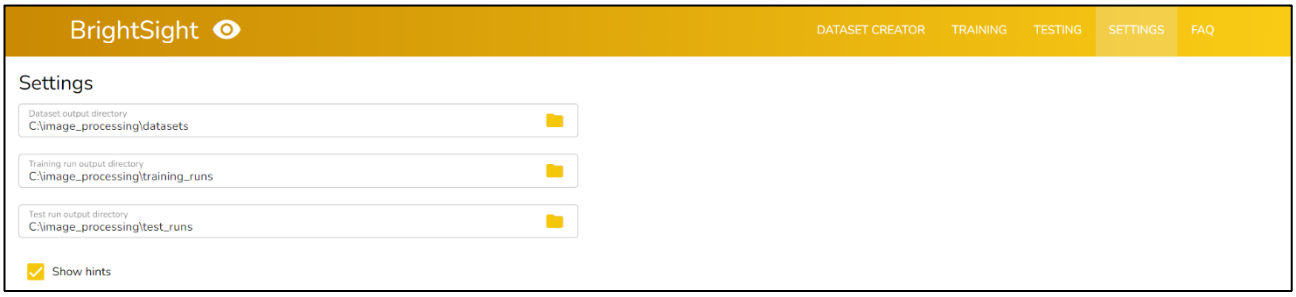
Security Testing

Applications need to be secure and protect their users' data. In the short term, security testing can reactively help you solve existing vulnerabilities in your application(s). In the long term increased security awareness can proactively streamline the process of fortifying your application(s) and new features by integrating security testing.

Figure 13: Testing page, enlarged image view.

3.2.4 Settings and FAQ

The settings page (Figure 14) allows the user to change the default directories for the datasets, trained models, and test runs. It also allows for the hints (icon with tooltip next to input field) to be turned off.



Settings

Dataset output directory: C:\image_processing\datasets

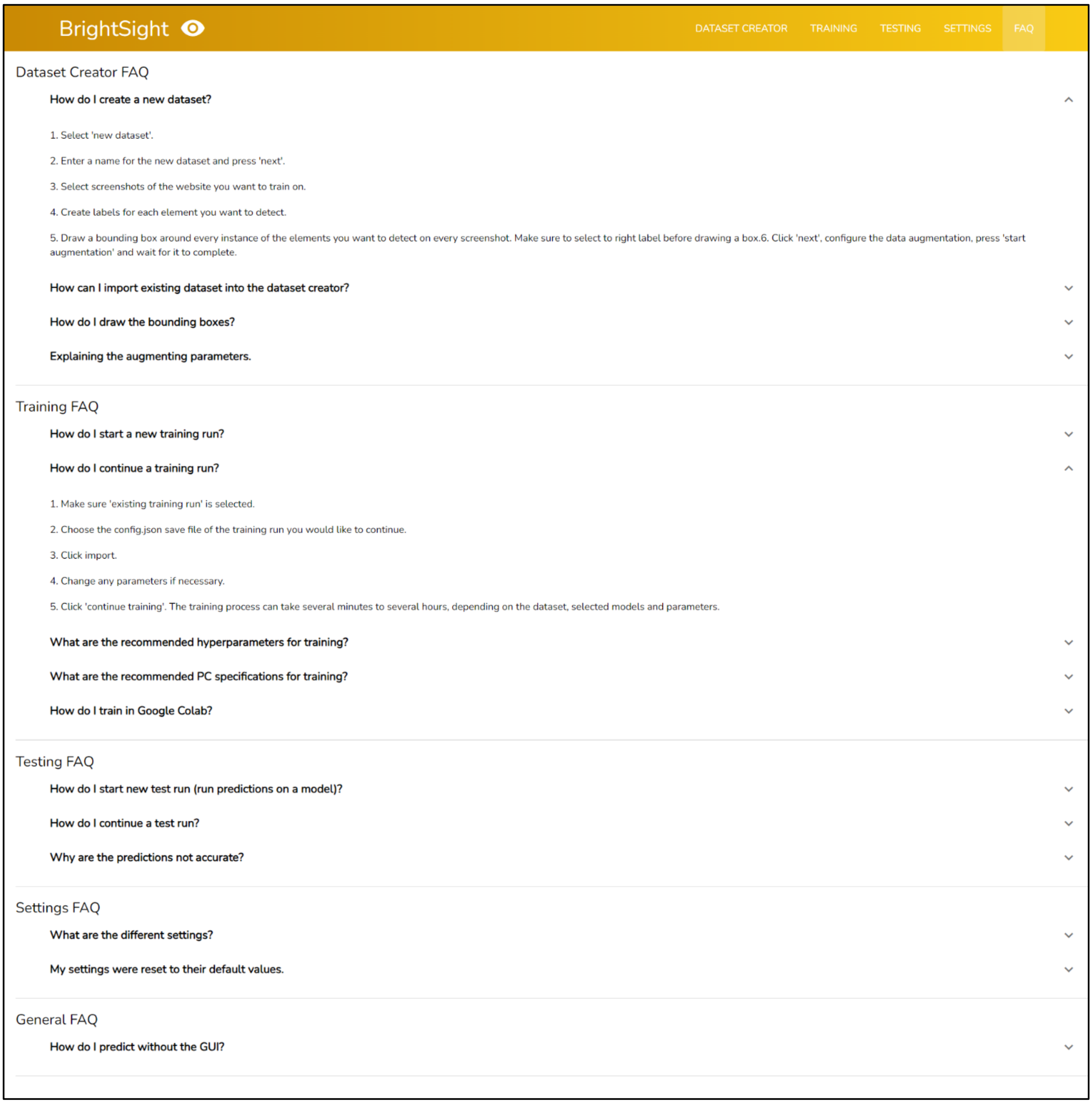
Training run output directory: C:\image_processing\training_runs

Test run output directory: C:\image_processing\test_runs

☑ Show hints

Figure 14: Settings page.

The frequently asked questions or FAQ page (Figure 15) contains answers to some questions that a new user might have on how to operate the application, sorted per page.



Dataset Creator FAQ

How do I create a new dataset?

1. Select 'new dataset'.
2. Enter a name for the new dataset and press 'next'.
3. Select screenshots of the website you want to train on.
4. Create labels for each element you want to detect.
5. Draw a bounding box around every instance of the elements you want to detect on every screenshot. Make sure to select to right label before drawing a box.6. Click 'next', configure the data augmentation, press 'start augmentation' and wait for it to complete.

How can I import existing dataset into the dataset creator?

How do I draw the bounding boxes?

Explaining the augmenting parameters.

Training FAQ

How do I start a new training run?

How do I continue a training run?

1. Make sure 'existing training run' is selected.
2. Choose the config.json save file of the training run you would like to continue.
3. Click import.
4. Change any parameters if necessary.
5. Click 'continue training'. The training process can take several minutes to several hours, depending on the dataset, selected models and parameters.

What are the recommended hyperparameters for training?

What are the recommended PC specifications for training?

How do I train in Google Colab?

Testing FAQ

How do I start new test run (run predictions on a model)?

How do I continue a test run?

Why are the predictions not accurate?

Settings FAQ

What are the different settings?

My settings were reset to their default values.

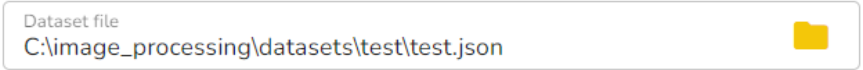
General FAQ

How do I predict without the GUI?

Figure 15: FAQ page.

3.2.5 Quality of life features

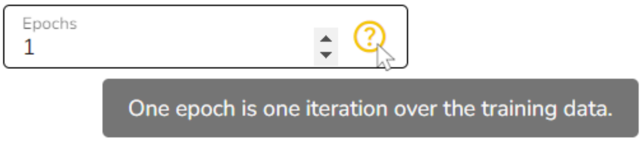
The application implements several quality-of-life (QoL) features that users have come to expect of modern applications. The first of these features is the file picker input field (Figure 16). These are text input fields with a folder button which opens the file picker when it is clicked. The text in the input field changes according to the location the user selected, but it also allows the user to manually type in the path. Moreover, autocomplete has been implemented. When 'ctrl' and 'enter' are pressed at the same time while the input field is focussed, it automatically attempts to complete the path if possible. The tab key could not be used for autocompletion since it is used by the browser to swap focus between inputs.



Dataset file
C:\image_processing\datasets\test\test.json

Figure 16: File picker input field.

Secondly, some inputs are accompanied by a question mark icon that, when hovered over with the cursor, displays a tooltip regarding the input (Figure 17). It does introduce some visual clutter however, which is why it is possible to hide the hint icons in the settings.

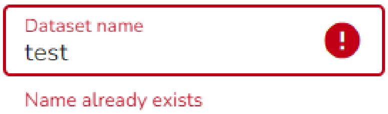


Epochs
1

One epoch is one iteration over the training data.

Figure 17: Input hint.

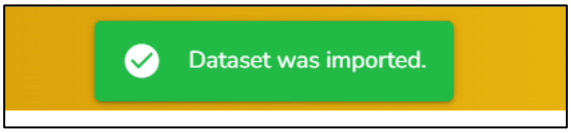
Next, most of the input fields have some sort of validation, making it harder for the user to make mistakes. More importantly however, throughout the entire application positive and negative feedback is used to guide the user and allow them to correct the problem. Figure 18 shows that input fields display a detailed error message when the current value isn't valid. Examples of a positive and negative notification are shown in Figure 19 and Figure 20 respectively. These are used to give feedback on actions that don't necessarily involve an input field and automatically disappear after 5 seconds.



Dataset name
test

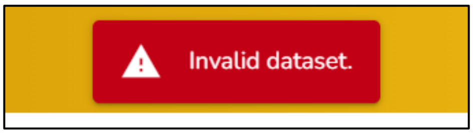
Name already exists

Figure 18: Error message when input is invalid.



Dataset was imported.

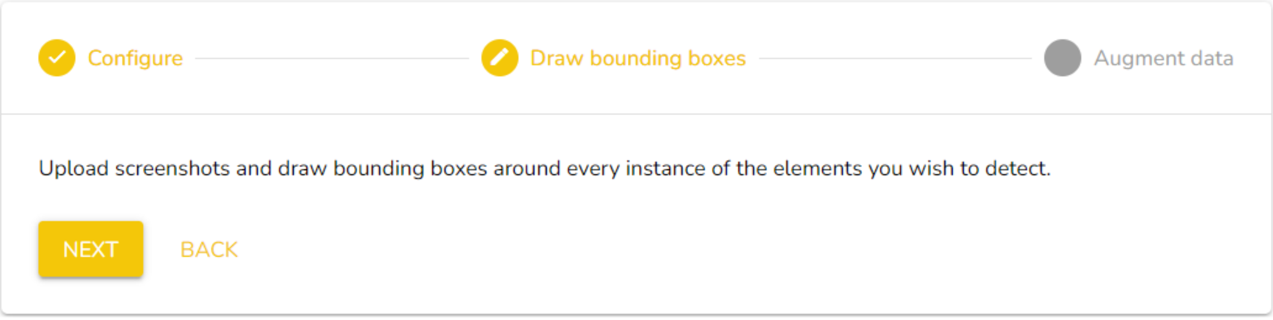
Figure 19: Positive notification on successful action.



Invalid dataset.

Figure 20: Negative notification on failed action.

In addition, the dataset creator contains what is known as a 'stepper'. This serves as a guideline through a process that most users will be unfamiliar with (Figure 21). Even though there are only three steps to creating a dataset, the stepper allows for some useful information to be displayed about the current step. It also creates more separation between the three steps, rather than having them all on a single page which could be overwhelming.



Upload screenshots and draw bounding boxes around every instance of the elements you wish to detect.

Figure 21: The stepper inside the dataset creator.

Furthermore, the canvas that displays the screenshot in the dataset creator allows for panning and zooming of the screenshot using the right mouse button and the scroll wheel respectively. This is not functionality that is available by default in the NiceGUI framework and thus had to be custom made. There are also buttons to control the zoom level or to completely reset the canvas view.

Finally, care has been taken to ensure that as many operations as possible can be edited and are not permanent or require starting over. The three main operations (dataset creation, training, testing) are saved in config files and can be imported to continue working on later. Other elements can be edited, for example the bounding boxes in the dataset creator can be moved and resized and do not require the user to delete the box and create a new one. Another example is the training parameters, which can be changed even when continuing a training run.

3.3 Usage without GUI

Since the future goal would be to use the application in an automated testing process, prediction must be available to be executed from code rather than using the GUI. The model classes were designed with this in mind, the classes function completely separate from the rest of the operation. Thus, it is possible to simply import the model class and call the predict method. However, a predict file has been provided to make the process more convenient. The file contains a predict function that can be used to predict on one or more images using one or more models and an optional label filter. In addition to calling this method by importing the file, it is also possible to run the predict file in the command line and provide command line arguments to perform the prediction. This way, the automated testing process doesn’t necessarily need to be inside a Python environment.

An example of making predictions in an automated testing process was constructed. In this example, Selenium is used to interact with the Brightest website and test some buttons. Due to Selenium for Python not allowing full-page screenshots to be taken, some way to get a specific button into the viewport is needed. This was done using the XPath for the buttons to be tested. Once a button is scrolled into the viewport, a screenshot is taken and used for prediction. The label of the buttons is passed to the prediction method to filter out all the other labels. The position of the detected button (if one is detected) is compared with the position of the button according to Selenium. If the positions are within a certain margin of error, the button has been detected. Selenium is then used to click at the center of the detected button. After navigating to the new page, the URL and page title are tested. Then, the same process to detect the earlier button is used to detect the button that will navigate back to the home page. This process is repeated for all the ‘solutions’ on the Brightest home page.

3.4 Deployment

Since the application is not a website, it cannot be deployed as a website. This would simply allow the user to access the server on which the application is running, which is not something desirable for a website. The other option for deployment is to package the application as an executable. This can be achieved using the PyInstaller package. Running the build file packages the application and results in a single executable that can be run without installing a Python interpreter or any modules [35]. However, the executable does only work on the operating system on which it was created.

3.5 Application primary flow

A diagram of the primary flow is shown in Figure 22. The flow consists of three main steps, starting with the dataset creator (section 3.2.1). In here a new dataset can be created or an existing dataset can be opened and modified. The dataset is created by drawing bounding boxes on screenshots of the website and performing data augmentation. The grey box contains operations that aren’t connected because a multitude of operations can (but don’t have to) happen in an order that cannot be determined beforehand. The box simply represents that at this point in the flow, any of those actions can be performed in any order, until all the bounding boxes have been drawn. The next step is training (section 3.2.2), either starting from scratch or using an existing training run save file. Training in Google Colab was not adopted into the primary flow since it is outside of the application and is largely equivalent to training in the application. A name for the training run must be entered and the models to train must be selected. The dataset file (output from previous step) must be set, as well as the number of epochs to train, the batch size, and the input image size. After training a model it is possible to test it inside the application (section 3.2.3). This is done by selecting which training run to use (output from the previous step) and which URL the browser should open. It is also possible to open a saved test run and continue working with it. The browser is opened by Selenium when the 'open browser' button is pressed. Screenshots of the website can be taken via the application or uploaded manually (which doesn't require the browser to be open). The application then runs inference on the screenshot(s) using all the selected models. The output screenshots are shown in the application and saved.

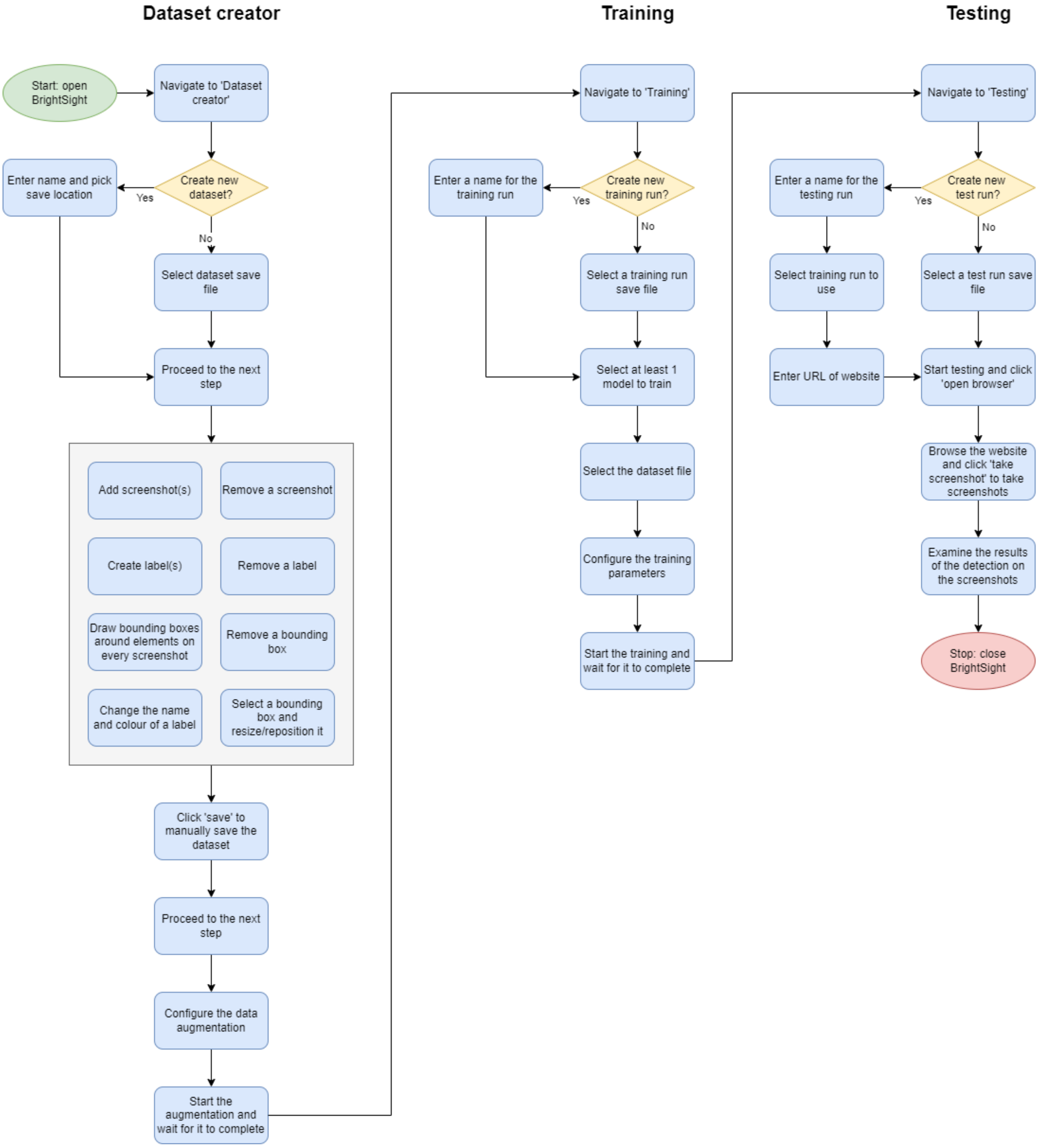
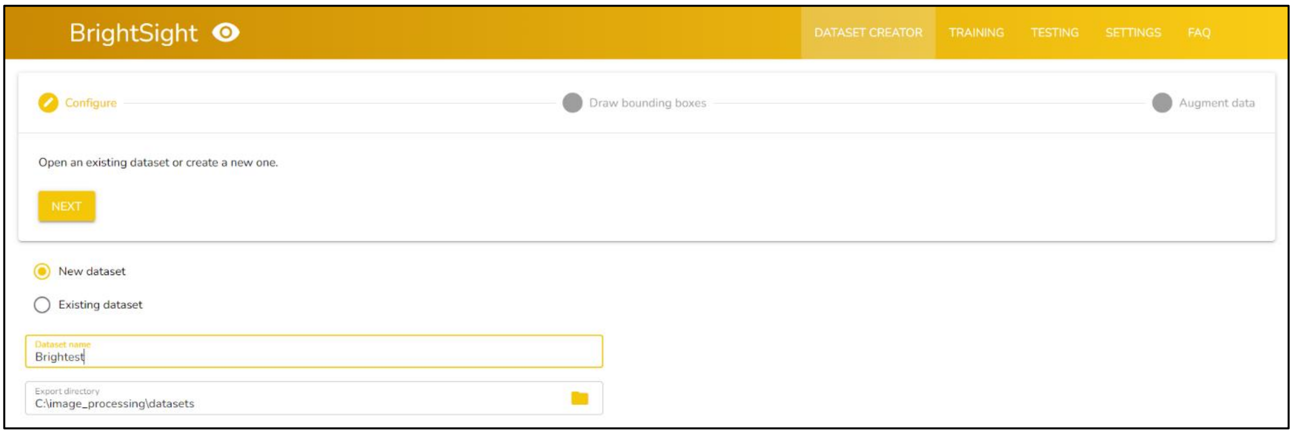

Dataset creator
Training
Testing
Start: open BrightSight
Navigate to 'Dataset creator'
Create new dataset?
Yes
No
Enter name and pick save location
Select dataset save file
Proceed to the next step
Add screenshot(s)
Remove a screenshot
Create label(s)
Remove a label
Draw bounding boxes around elements on every screenshot
Remove a bounding box
Change the name and colour of a label
Select a bounding box and resize/reposition it
Click 'save' to manually save the dataset
Proceed to the next step
Configure the data augmentation
Start the augmentation and wait for it to complete
Navigate to 'Training'
Create new training run?
Yes
No
Enter a name for the training run
Select a training run save file
Select at least 1 model to train
Select the dataset file
Configure the training parameters
Start the training and wait for it to complete
Navigate to 'Testing'
Create new test run?
Yes
No
Enter a name for the testing run
Select training run to use
Enter URL of website
Select a test run save file
Start testing and click 'open browser'
Browse the website and click 'take screenshot' to take screenshots
Examine the results of the detection on the screenshots
Stop: close BrightSight

Figure 22: Primary flow diagram of the application.

3.6 Case study

This section details the usage of the BrightSight application by providing an example. Once BrightSight has started, the dataset creator is opened (Figure 23). The option 'new dataset' is selected, and a name and location are selected. Then the 'next' button is clicked.



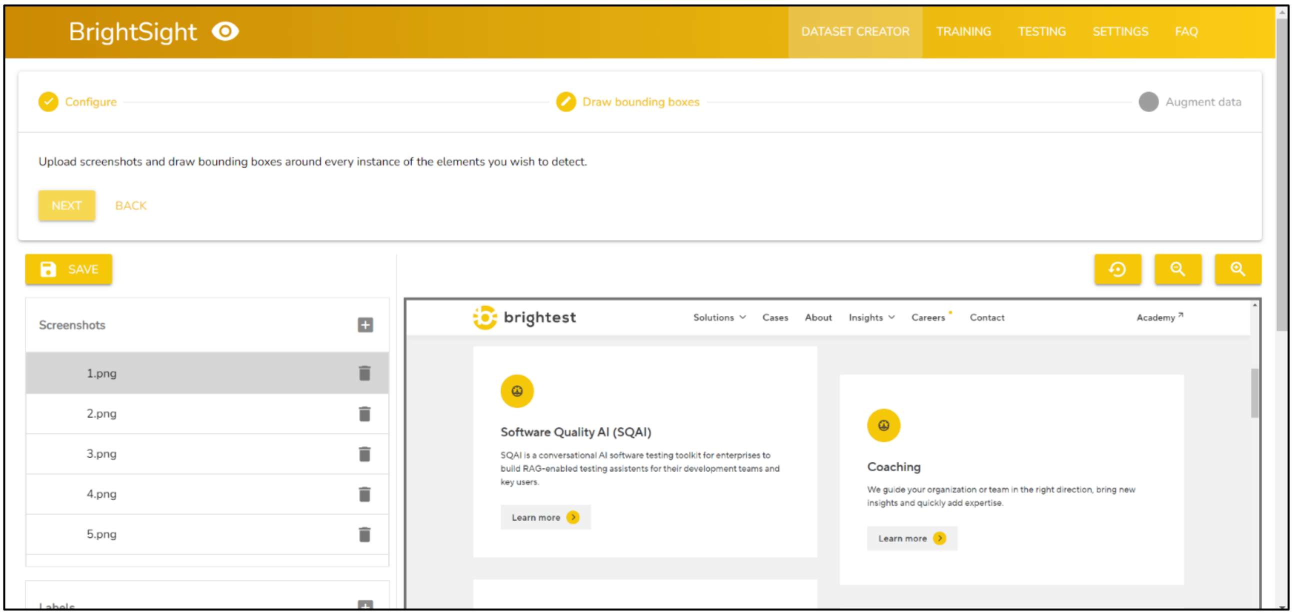
Open an existing dataset or create a new one.

● New dataset
○ Existing dataset

Dataset name: Brightest
Export directory: C:\image_processing\datasets

Figure 23: Case study step 1; dataset configuration.

Next, screenshots of the website are selected by clicking the plus icon in the screenshots list. More screenshots are usually better, though it is good to try to keep the element distribution even. If the dataset consists of 50% learn more buttons and 50% other elements, that is not an ideal composition.



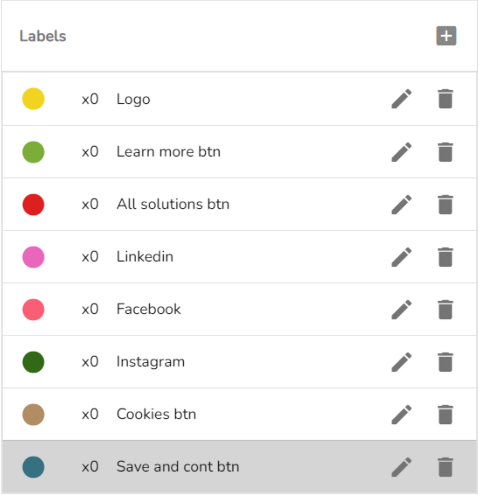
Upload screenshots and draw bounding boxes around every instance of the elements you wish to detect.

SQAI is a conversational AI software testing toolkit for enterprises to build RAG-enabled testing assistents for their development teams and key users.

We guide your organization or team in the right direction, bring new insights and quickly add expertise.

Figure 24: Case study step 2; uploading screenshots.

After selecting screenshots, the labels are created. Though of course it is still possible to add or change labels during rest of the dataset creation process. Figure 25 show the labels that were created for the elements to be detected. The label list also shows how many bounding boxes have been drawn for each label. This can be used to create a more evenly distributed dataset. However, *all* instances of every label on every screenshot must get a bounding box. If this distribution is off, add or remove screenshots to even it out. If not every instance has a bounding box, the model will receive conflicting information.



Labels

● x0 Logo

● x0 Learn more btn

● x0 All solutions btn

● x0 Linkedin

● x0 Facebook

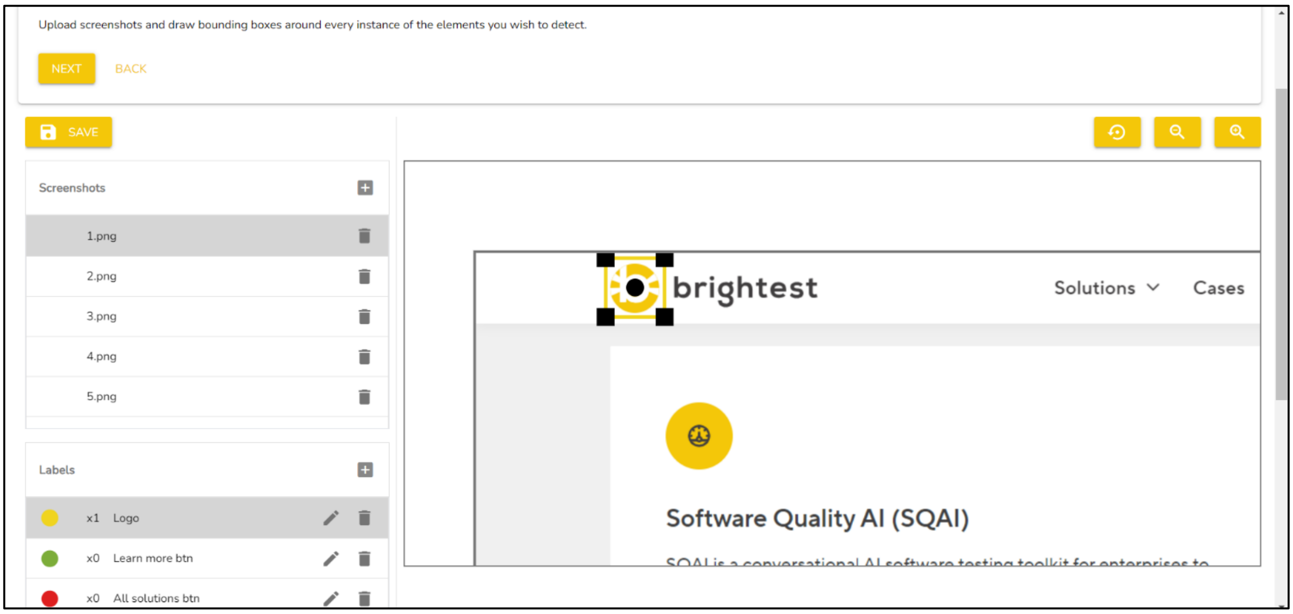
● x0 Instagram

● x0 Cookies btn

● x0 Save and cont btn

Figure 25: Case study step 3; creating labels.

Once the labels have been created, bounding boxes can be drawn. As mentioned, every instance of every label on every screenshot must get a bounding box. The screenshot can be selected by clicking on it in the screenshot list. Then a label is selected, after which the box is drawn around the element. To have more detailed control on smaller elements, the canvas can be zoomed in by scrolling, and the screenshot can be moved around by dragging with the right mouse button pressed. Clicking on the box also allows to change the dimensions and position of the box. It is good to save the dataset every couple bounding boxes so not too much progress is lost in case something goes wrong. Figure 27 shows an example of a completed screenshot with the bounding boxes around every element to be detected. Once all boxes have been drawn, the 'next' button is clicked.



Upload screenshots and draw bounding boxes around every instance of the elements you wish to detect.

Screenshots

1.png

2.png

3.png

4.png

5.png

Labels

● x1 Logo

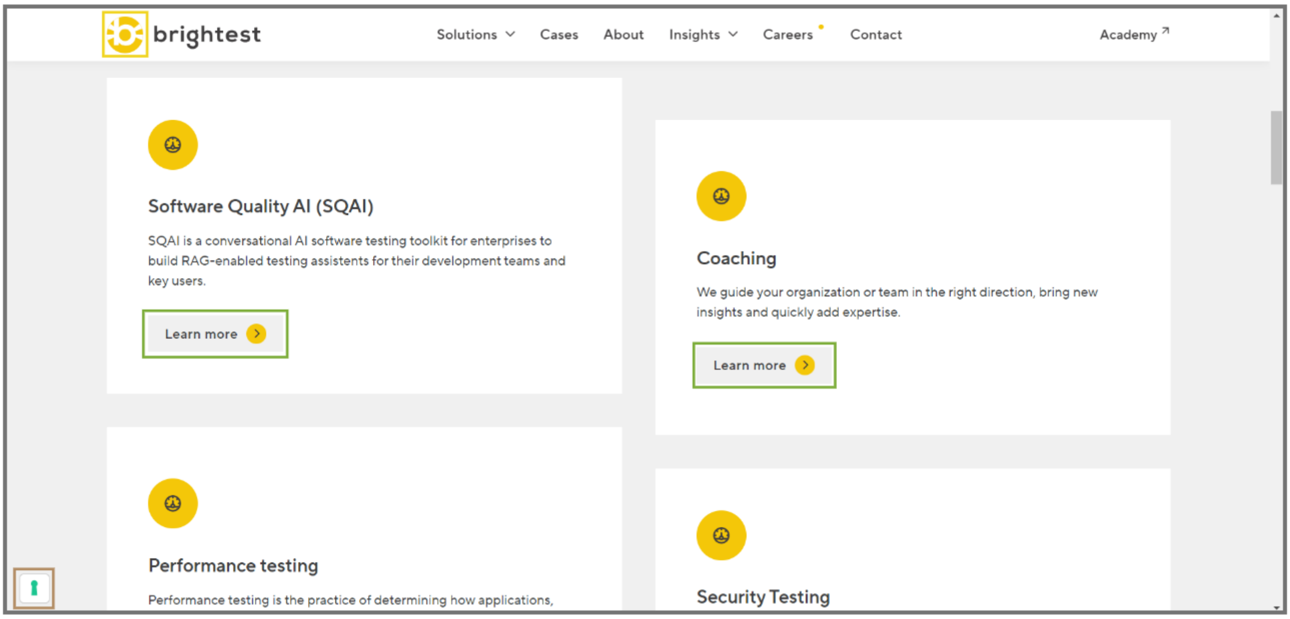
● x0 Learn more btn

● x0 All solutions btn

Software Quality AI (SQAI)

SQAI is a conversational AI software testing toolkit for enterprises to

Figure 26: Case study step 4; drawing bounding boxes.



Software Quality AI (SQAI)

SQAI is a conversational AI software testing toolkit for enterprises to build RAG-enabled testing assistents for their development teams and key users.

Coaching

We guide your organization or team in the right direction, bring new insights and quickly add expertise.

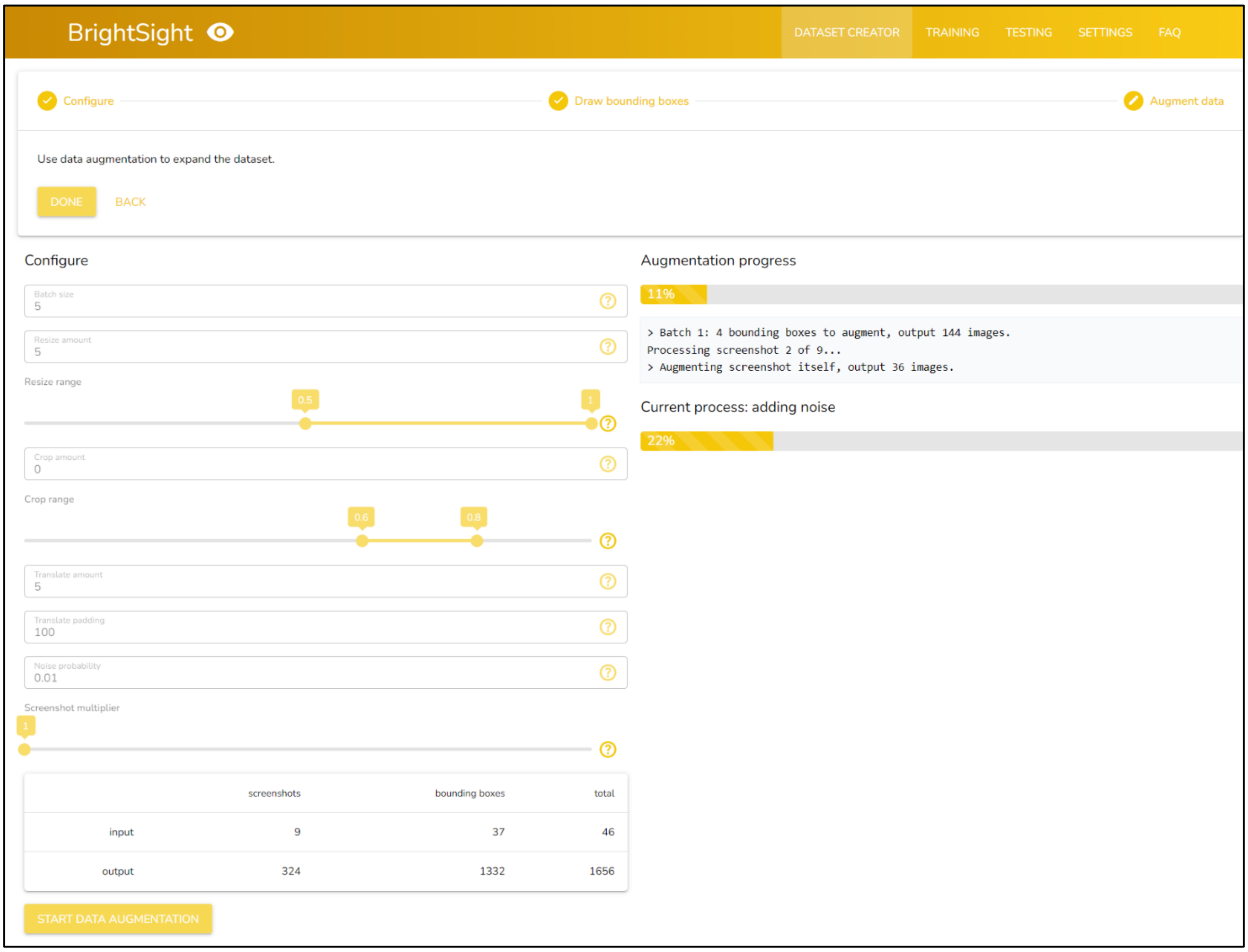
Performance testing

Performance testing is the practice of determining how applications,

Security Testing

Figure 27: Case study step 4; example of completed screenshot.

Most of the data augmentation settings are kept at default. Only the resize range is changed to 0.5-1.0 since most of the elements are on the smaller size. After data augmentation the dataset contains a total of 1656 images. The entire dataset creation process took around 20 minutes (though it is a relatively small dataset).



Use data augmentation to expand the dataset.

Configure

Batch size: 5

Resize amount: 5

Resize range: 0.5 – 1

Crop amount: 0

Crop range: 0.6 – 0.8

Translate amount: 5

Translate padding: 100

Noise probability: 0.01

Screenshot multiplier: 1

	screenshots	bounding boxes	total
input	9	37	46
output	324	1332	1656

Augmentation progress

11%

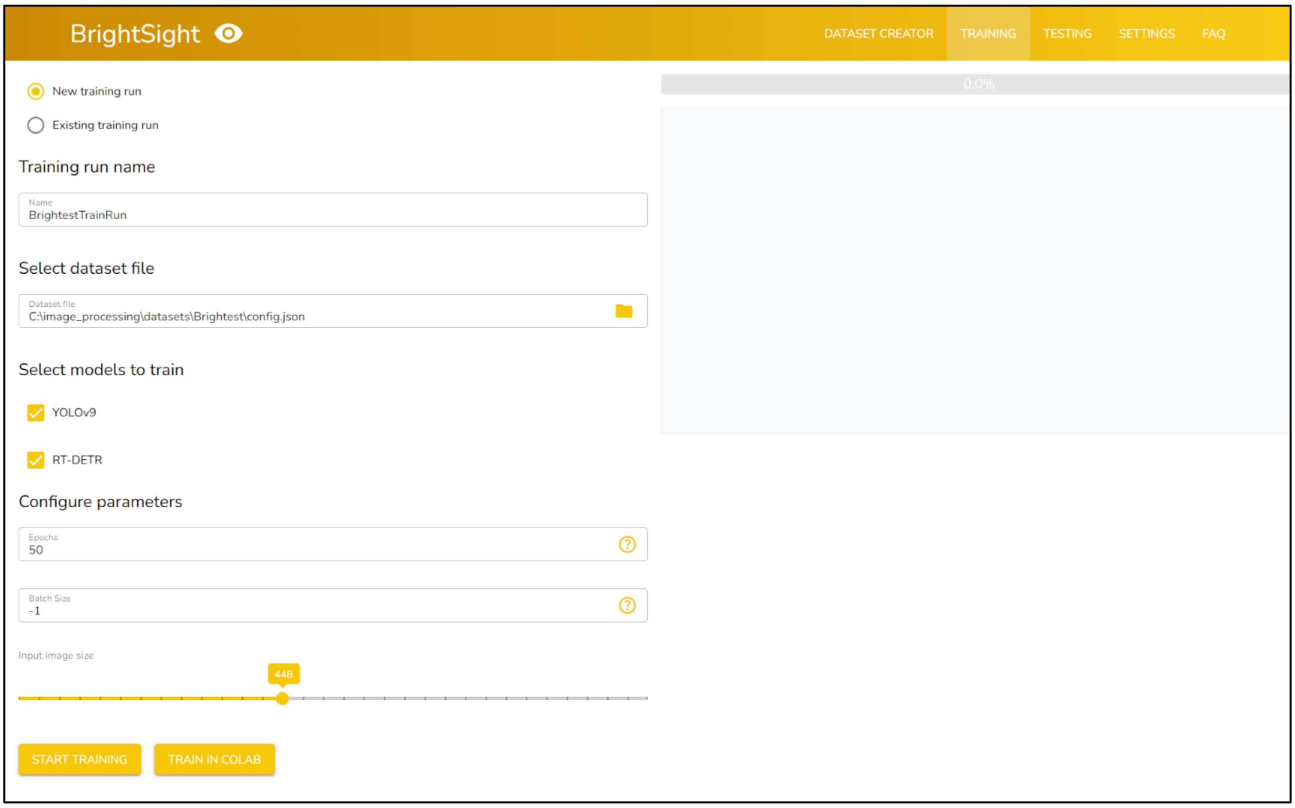
```
> Batch 1: 4 bounding boxes to augment, output 144 images.
Processing screenshot 2 of 9...
> Augmenting screenshot itself, output 36 images.
```

Current process: adding noise

22%

Figure 28: Case study step 5; data augmentation.

With the newly created dataset, the training process can be started. The 'new training run' option, a name, and the dataset are selected. Due to the lack of a GPU, training will be done in Colab. Because of this the other configuration options don't have to be set yet. When the 'train in Colab' button is clicked and output location selected, a notification appears at the top of the page with the message that zipping is in progress. After this is done, the notebook in the output location is opened in Colab and a GPU runtime is started (Figure 30). While the two zip files from the output location are being uploaded, the training parameters are set. Both YOLOv9 and RT-DETR are selected for training and since both train relatively fast, the number of epochs is set at 50. The batch size is set at -1 to enable auto-batching. The input image size is set at 448 (from experience, YOLOv9 doesn't do very well with a smaller input image size). After setting al the parameters, the cell is run by clicking the play button.



● New training run

○ Existing training run

Training run name

Name: BrightestTrainRun

Select dataset file

Dataset file: C:\image_processing\datasets\Brightest\config.json

Select models to train

☑ YOLOv9

☑ RT-DETR

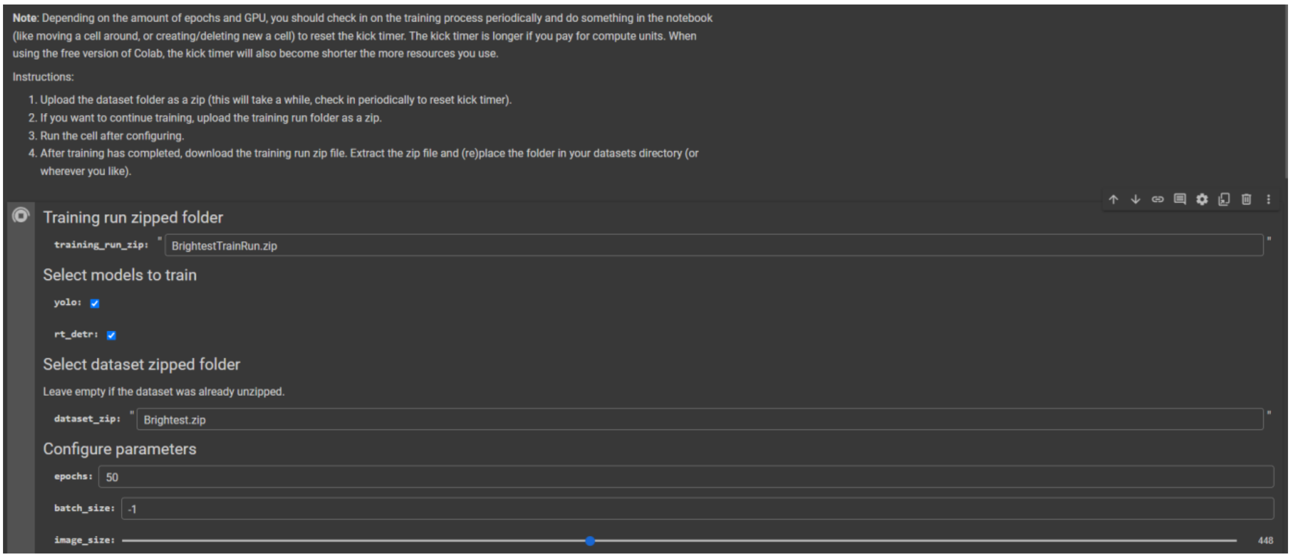
Configure parameters

Epochs: 50

Batch Size: -1

Input image size: 448

Figure 29: Case study step 6; training configuration.



Note: Depending on the amount of epochs and GPU, you should check in on the training process periodically and do something in the notebook (like moving a cell around, or creating/deleting new a cell) to reset the kick timer. The kick timer is longer if you pay for compute units. When using the free version of Colab, the kick timer will also become shorter the more resources you use.

Instructions:

1. Upload the dataset folder as a zip (this will take a while, check in periodically to reset kick timer).
2. If you want to continue training, upload the training run folder as a zip.
3. Run the cell after configuring.
4. After training has completed, download the training run zip file. Extract the zip file and (re)place the folder in your datasets directory (or wherever you like).

Training run zipped folder

training_run_zip: " BrightestTrainRun.zip "

Select models to train

yolo: ☑

rt_detr: ☑

Select dataset zipped folder

Leave empty if the dataset was already unzipped.

dataset_zip: " Brightest.zip "

Configure parameters

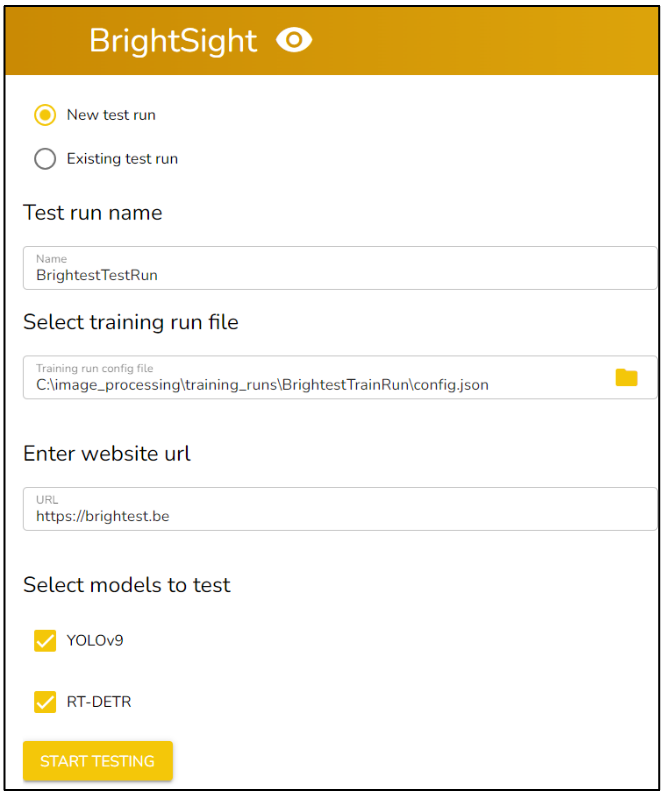
epochs: 50

batch_size: -1

image_size: 448

Figure 30: Case study step 6; training in Colab.

After training for 1 hour and 45 minutes using a T4 GPU, the models can be tested. First, the training run zip is downloaded an unzipped in the default training run location. The testing page is opened and 'new test run' is selected (Figure 31). A name is given, and the training run config file is selected. The website on which predictions will be made is the Brightest website and both models will be tested.



● New test run
○ Existing test run

Test run name

Name: BrightestTestRun

Select training run file

Training run config file: C:\image_processing\training_runs\BrightestTrainRun\config.json

Enter website url

URL: https://brightest.be

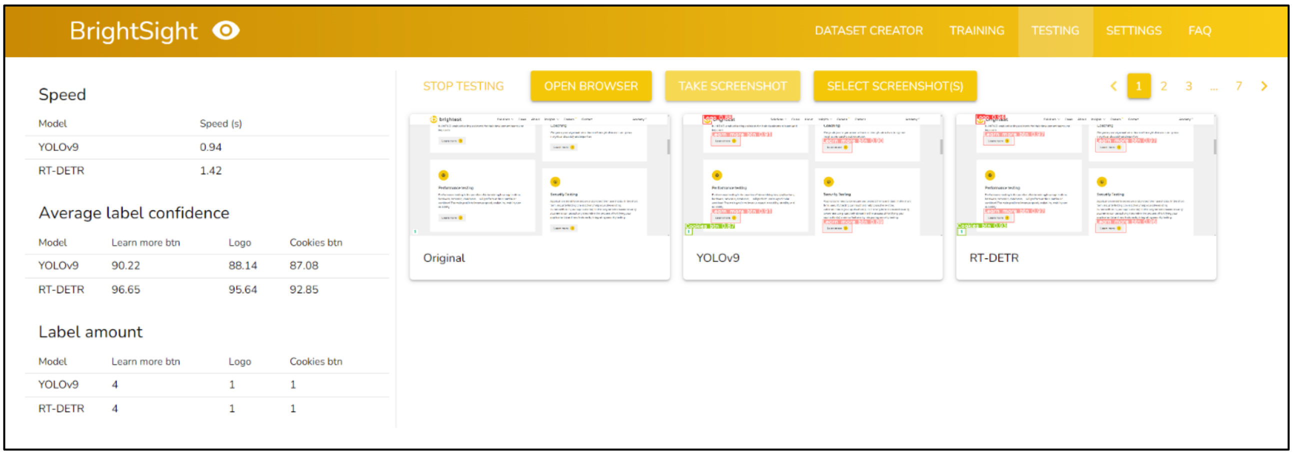
Select models to test

☑ YOLOv9

☑ RT-DETR

Figure 31: Case study step 7; testing configuration.

After configuring the test run, several new screenshots of the website are selected for testing. After a couple minutes (on an HP Elitebook 850 G5), the results are shown (Figure 32). Both models seem to perform well, detecting all the elements at high confidence levels (Figure 33). A couple mistakes were present on one of the elements that had accounted for a much smaller portion of the dataset than other elements. This makes it clear why an evenly distributed dataset, preferably with multiple instances per element, can be important.



Speed

Model	Speed (s)
YOLOv9	0.94
RT-DETR	1.42

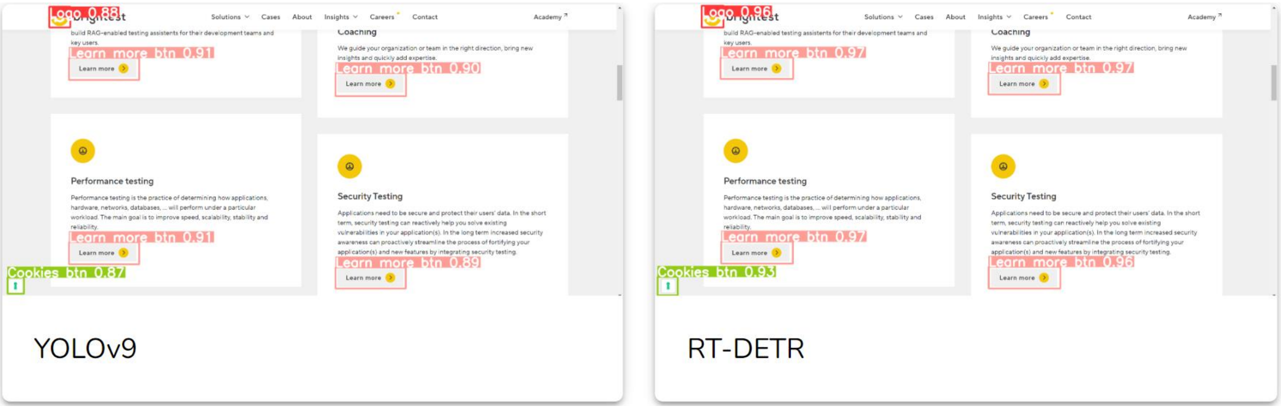
Average label confidence

Model	Learn more btn	Logo	Cookies btn
YOLOv9	90.22	88.14	87.08
RT-DETR	96.65	95.64	92.85

Label amount

Model	Learn more btn	Logo	Cookies btn
YOLOv9	4	1	1
RT-DETR	4	1	1

Figure 32: Case study step 8; testing.



build RAG-enabled testing assistents for their development teams and key users.

We guide your organization or team in the right direction, bring new insights and quickly add expertise.

Performance testing is the practice of determining how applications, hardware, networks, databases, ... will perform under a particular workload. The main goal is to improve speed, scalability, stability and reliability.

Applications need to be secure and protect their users' data. In the short term, security testing can reactively help you solve existing vulnerabilities in your application(s). In the long term increased security awareness can proactively streamline the process of fortifying your application(s) and new features by integrating security testing.

Figure 33: Case study step 8; one of the testing results.

Once the models have been tested and their performance deemed satisfactory, the application is no longer of use. The models can now be used in an automated testing process like, for example, the prediction demo mentioned in section 3.3.

4 Discussion

4.1 Implementation of models

The architecture of the application is set up in such a way that new models can be added very easily. A new model class must simply inherit the abstract base model class and implement its methods. The class must then be instantiated inside the model getter method in the training class, and it will be automatically added to the application in all the required places.

4.1.1 YOLOv9 and RT-DETR

The implementation of YOLOv9 and RT-DETR was very simple. The usages of YOLOv9 and RT-DETR are very similar to each other, but very different from Co-DINO. This makes it clear why it is so important to construct a good base class that is independent of the implementation of its child classes (even though Co-DINO isn't used in the application). The interactions between the base class and the rest of the application stays the same, while the implementation of the child classes can vary wildly. Since YOLOv9 and RT-DETR are almost identical in terms of usage, a second base class was created for Ultralytics models that works in this same way.

4.1.2 Co-DINO

It is unfortunate that Co-DINO could not be implemented in the application, especially because it is the state-of-the-art object detection model at the time of writing this. However, there were simply too many problems which caused training to start functioning later than the other models. At the point in time where training was working, there was not enough time left to troubleshoot this model and other aspects of the project had a higher priority. Co-DINO also took about 30 minutes to install in Colab, after which it would still take around an hour per epoch. Consequently, trying out different datasets without a concrete idea of the problem consumes a lot of time.

The main problem presenting itself with Co-DINO was high accuracy in training but almost no detection after training. This issue of having great accuracy during training but bad detection during prediction is usually an indication of overfitting. However, both other models did not have this problem with the same dataset. Even when predicting on one of the training images, Co-DINO detects nothing. Possible future work would be to create an even more varied dataset, which as mentioned in section 7 is difficult for this use case. Though it is also entirely possible that a simple mistake was made.

4.2 Application

4.2.1 Dataset creator

The dataset creator is an important part of the proof-of-concept application. Not much active time is spent on training and testing, the dataset creator requires the most interaction with the user. Though solutions exist for creating datasets by drawing bounding boxes, they are usually behind a paywall and do not always perform data augmentation. It is also more convenient for the user when everything is contained in a single application and thus there is no need to go to a third-party application. Furthermore, it makes the user experience more consistent between users and allows for more control over the quality of the datasets.

At the very start of development Tkinter was used instead of NiceGUI, but the dated and simplistic look of the GUI prompted a switch to NiceGUI and its contemporary look allowed by the web-based approach. The framework made it very easy to get started and yet it allows some pretty complex things to be achieved, as evidenced by the dataset creator. This complexity is supported by their use of the Vue, Quasar and Tailwind frameworks. However, this also leads into what might be NiceGUI's biggest problem: incomplete and fragmented documentation. Some things are not covered in the NiceGUI documentation but instead in the Quasar or Tailwind documentation. It is then required to figure out how to apply this documentation to NiceGUI. Another positive point is NiceGUI's modularity, which makes it easy and convenient to split a page into reusable and encapsulated elements represented by classes. Finally, the framework is not very established yet online, making it difficult to find people encountering the same issues. In conclusion, NiceGUI made it relatively convenient to make a modern looking, responsive GUI with some complex elements.

Care was taken to make the user experience of the dataset creator as smooth as possible with the tools at hand and within the allotted time. Quality-of-life features that users have come to expect of applications were added wherever fitting, though of course not all possible features have been exhausted. Error prevention and handling is applied as much as possible throughout the application. Saved files are checked for validity before being imported. Input fields perform validation and display error messages when necessary. Important results are accompanied by a notification toast on the top of the website.

The dataset creator ended up almost exactly as first envisioned and is very useful, even being used during development to create datasets. The creator can be used to create datasets for models that support the COCO annotation format or the Ultralytics input data format, making it useful even outside the field of web element detection.

Possible future work would include the optimization of the canvas, since it is not uncommon for the canvas to lag slightly. The robustness of the dataset creator, especially when it comes to responsivity, could also be increased.

4.2.2 Training

Training in the application is very straight-forward: selecting a dataset, configuring the training parameters, and then simply waiting. Therefore, there is not much to be said about the training page other than that it works. It is possible to train the models starting from a dataset created in the dataset creator and end up with a model with high accuracy. In addition, it is possible to generate data to be able to train in a Google Colab notebook. Future work would be to add a clean way to stop the training before its natural end, rather than exiting the application.

4.2.3 Testing

The testing page allows the user to review whether the trained models are functional and how well they are performing. It was important to be able to view the results of the different models simultaneously to make it easier to compare them. This was achieved by displaying numerical results in tables and using thumbnails for the visual results which can be enlarged for more detail. Possible future work would be to add an option to generate the code that can be used to predict without using the GUI. Especially since there are a few paths that the prediction method requires, which are somewhat cumbersome to enter manually.

4.2.4 Settings and FAQ

Both the settings and FAQ page are not very intricate, but they contribute significantly to the user experience. The default directory locations in the settings allow the user to get where they want to go much quicker while navigating the file picker. In addition, more experienced users can turn off the hints to reduce visual clutter and have a cleaner experience. Future work would simply be to add more settings for customizing the user experience.

The FAQ page is a ubiquitous concept that has been applied to many websites. Since the process that the application facilitates isn't common knowledge, a FAQ can be helpful for users experiencing some confusion. Future work would be to add a search bar so the user doesn't have to scroll through the entire page to find what they need.

4.2.5 Quality of life features

Even though it is a proof-of-concept, it was still important for the application to be easy and convenient to use. One way of ensuring that the user experience is positive is to add QoL features. Most of the QoL features in the application are small but they make a significant, positive impact on the usability. Future work would simply be to add more of these QoL features. For example, allowing the option to collapse the stepper in the dataset creator could improve the experience for more advanced users.

4.3 Usage without GUI

Being able to perform prediction trough code instead of a GUI was important since the latter would not be much use in an automated testing process. For dataset creation and training this is not necessary because it is not part of the automated testing process but rather something that ideally is only done once for each website. The application facilitates the dataset creation and training process in a way that is much easier to use than if it had to be done exclusively through code.

4.4 Deployment

Deploying the application as a website would be ideal since it could be deployed on a server with an assortment of powerful GPUs to speed up the training process. However, this would increase the scope of the project too much and perhaps more importantly, no server with GPUs was made available for this project. Turning the application into a website would require an algorithm to share the GPUs between concurrent users. Also, there would have to be a way to store the trained models (which would require a lot of memory for use by multiple people), or users must go through the trouble of downloading and uploading files of multiple GBs every time the application is used (Google Colab does not save files that were uploaded to the runtime). The former option would then also require an account system to allow users to access their own models, and to stop non-authorized persons from accessing the application. Thus, an executable was the second-best option. The user simply runs the build script and after some time the executable is ready. Or alternatively, if the executable is downloadable somewhere, the user doesn't even have to run the build script or have Python installed.

4.5 Primary flow

The primary flow consists of three main steps which don't have to be completed in one go. It is even possible to save inside of one of the main steps and continue later. The flow is simple and consistent thanks to the separation into three steps as well as every step consisting of a relatively simple sequence of operations. Each main step (except the first) always uses the output of a previous step, which makes it feel natural.

4.6 Case study

The case study gives an entire overview of the usage of BrightSight using an example. It shows how the process of creating a database and training models can be easy and convenient, even for users who aren't entirely familiar with machine learning (though some basic knowledge about its purpose is helpful). In a little over two hours two very strong models to detect a variety of patterns were created. This means that it is certainly a possibility to re-train the models on a per sprint basis during development. The rest of this section will go into more detail on the positive points and possible points of improvement.

The dataset creator is the part of the application that requires the most interaction with the user. Therefore, the user actions are restricted to the absolute bare necessities to be able to complete the task, while also providing plenty of feedback and a modern user interface. This way the niche task of drawing bounding boxes around objects becomes less foreign. The canvas unfortunately must do a lot of work and consequently can lag sometimes while drawing a box. This is a possible point of optimization.

The fact that the entire process can always be saved and returned to later is very convenient. If this was not a feature, the application would certainly be a drag to use. In addition, the positive and negative feedback throughout the application makes the user feel less helpless and puts their mind at ease while performing an action.

It is unfortunate that the user must leave the application to be able to train in Google Colab. Ideally the application would be run on a server with powerful GPUs that would also save all the datasets and models. This is a much larger undertaking, however. All-in-all the process is very easy and convenient, though it has not been tested by users unfamiliar with the application. Which would certainly provide more useful insights and optimizations.

5 Conclusion

This application note has presented ‘BrightSight’, an application to facilitate the entire process of setting up an object detection model to test the presence of certain elements on specific web pages. This process includes the creation of a dataset, and training and testing of the selected models. A choice of two models is currently available, namely YOLOv9 and RT-DETR. These models where selected for use in the application after making a weighted scoring model to compare and rank several popular and state-of-the-art object detection models. The application allows a user to generate an accurate object detection model in a few simple, streamlined steps, without the need for any machine learning knowledge. A positive user experience is supported by the many quality-of-life features and a user interface that conforms with modern design standards. Thus, the goal of developing a proof-of-concept for an application that facilitates the creation of object detection models to detect elements on web pages has certainly been reached.

The only question that remains is whether machine learning, and specifically object detection, is the right solution for testing the presence and display of web elements. The entire purpose of an object detection model is to generalize over its training set to be able to detect objects in unseen data. This contradicts somewhat with the requirement of detecting specific web elements. In a way, overfitting is necessary in this case since generalization is not the goal (to a certain extent, slight variations should still be detected). This has also been made clear by the fact that many data augmentation methods, used to prevent overfitting, cannot be applied in this case. Machine learning also makes it difficult to get consistent results. Some datasets might work a lot better or worse than others without a clear reason why. However, machine learning does provide something that can’t be done as well using other methods. Namely, it allows for the testing of the quality of an element being displayed. This can be seen as more of a non-functional test than purely testing the visibility, which is a functional test.

In conclusion, it is difficult to determine the usefulness of object detection in automated testing without BrightSight being used by testers in real projects. However, as mentioned object detection does have one major unprecedented capability in the current state of automated testing.

6 Reference list

[1] Selenium, „Getting started," [Online]. Available: https://www.selenium.dev/documentation/webdriver/getting_started/ [Accessed: 02/04/2024].

[2] Selenium, „Locators," [Online]. Available: https://www.selenium.dev/documentation/webdriver/elements/locators/ [Accessed: 02/04/2024].

[3] J. Chen, M. Xie, Z. Xing, C. Chen, X. Xu, L. Zhu en G. Li, „Object detection for graphical user interface: old fashioned or deep learning or a combination?," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.

[4] A. Kumar, K. Morabia, W. Wang, K. Chang en A. Schwing, „CoVA: Context-aware Visual Attention for Webpage Information Extraction," in *Proceedings of The Fifth Workshop on e-Commerce and NLP (ECNLP 5)*, 2022.

[5] T. Gogar, O. Hubacek en J. Sedivy, „Deep Neural Networks for Web Page Information Extraction," in *Artificial Intelligence Applications and Innovations*, Cham, 2016.

[6] M. Altinbas en T. Serif, „GUI Element Detection from Mobile UI Images Using YOLOv5," 2022, pp. 32-45.

[7] M. Xie, „UIED," [Online]. Available: https://github.com/MulongXie/UIED?tab=readme-ov-file [Accessed: 27/03/2024].

[8] T. Yeh, T.-H. Chang en R. C. Miller, „Sikuli: using GUI screenshots for search and automation," in *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, New York, NY, USA, 2009.

[9] J. C. Luna, „Top programming languages for data scientists in 2023," [Online]. Available: https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022 [Accessed: 02/04/2024].

[10] TIOBE, „TIOBE Index," [Online]. Available: https://www.tiobe.com/tiobe-index/ [Accessed: 02/04/2024].

[11] NiceGUI, „NiceGUI," [Online]. Available: https://nicegui.io/ [Accessed: 02/04/2024].

[12] M. Roseman, „TkDocs," [Online]. Available: https://tkdocs.com/index.html [Accessed: 02/04/2024].

[13] Google, „Welcome To Colab," [Online]. Available: https://colab.research.google.com/ [Accessed: 30/04/2024].

[14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár en C. L. Zitnick, „Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014*, Cham, 2014.

[15] COCO, „Detection evaluation," [Online]. Available: https://cocodataset.org/#detection-eval [Accessed: 13/03/2024].

[16] D. Shah, „Intersection over Union (IoU): Definition, Calculation, Code," [Online]. Available: https://www.v7labs.com/blog/intersection-over-union-guide [Accessed: 13/03/2024].

[17] D. Li, „Calculate Computational Efficiency of Deep Learning Models with FLOPs and MACs," [Online]. Available: https://www.kdnuggets.com/2023/06/calculate-computational-efficiency-deep-learning-models-flops-macs.html [Accessed: 13/03/2024].

[18] O. E. Olorunshola, A. K. Ademuwagun en C. Dyaji, „Comparative Study of Some Deep Learning Object Detection Algorithms: R-CNN, FAST RCNN, FASTER R-CNN, SSD, and YOLO," *Nile Journal of Engineering & Applied Science,* vol. 1, p. 70–80, September 2023.

[19] S. A. Sanchez, H. J. Romero en A. D. Morales, „A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework," *IOP Conference Series: Materials Science and Engineering,* vol. 844, p. 012024, May 2020.

[20] S. Srivastava, A. V. Divekar, C. Anilkumar, I. Naik, V. Kulkarni en V. Pattabiraman, „Comparative analysis of deep learning image detection algorithms," *Journal of Big Data,* vol. 8, p. 66, 2021.

[21] Z.-Q. Zhao, P. Zheng, S.-T. Xu en X. Wu, „Object Detection With Deep Learning: A Review," *IEEE Transactions on Neural Networks and Learning Systems,* vol. PP, pp. 1-21, January 2019.

[22] H. Zota en M. Dhande, „A Comparative Study of Widely Used Image Detection Algorithms," *International Research Journal of Engineering and Technology,* vol. 7, p. 5183–5187, July 2020.

[23] PyTorch, „Models and pre-trained weights: Object detection," [Online]. Available: https://pytorch.org/vision/stable/models.html#object-detection [Accessed: 13/03/2024].

[24] Kaggle, „Models," [Online]. Available: https://www.kaggle.com/models?tfhub-redirect=true&task=17074&fine-tunable=true [Accessed: 13/03/2024].

[25] G. Jocher en Laughing-q, „Models Supported by Ultralytics," 12 November 2023. [Online].

[26] PapersWithCode, „Object Detection on COCO test-dev," [Online]. Available: https://paperswithcode.com/sota/object-detection-on-coco [Accessed: 13/03/2024].

[27] H. Ouchra en A. Belangour, „Object Detection Approaches in Images: A Weighted Scoring Model based Comparative Study," *International Journal of Advanced Computer Science and Applications,* vol. 12, 2021.

[28] B. Zoph, E. D. Cubuk, G. Ghiasi, T.-Y. Lin, J. Shlens en Q. V. Le, „Learning Data Augmentation Strategies for Object Detection," *CoRR,* vol. abs/1906.11172, 2019.

[29] J. Nelson, „What is Image Preprocessing and Augmentation?," 26 January 2020. [Online]. Available: https://blog.roboflow.com/why-preprocess-augment/ [Accessed: 20/03/2024].

[30] Z. Zong, G. Song en Y. Liu, „co_dino_5scale_swin_large_16e_o365tococo.py," [Online]. Available: https://github.com/Sense-X/Co-DETR/blob/main/projects/configs/co_dino/co_dino_5scale_swin_large_16e_o365tococo.py [Accessed: 30/04/2024].

[31] Z. Zong, G. Song en Y. Liu, „co_dino_5scale_r50_1x_coco.py," [Online]. Available: https://github.com/Sense-X/Co-DETR/blob/main/projects/configs/co_dino/co_dino_5scale_r50_1x_coco.py [Accessed: 30/04/2024].

[32] Z. Zong, G. Song en Y. Liu, „Co-DINO Swin-DETR checkpoint files," [Online]. Available: https://drive.google.com/drive/folders/1r7bTh-DXkkQsCqkHYAHv4U-2hoEsLEoM?usp=drive_link [Accessed: 30/04/2024].

[33] Z. Zong, G. Song en Y. Liu, „Co-DETR," [Online]. Available: https://github.com/Sense-X/Co-DETR [Accessed: 13/03/2024].

[34] F. Schindler en R. Trappe, *NiceGUI: Web-based user interfaces with Python. The nice way.,* 2024.

[35] D. Cortesi en W. Caban, „PyInstaller Manual," [Online]. Available: https://pyinstaller.org/en/stable/ [Accessed: 30/04/2024].

[36] Z. Ge, S. Liu, F. Wang, Z. Li en J. Sun, „YOLOX," [Online]. Available: https://github.com/Megvii-BaseDetection/YOLOX [Accessed: 13/03/2024].

[37] C.-Y. Wang, A. Bochkovskiy en H.-Y. M. Liao, „yolov7," [Online]. Available: https://github.com/WongKinYiu/yolov7 [Accessed: 13/03/2024].

[38] G. Jocher, A. Chaurasia, F. C. Akyon en Laughing-q, „YOLOv8," 12 November 2023. [Online]. Available: https://docs.ultralytics.com/models/yolov8/ [Accessed: 13/03/2024].

[39] G. Jocher, B. Qaddoumi en Laughing-q, „YOLOv9: A Leap Forward in Object Detection Technology," 26 February 2024. [Online]. Available: https://docs.ultralytics.com/models/yolov9/ [Accessed: 13/03/2024].

[40] G. Jocher, „Baidu's RT-DETR: A Vision Transformer-Based Real-Time Object Detector," 12 November 2023. [Online]. Available: https://docs.ultralytics.com/models/rtdetr/ [Accessed: 13/03/2024].

[41] PyTorch, „Faster R-CNN," [Online]. Available: https://pytorch.org/vision/main/models/faster_rcnn.html [Accessed: 13/03/2024].

[42] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo en R. Girshick, *Detectron2,* 2019.

[43] J. Dai, Y. Li, Y. Xiong en H. Qi, „R-FCN," [Online]. Available: https://github.com/daijifeng001/R-FCN [Accessed: 13/03/2024].

[44] PyTorch, „SSD," [Online]. Available: https://pytorch.org/vision/main/models/ssd.html [Accessed: 13/03/2024].

[45] PyTorch, „FCOS," [Online]. Available: https://pytorch.org/vision/main/models/fcos.html [Accessed: 13/03/2024].

[46] PyTorch, „RetinaNet," [Online]. Available: https://pytorch.org/vision/main/models/retinanet.html [Accessed: 13/03/2024].

[47] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li en others, „InternImage," [Online]. Available: https://github.com/opengvlab/internimage [Accessed: 13/03/2024].

[48] PapersWithCode, „Torchvision," [Online]. Available: https://paperswithcode.com/lib/torchvision [Accessed: 13/03/2024].

[49] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu en A. C. Berg, „SSD: Single Shot MultiBox Detector," in *Computer Vision – ECCV 2016*, Cham, 2016.

[50] PapersWithCode, „NAS-FCOS: Fast Neural Architecture Search for Object Detection," [Online]. Available: https://paperswithcode.com/paper/nas-fcos-fast-neural-architecture-search-for [Accessed: 13/03/2024].

7 Attachment

7.1 Comparison of object detection models

Table 2: Comparison of object detection model accuracy

Model		Accuracy		
Name	**Variant**	**Score**	**mAP (%)**	**Dataset**
YOLOX	s	42.30	40.50	COCO
	m	57.95	46.90	COCO
	l	64.79	49.70	COCO
	x	68.22	51.10	COCO
YOLOv7		68.46	51.20	COCO val
	X	72.62	52.90	COCO val
	W6	76.77	54.60	COCO val
	E6	79.95	55.90	COCO val
	D6	80.93	56.30	COCO val
	E6E	82.15	56.80	COCO val
YOLOv8	n	34.47	37.30	COCO val
	s	53.06	44.90	COCO val
	m	66.01	50.20	COCO val
	l	72.62	52.90	COCO val
	x	75.06	53.90	COCO val
YOLOv9	S	57.70	46.80	COCO val
	M	68.95	51.40	COCO val
	C	72.86	53.00	COCO val
	E	**79.22**	**55.60**	**COCO val**
RT-DETR	L	72.86	53.00	COCO val
	X	**77.26**	**54.80**	**COCO val**
Co-DETR	**Co-DINO**	**100.00**	**64.10**	**COCO val**
	Co-Deformable-DETR	86.31	58.50	COCO val
Faster R-CNN	R50-FPN	33.74	37.00	COCO val
	R50-FPN	41.56	40.20	COCO val
	R101-FPN	45.97	42.00	COCO val
	X101-FPN	48.41	43.00	COCO val
R-FCN		20.29	31.50	COCO val
SSD	300 VGG16	0.00	23.20	COCO test-dev
FCOS	R50-FPN	40.59	39.80	COCO test-dev
RetinaNet	R50-FPN	32.27	36.40	COCO minival
	R50	37.90	38.70	COCO val
	R101	42.05	40.40	COCO val
InternImage	T	63.33	49.10	COCO
	S	64.79	49.70	COCO
	B	66.26	50.30	COCO
	L	80.44	56.10	COCO
	XL	80.68	56.20	COCO
Weight		**0.60**		

Table 3: Comparison of object detection model speed

Model		Speed		
Name	**Variant**	**Score**	**FLOPs (B)**	**FPS**
YOLOX	s	98.98	26.80	102.04
	m	96.33	73.80	81.30
	l	91.72	155.60	68.97
	x	84.59	281.90	57.80
YOLOv7		94.59	104.70	161.00
	X	89.78	189.90	114.00
	W6	80.19	360.00	84.00
	E6	71.44	515.20	56.00
	D6	54.99	806.80	44.00
	E6E	52.94	843.20	36.00
YOLOv8	n	100.00	8.70	
	s	98.88	28.60	
	m	96.06	78.60	
	l	91.17	165.20	
	x	85.95	257.80	
YOLOv9	S	98.98	26.70	
	M	96.16	76.80	
	C	94.69	102.80	
	E	**89.64**	**192.50**	
RT-DETR	L	94.29	110.00	
	X	**87.29**	**234.00**	
Co-DETR	**Co-DINO**	**84.76**	**279.00**	
	Co-Deformable-DETR	51.99	860.00	
Faster R-CNN	R50-FPN	75.28	447.00	10.20
	R50-FPN	13.17		26.32
	R101-FPN	8.85		19.61
	X101-FPN	2.79		10.20
R-FCN		0.00		5.88
SSD	300 VGG16	34.24		59.00
FCOS	R50-FPN	89.80	189.60	
RetinaNet	R50-FPN	70.77	527.00	24.39
	R50	11.93		24.39
	R101	11.93		24.39
InternImage	T	85.26	270.00	
	S	81.32	340.00	
	B	72.24	501.00	
	L	21.60	1399.00	
	XL	0.00	1782.00	
Weight		**0.30**		

Table 4: Comparison of object detection model availability

Model		Availability		
Name	**Variant**	**Score**	**How**	**Where**

YOLOX	s	0.00	download	[36]
	m	0.00	download	
	l	0.00	download	
	x	0.00	download	
YOLOv7		0.00	download	[37]
	X	0.00	download	
	W6	0.00	download	
	E6	0.00	download	
	D6	0.00	download	
	E6E	0.00	download	
YOLOv8	n	100.00	lib, no data loading	[38]
	s	100.00	lib, no data loading	
	m	100.00	lib, no data loading	
	l	100.00	lib, no data loading	
	x	100.00	lib, no data loading	
YOLOv9	S	100.00	lib, no data loading	[39]
	M	100.00	lib, no data loading	
	C	100.00	lib, no data loading	
	E	**100.00**	**lib, no data loading**	
RT-DETR	L	100.00	lib, no data loading	[40]
	X	**100.00**	**lib, no data loading**	
Co-DETR	**Co-DINO**	**0.00**	**download**	[33]
	Co-Deformable-DETR	0.00	download	
Faster R-CNN	R50-FPN	50.00	lib	[41]
	R50-FPN	50.00	lib	[42]
	R101-FPN	50.00	lib	
	X101-FPN	50.00	lib	
R-FCN		0.00	download	[43]
SSD	300 VGG16	50.00	lib	[44]
FCOS	R50-FPN	50.00	lib	[45]
RetinaNet	R50-FPN	50.00	lib	[46]
	R50	50.00	lib	[42]
	R101	50.00	lib	
InternImage	T	0.00	download	[47]
	S	0.00	download	
	B	0.00	download	
	L	0.00	download	
	XL	0.00	download	
Weight		**0.10**		

Table 5: Comparison of object detection model scores

Model		Scoring		Data source
Name	**Variant**	**Weighted score**	**Rank**	
YOLOX	s	55.07	27	[36]
	m	63.67	21	
	l	66.39	17	

	x	66.31	18	
YOLOv7		69.45	14	[37]
	X	70.50	12	
	W6	70.12	13	
	E6	69.40	15	
	D6	65.06	20	
	E6E	65.17	19	
YOLOv8	n	60.68	25	[38]
	s	71.50	11	
	m	78.43	9	
	l	80.92	6	
	x	80.82	7	
YOLOv9	S	74.32	10	[39]
	M	80.22	8	
	C	82.12	4	
	E	**84.42**	**2**	
RT-DETR	L	82.00	5	[40]
	X	**82.55**	**3**	
Co-DETR	**Co-DINO**	**85.43**	**1**	[33]
	Co-Deformable-DETR	67.38	16	
Faster R-CNN	R50-FPN	47.83	30	[48]
	R50-FPN	33.89	34	[42]
	R101-FPN	35.23	32	
	X101-FPN	34.88	33	
R-FCN		12.18	38	[43]
SSD	300 VGG16	15.27	37	[49]
FCOS	R50-FPN	56.29	26	[50]
RetinaNet	R50-FPN	45.60	31	[48]
	R50	31.32	36	[42]
	R101	33.81	35	
InternImage	T	63.57	22	[47]
	S	63.27	23	
	B	61.43	24	
	L	54.74	28	
	XL	48.41	29	
Weight				