Development of an Artificial Intelligence-Aided Software for Annotating Image Datasets

Paulo Victor de Magalhães Rozatto
Universidade Federal de Juiz de Fora
Juiz de Fora, Brazil
paulo.rozatto@estudante.ufjf.br

Luiz Maurílio da Silva Maciel
Universidade Federal de Juiz de Fora
Juiz de Fora, Brazil
luiz.maciel@ufjf.br

Marcelo Bernardes Vieira
Universidade Federal de Juiz de Fora
Juiz de Fora, Brazil
marcelo.bernardes@ufjf.br

Saulo Moraes Villela
Universidade Federal de Juiz de Fora
Juiz de Fora, Brazil
saulo.moraes@ufjf.br

Abstract

Context: Deep learning is a highly successful class of methods in the field of artificial intelligence (AI) that has a variety of applications. To perform well, deep learning models require a large amount of high-quality annotated data. **Problem:** Data annotation is a time-consuming and laborious task that requires a significant amount of human labor, which makes it expensive. **Solution:** This work aims to reduce the time required to annotate image datasets by building an easy-to-use software tool that has semi-automated annotation powered by an artificial intelligence model. **IS Theory:** The work is based on the Socio-technical theory because we developed and evaluated a tool to be acceptable and useful for the users. **Method:** We developed a web-based tool and employed HQ-SAM, a deep neural network for image segmentation based on Vision Transformers, to generate polygon annotations based on the user's prompts. Although HQ-SAM has a good zero-shot generalizability, we fine-tuned it on the Bean Leaf Dataset to evaluate how well the network adapts to specific tasks. **Summary of results:** We observed an increase in accuracy of the fine-tuned model compared to the pre-trained one. We tested our tool with 20 participants, all of whom are from the computer vision and graphics fields. We asked them to annotate the same two images both manually and AI-aided, and recorded the annotation times. Lastly, we asked the participants to fill out a usability form about their user experience. In our evaluation, we registered a median speedup of 1.5× regarding the AI-aided annotation compared to manual annotation and overly positive answers regarding our tool's ease of use and usefulness. **Contribution:** We expect the proposed system to significantly reduce the human effort required for image dataset annotation, leading to faster annotation times. This offers a valuable contribution to the computer vision and AI communities, speeding up the dataset creation process.

CCS Concepts

• **Computing methodologies** → **Computer vision**; • **Information systems** → **Information systems applications**.

Keywords

Deep learning, Web application, Semi-automated annotation, Image datasets, Vision Transformers

1 Introduction

Deep learning is a class of methods that can learn from raw data the representations required to complete a certain task. They are made up of several simple modules called neurons, each of which slightly transforms the input in a non-linear way. Those neurons are arranged into layers. By stacking multiple layers and interconnecting them, it is possible to learn a wide range of complicated functions that model the data fed in [15].

For more than a decade now, deep learning has been the state of the art of many artificial intelligence applications, including speech recognition, scientific data processing (such as those from particle accelerators), predicting the behavior of potential therapeutic drug molecules, and many others [15].

According to LeCun et al. [15], the most common method for fitting a deep learning model is through a technique known as supervised learning. It works by first collecting a large amount of data, annotating it, and then executing the so-called training algorithm. For example, given a set of pictures of cats and dogs, each with a label that correctly identifies the image class, a deep learning model can iteratively pass through the dataset, attempt to guess which class of animal is present in each picture, compare its predictions to the correct labels, and then self-correct itself from its mistakes.

Having high-quality annotations is essential for a supervised learning. Aside from labels that indicate which class an image belongs to, there are different forms of annotation for different purposes. As an instance, object detection in a computer vision context refers to locating objects in an image such as faces, cars, pedestrians, and so on [31]. Annotations for that type of problem, in addition to the object class, require the object's coordinates in the image, such as its bounding box.

Another classic problem is image segmentation. There are different kinds of segmentation, but overall, the main goal is to create multiple meaningful partitions of an image [18]. Segmentation annotations are frequently more complex than those for other tasks, since the problem requires pixel-wise class assignment.

Despite its effectiveness, deep learning presents numerous challenges, including being intrinsically a blackbox — i.e., how the model makes decisions is hardly explainable in human language — and data-related issues, such as its scarcity for specific tasks and overfitting [23]. Overfitting occurs when a model performs well on training data but not well on unseen data, implying that it does not

generalize its learning. Among other factors, overfitting is induced by using a dataset with an insufficient amount of elements, so noise and outliers cause significant influence on the training results [29].

One of the primary causes of data scarcity is the cost of annotating a dataset. Consider the case of MS COCO, one of the most popular datasets in computer vision for object detection and segmentation. Over 70,000 hours of human labor were required to provide approximately 160,000 annotated images, which represented roughly fifty percent of the initial plan [17].

Annotating datasets is expensive mostly because it requires many hours of human labor. Thus, introducing automation, such as an artificial intelligence agent, which can carry out at least some part of the work for itself, could significantly reduce dataset creation costs.

This work proposes an annotation software tool that integrates with such artificial intelligence agent. We follow a socio-technical approach, aiming to optimize human-machine collaboration and addressing the fourth challenge by SBC for information systems from 2016 to 2026 [3]. The tool uses a pre-trained deep learning model to assist annotation, reducing the human annotator job to reviewing and validating the agent's forecast. Rather than replacing users, the AI functions as a supportive agent, enhancing efficiency and diminishing human workload. We evaluated the contribution of our proposed software in terms of annotation speedup and ease of use with 20 volunteers from the main target user group, which is people who conduct research in computer vision and computer graphics.

2 Theoretical Foundation

This section outlines the theoretical principles that underlie this work. Section 2.1 discusses the integration of human and technological systems using the socio-technical theory. Section 2.2 introduces the core concepts of Vision Transformers, which serve as the foundation for the HQ-SAM [13], a promptable segmentation model used in this work as an AI agent, which is described in Section 2.3.

2.1 Socio-Technical Theory

According to Abbas and Katina [1], "Socio-technical theory is an organisational theory that conceptualises a given work or other system in view of its constituent social and technical subsystems, with the goal of achieving system success through joint optimisation." It emphasizes integrating human factors, such as user needs and communication, with technical elements like tools and processes to create effective, adaptable, and sustainable systems.

Applying the socio-technical theory to AI-driven systems means ensuring that automation is designed not only for technical efficiency but also in consideration of how humans will interact with the technology. Because technology and artificial intelligence are inseparably linked to social interactions, failing to give technical and social issues equal weight may result in a class struggle [24].

The present work is inspired by socio-technical theory, and it approaches reducing annotation time as a joint optimization problem. The main technical subsystem is the AI model described in Section 2.3, a promptable segmentation model with high generalizability capacity. Prompts are the core communication channel between technical and social subsystems. The social subsystem is represented by human annotators, whose role is to define the prompts for the AI, refine and validate the AI-generated annotations. As a facilitator for the annotation process, we developed a system with a human-centered design in mind. The software has a straightforward user interface, and it handles the subsystems communication, including prompts and converting AI's output to an annotation format more easily editable to humans. The expected joint outcomes are annotations done faster and more accurately than the ones done by non-collaborative workflows (i.e., humans or AI alone).

2.2 Vision Transformers

Vaswani et al. [28] proposed the Transformer architecture for machine translation. The novelty was that this model only used the so-called attention mechanism, whereas most previous state of the art approaches used techniques such as recurrency and convolution. The Transformers launched a new era of high-quality models in natural language processing. As a result, new efforts began to utilize them in other fields as well.

Dosovitskiy et al. [8] introduced the Vision Transformer (ViT) for image classification. The authors demonstrated that the ViT outperformed the state of the art at the time on large datasets. The ViT has since become part of a variety of computer vision models, including the one employed in this study. Thus, we present ViT here with further details.

As Figure 1 illustrates, how the Vision Transformer works can be divided into six main steps. First, the image input is split in a sequence of fixed-size 2D patches, because Transformers work with sequential data. Then, the patches are flattened and mapped to D dimensions through a learnable linear projection. In addition, an extra learnable embedding, which later is going to be used as a classification token, is prepend to the sequence of patches embeddings. In order to retain positional information, a learnable positional encoding is applied. Next, the Transformer encoder takes the embeddings as input, computes the attention between them, updating each embedding based on all the others. Finally, the classification is performed by a multi-layer perceptron (MLP) fed with the final state of the extra embedding, which is expected to serve as the image representation.

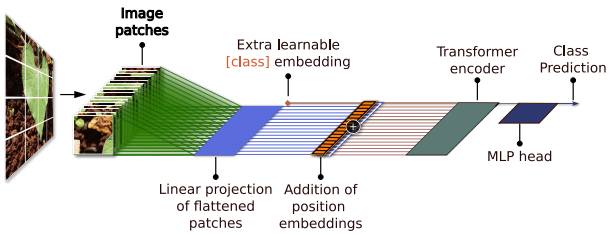


Figure 1: Vision Transformer. Adapted from Dosovitskiy et al. [8].

Figure 2 depicts the internal structure of the Transformer encoder. Essentially, it is made up of L blocks of normalization, multi-head attention, residual connection, a second normalization, MLP, and a final residual connection. A residual connection is a connection that bypasses one or more layers [12].

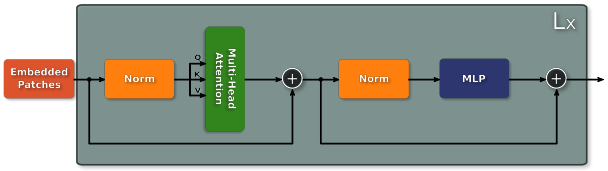


Figure 2: Transformer encoder. Adapted from Dosovitskiy et al. [8].

The attention layer has three inputs, which are denoted as queries, keys, and values — in ViT, all the three are distinct linear projections of the same data, but queries source could be different than the values and keys sources in some other cases. The model compares queries and keys for a given set of inputs to determine how much each value should contribute to the final output. Let Q, K, and V be matrices containing, respectively, queries, keys, and values. The matrices Q and K share the same dimensionality d_k, while V has dimensionality d_v. The similarity of Q and K is computed through the dot product and, for numerical stability reasons, it is divided by $\sqrt{d_k}$. Then, a softmax function is applied to obtain a probability distribution. Finally, the result is multiplied by V to produce the attention. Equation (1) summarizes these processes.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \quad (1)$$

Also, the queries, keys and values can be linearly projected onto h different heads. According to Vaswani et al. [28], "Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions." In the end, the outputs from the heads are concatenated and linearly projected in order to get the final values.

2.3 Segment Anything in High Quality

The HQ-SAM [13] is a highly accurate segmentation model with zero-shot generalizability, which is used in the present work to semi-automate annotation. It is based on the Segment Anything Model (SAM) [14], a ViT-based architecture trained on the largest segmentation dataset ever created at publishing time, consisting of around 1.1 billion masks and 11 million images. As most of SAM's training dataset was generated automatically, it struggles with some fine-grained segmentation tasks. Furthermore, due to the size of the model and dataset, direct fine-tuning is difficult to accomplish without compromising generalization performance.

HQ-SAM tackles the issue by reusing the SAM's weights wherever feasible, adding a few new structures, and curating some existing datasets to create a new one with fine-grained masks. HQ-SAM keeps the encoder, the heavier component of SAM, untouched. The encoder is made of ViTs, and its main role is to take an image input, extract useful features, and represent it in a lower dimension. The encoder output is called image embedding.

The image embedding, alongside the user prompts, is then fed into the HQ-SAM's second component, known as the decoder, which generates segmentation masks. The user prompts define the region in the image to be segmented and they might be a set of points, a bounding box, or a coarse segmentation mask. Those prompts are tokenized, i.e., they are all represented by vectors with the same dimensions, and learnable output tokens are prepended to the embeddings collection.

Figure 3 represents the HQ-SAM decoder. The blue area in the top half corresponds to the original SAM's decoder and it is frozen during training time. The red area in the bottom half corresponds to the new elements added by HQ-SAM and only they are trainable.

Focusing first on original SAM, we can see that the model computes a series of attention values: self-attention between output and prompt tokens, then tokens attend to the image embedding, and lastly the image embedding attend to the tokens. That block of multiple attentions is repeated twice. There is a final attention between token and image embeddings, then the model predicts the segmentation mask through a point-wise product between the updated image embedding upscaled through two transposed convolutions and the updated output token upscaled through a MLP.

Looking at the HQ-SAM part now, we see that the result of the transposed convolutions are summed up with the results of an early and final layer of the encoder, then the model predicts the segmentation with a point-wise product with the new HQ-Output token upscaled by a MLP. In addition, during inference time, the original SAM and the HQ-SAM predictions can be combined for error correction.

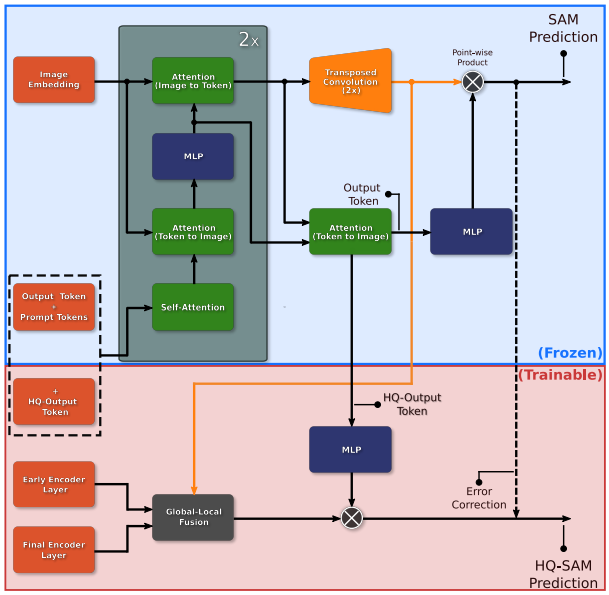


Figure 3: HQ-SAM decoder. Adapted from Ke et al. [13].

3 Related Work

Over the years, various dataset annotation softwares have been developed. Some were designed for specific cases, while others had a broader reach. For example, the MS Coco project created its own annotation tool, where users painted over objects to create segmentation masks [17]. On the other hand, LabelMe, previously one of the most influential annotation softwares, had a wider purpose. It was an open-source project and an online platform where individuals collaborated to construct an open dataset using polygon

annotations [27]. However, a limitation that both LabelMe and MS Coco share is that they rely entirely on human labor. Subsequent works on dataset annotation employed a variety of computer vision techniques to add automation into this task.

Boonsri and Limpiyakorn [4] applied traditional computer vision techniques to detect and annotate the coordinates of cannabis seeds in an image. The authors intended to help Thai growers distinguish between female and male cannabis plants before planting them because only female cannabis is commercially viable. The authors first needed to build a dataset, so they proposed a software that takes as input an image containing a handful of seeds distributed on a white background, converts it to gray-scale, then smooths it with Gaussian blur. The image is then passed through a threshold, and contours are calculated to find the bounding boxes for the seeds. Their tool writes out the results in JSON files, and they still need to perform the male/female classification.

Qin et al. [21] used edge detection to create a general-purpose annotation software called ByLabel. Their tool uses an algorithm called Edge Drawing [26] to obtain edges segments all across the image. These segments are partitioned into smaller fragments either manually or automatically — automatic partition is based on turning angle heuristics. The user then chooses the edge fragments which will compound the final annotation. The expectation is that edge selection will be less costly than traditional annotation methods. The authors led an experiment with 10 volunteers with no experience in annotation. According to their evaluation, using ByLabel reduces the annotatiom time by a factor of 56% compared to using the traditional LabelMe.

Entering in the realm of semi-automation powered by artificial intelligence, Yu et al. [30] proposed the use of unsupervised learning algorithms to simplify annotation. The authors work with a specialized dataset containing images of rock thin sections utilized in geological and mineralogical studies. Their tool uses a clustering algorithm to split the image into several non-overlapping groups of similar pixels called superpixels. The human annotation is then reduced to labeling the superpixels of interest. The authors report a speedup of 5× to 8× when comparing experts using their tool versus generating segmentation masks with QGis[1] or Photoshop.

Supervised learning also plays an important role in dataset annotation, and is applied in various approaches. Fîciu et al. [10] aimed to automate the annotation process completely. The authors employed an architecture called Mask R-CNN [11] to predict bounding boxes and segmentation masks. The bounding box is utilized as input, alongside the image, to help Polygon-RNN [5] predict an annotation polygon. The results are exported in JSON format. One limitation of this method is that it cannot generalize to previously unseen classes. In fact, the authors report results for only four categories: people, cars, trucks, and bicycles. The addition of new classes would require retraining the Mask R-CNN.

Most annotation tools allow human users to fix the neural network's predictions. For example, Philbrick et al. [19] developed a software to annotate medical images. Their system supports plugins that enable the deployment and use of different neural networks to predict segmentation masks, as long as the neural network is implemented in Keras[2] running on TensorFlow.[3] The user can change the predictions using drawing tools, including painting, erasing, and filling. Also, annotated images can be used to fine-tune models within the software. According to the authors, their software design is limited to the annotation of images stored in the Neuroimaging Informatics Technology Initiative (NIfTI) file format.

Other strategies attempt to use annotators' knowledge to improve the model's predictions. For example, in the case of object detection, Pugdeethosapol et al. [20] proposed asking users to click on an object and then use the click location to select among bounding boxes generated by a pretrained backbone. Moreover, the user can correct the predictions, and the corrected data is used to train a second network incrementally during runtime. Over time, the second model is expected to make better predictions and require fewer user adjustments during annotation.

There are interactive approaches for image segmentation as well. Sambaturu et al. [22] developed a framework for annotating urban city scenes in which a pretrained network gives mask predictions, and the user can make adjustments by scribbling where the network made mistakes. These corrections are utilized to determine local loss at scribbling pixel coordinates and to backpropagate during inference. Based on an evaluation conducted with two experts, the authors claim a time savings of up to 14.7× compared to the complete human annotation.

Differently from previously cited works, we propose to use segmentation prompts to reduce annotation time. Prompts are now widely used in AI text generators like ChatGPT.[4] However, unlike ChatGPT, our prompts are the ones supported by the SAM/HQ-SAM model rather than text prompts. Given that HQ-SAM has zero-shot capabilities, our tool can be generalized for a wide range of classes without additional training, while also being possible to fine-tune the model to perform better in specific classes. In addition, we propose conducting a more comprehensive quantitative and qualitative evaluation of the proposed tool compared to other studies. We proposed evaluating our tool with both experienced and inexperienced participants, aiming for a significant sample size based on existing literature.

4 Proposed System

The proposed system for this work is a single-page web application with a desktop-like workflow. Our application is backed up by a server that serves the content, manages the data, and handles the interaction AI model. Thus, we use a client-server architecture, as shown in Figure 4.

In the following subsections, we will discuss the software with further details. In Section 4.1, we present the main languages, libraries, and tools used in the application. In Section 4.2, we describe the fine-tuning process we performed in the HQ-SAM network. We discuss the server-side implementation in Section 4.3. Lastly, in Section 4.4, we present the client-side and the main features available to users through the user interface (UI).

[1] https://www.qgis.org/

[2] https://keras.io/

[3] https://www.tensorflow.org/

[4] https://chatgpt.com/

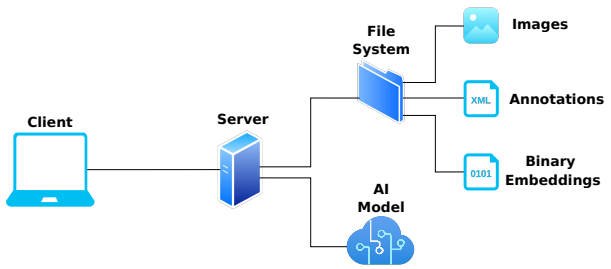


Figure 4: Overall system architecture.

4.1 Languages, Libraries, and Tools

The web client was developed with HTML, SASS[5] (a CSS extension language), and plain JavaScript. We also used Parcel,[6] a JavaScript build tool that enabled us to use NPM[7] for dependency management, as well as supplying source-mapping and code minification out-of-the-box.

The server was implemented in Java 17, and uses the Spring Boot framework to create API endpoints, handle HTTP requests, and serve static files. We chose Gradle as the build tool and dependency manager for the back-end. We used the OpenCV[8] library for image processing and to perform a few computer vision tasks.

The HQ-SAM was written in Python, and we used existing Python scripts to fine-tune the network. To integrate HQ-SAM into the system, we exported the trained weights to an ONNX[9] file, which can be run directly by the server powered by the ONNX Runtime for Java.

4.2 Fine-tuning the HQ-SAM Network

Although our ultimate goal is to produce a general-purpose annotation tool, the software development was initially focused on meeting the requirements for annotating the Bean Leaf Dataset [7]. This set consists of images containing a prominent bean leaf and an augmented reality marker (Figure 9). We fine-tuned the model to improve HQ-SAM accuracy in that dataset and developed a feature in our system that supports multiple HQ-SAM checkpoints. During the fine-tuning process, we froze the original SAM layers and trained only the structures introduced by HQ-SAM.

Only leaves were used in the fine-tuning because the marker does not need to be recognized through a neural network, as it is simpler and can already be recognized by augmented reality libraries. The available dataset contained a total of 3756 images organized into 300 folders, each folder containing different pictures from the same leaf. We randomly selected 20% of the folders (60 in total) to provide the images of our test set, while the remaining 80% provided the training set. The original images had a resolution of 3468×4624. They were cropped to 1024×1024, the HQ-SAM input resolution, keeping the leaf in the center.

We kept almost the same parameters used by HQ-SAM:

- Optimizer: Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, $\lambda = 0$);
- Initial learning rate: 10^{-3};
- Learning rate decay factor: 10^{-1} every 10 epochs;
- Number of epochs: 12;
- Seed: 42.

The only difference is that we reduced the training batch size from 4 to 2 due to VRAM limitations. We performed the fine-tuning on a Quadro M5000 GPU with 8GB VRAM, which took a total of 11.7 hours to complete. The results will be presented in Section 5.1.

4.3 Server

The server code is self-contained; it compiles into a JAR that includes all required libraries, ONNX files, and web files. The application creates a directory for itself in the user's home folder, a common place to store application data. The application's directory contains three automatically created subdirectories:

(1) *datasets*: the directory where the server will look for the images and annotations data. The content in this directory is statically served, and its subdirectory structure is exposed through an endpoint so the client can implement directory navigation;
(2) *checkpoints*: the directory where the server will look for user's fine-tuned checkpoints of the HQ-SAM decoder. They are recognized as alternatives to the default checkpoint included in the JAR file;
(3) *embeddings*: the directory where the server will look for image embeddings before running HQ-encoder. Whereas the HQ-SAM decoder is lightweight, the encoder is heavy. As the encoder's output for an image does not change no matter the prompt, we save it to the disk for speeding up later usages of the same image.

When handling prediction requests, the server responds with a list of coordinates, which represents the segmentation polygon of an object. However, the output of HQ-SAM is a binary segmentation mask. The conversion from a binary image to a sequence of points is performed through OpenCV, first by using the Suzuki et al. [25] algorithm to find the mask contour and then by using the Douglas and Peucker [9] algorithm to obtain the corners of an approximated polygon.

The server only stores annotations on disk at the client's request. In the save request, the client can specify where will be the location of the annotation within the datasets directory, as well as the file name and format, as long as it is in text format. This allows for support for multiple dataset structures and annotation formats as long as the client knows how to read them.

In addition, the server serves the default client on the user's localhost port 8080. As a result, a single JAR file can be provided as a complete software package, and it does not require installation.

4.4 Client

The UI of the system displays the client's main features, as shown in Figure 5. The interface is mostly in Portuguese, as it is initially intended for Brazilian users. The screen is divided into four sections: a bar with the main features at the top, a bar with secondary information at the bottom, a file list on the right side, and a canvas that displays the content in the center.

[5]https://sass-lang.com/
[6]https://parceljs.org/
[7]https://www.npmjs.com/
[8]https://opencv.org/
[9]https://onnx.ai/

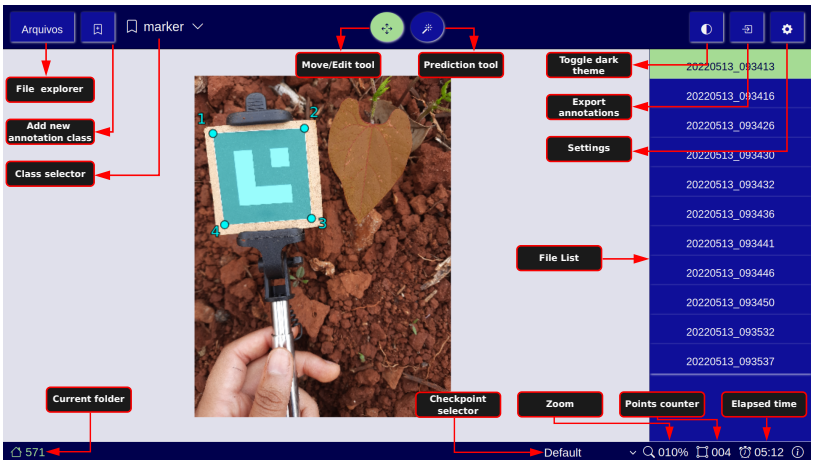


Figure 5: User interface main elements.

In the center of the top bar, we placed the buttons that should be used more frequently, which are the tools to edit annotations manually and to use the AI model. We assume that the file explorer, class selector, and class creation are the second most commonly used buttons, thus we positioned them on the left. The right-side buttons are likely to be used less frequently. Respectively, they toggle the dark theme, export annotations, and open the settings modal.

We employed modals to keep the client as a single-page application while having a few additional and straightforward interfaces. The file explorer modal displays the browseable list of folders in the server's *datasets* folder. The class creation modal contains only some fields to choose the class name, set an optional limit of points, pick a color, and choose whether the points should be enumerated in the UI. The settings modal only include three sliders for controlling the opacity of the polygons, the maximum magnification allowed, and the scrolling speeding.

When a folder is selected, the client retrieves a list of image files and displays their names in the right bar, in alphabetical order. The first image of the list and its corresponding annotations (if they exist) are rendered on canvas. The user can switch to the other images by clicking on their names on the list.

Once an image has been rendered on the canvas, the user can manually add annotations by clicking on the screen with the left mouse button while using the editing tool. Each click generates a point, resulting in a polygon. Users can also edit annotations by clicking and dragging points to different locations, or delete them by clicking on them and typing DELETE or BACKSPACE on the keyboard.

The AI tool provides users with two prompt alternatives. The first option is to use point prompts to indicate foreground or background. The foreground prompts can be added with a left click and are represented by green dots, while the background prompts need CTRL + left click and are represented by red dots. The second option lets users draw a box around the object they want to segment. In both cases, users should press the ENTER key after completing the prompts. These prompts serve as input alongside the image to the HQ-SAM, which then computes a segmentation mask based on them.

Regardless of the tool used, users can move the image on the canvas by clicking and dragging with the mouse's right button, and zoom in or out using the mouse scroll. Furthermore, when the mouse cursor hovers over a selectable object (polygons or points), the object is highlighted with an outline to provide visual feedback.

The bottom bar displays secondary but useful information such as the name of the currently open directory, the number of points of the selected annotation, the amount of zoom currently applied to the rendered image, and a timer that begins when the user opens a folder. Additionally, users can see which checkpoint is currently being used by HQ-SAM in the back-end, and they can click on it to select another one if there are other options stored in the checkpoints folder in the server.

Lastly, users can save their work to the server using the conventional CTRL + S shortcut. The annotations are normalized by the image's height and saved in an XML file. Users can also export and save locally the annotation files by clicking the export option in the top bar. The download is a zip file including one folder with the XMLs and one folder with the annotations in a JSON file following the MS COCO format.

5 Experiments and Results

This section presents the methodology and findings of our research and is divided in two subsections. Section 5.1 evaluates our main technical subsystem by assessing the segmentation model's performance and fine-tuning capabilities. Section 5.2 describes our software evaluation, which comprises a social evaluation via a usability research to examine the software's usability, as well as an evaluation of the joint outcome in the form of the performance of our AI-aided annotation approach compared to manual annotation.

5.1 Fine-tuning HQ-SAM Network

To evaluate the effect of the fine-tuning for the Bean Leaf Dataset, we used the intersection over union (IoU) and the Boundary IoU [6]. These metrics are the same as those used by the HQ-SAM authors. In fact, we calculated them using their implementation.

The IoU is obtained using Equation (2). We divide the area of the intersection between the ground truth mask (G) and the prediction mask (P) by the area of their union. The Boundary IoU in Equation (3) is close to the IoU, but it focuses on the edges. As Cheng et al. [6] define, "G_d and P_d are the sets of all pixels within d pixels distance from the ground truth and prediction contours respectively." The parameter d used by HQ-SAM is 2% of the image diagonal and we kept it unchanged.

$$\text{IoU} = \frac{|G \cap P|}{|G \cup P|}, \tag{2}$$

$$\text{Boundary IoU} = \frac{|(G_d \cap G) \cap (P_d \cap P)|}{|(G_d \cap G) \cup (P_d \cap P)|}. \tag{3}$$

We evaluated both the predictions using box prompts and point prompts on the test set, keeping the same patterns. For the box prompts, we used the leaves' bounding boxes. For the point prompts, we used only the center of the bounding box. We chose using a single point prompt both for simplicity and because it represents the minimal effort annotation method in our tool.

The graphs in Figure 6 summarize the results for the box prompts, whereas Figure 7 shows the results for the point prompts. In both figures, when comparing pre-trained weights to fine-tuned ones, we can observe that the mean increased while the spread decreased for both metrics. For the box prompts the mean IoU went up from 98.6% ± 0.7% to 99.1% ± 0.3% and the mean Boundary IoU went up from 90.9%±3.7% to 93.3%±1.8%. Meanwhile, for the point prompts, the mean IoU increased from 86.6% ± 19.0% to 97.7% ± 2.4% and the mean Boundary IoU increased from 66.3% ± 25.5% to 80.3% ± 15.4%.

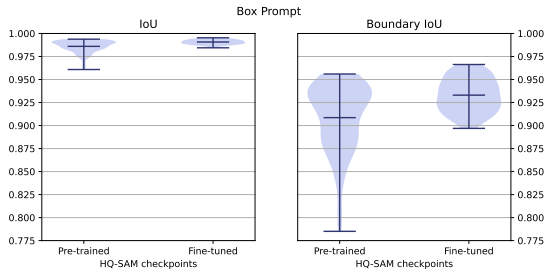


Figure 6: Comparison of IoU and Boundary IoU values for the pre-trained and the fine-tuned checkpoints using box prompts.

Figure 8 shows a qualitative example of the fine-tuning effect. As shown in Figure 8a and Figure 8c, the model without fine-tuning incorrectly assigns some green background elements to the leaf. The Figure 8c contains a specially large background area that was mistakenly included in the segmentation, suggesting that single-point prompts tend to be more ambiguous than box prompts. The fine-tuned model on the same images for both prompt types predicts the segmentation masks more accurately, following the actual leaf contour, as shown in Figure 8b and Figure 8d.

In summary, we conclude that the fine-tuning was successful. Box prompts carry more information than single-point prompts, but the fine-tuning method considerably reduced the difference between them, making both of them viable options. Furthermore,

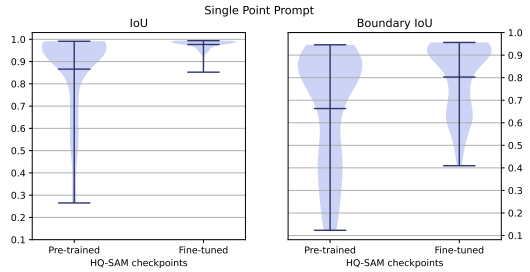
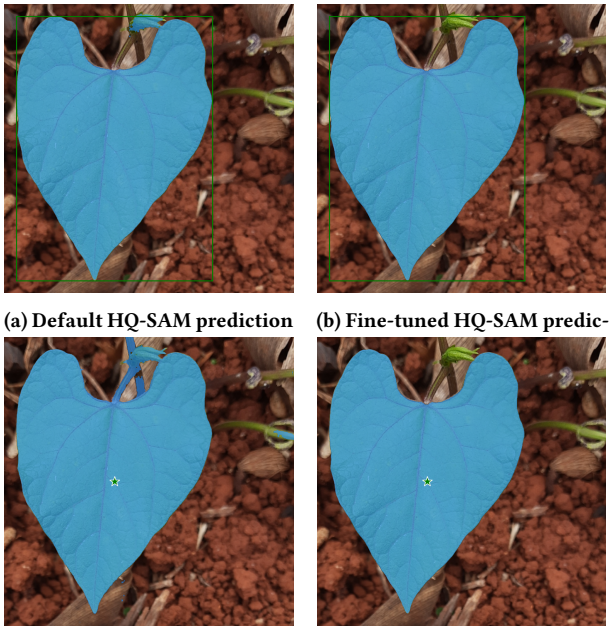


Figure 7: Comparison of IoU and Boundary IoU values for the pre-trained and the fine-tuned checkpoints using point prompts.



(a) Default HQ-SAM prediction

(b) Fine-tuned HQ-SAM prediction

(c) Default HQ-SAM prediction with point prompt.

(d) Fine-tuned HQ-SAM prediction with point prompt.

Figure 8: Qualitative demonstration of the fine-tuning effect. The blue region represents the segmentation predicted by the network.

the predictions became more consistent, making bad performance outliers less likely to occur.

5.2 Software Evaluation

In order to assess whether our proposed software meets its objectives (ease-of-use and annotation time reduction), we conducted an evaluation with 20 undergraduate and graduate students working on computer graphics or computer vision projects. The sample size is consistent with the recommendations of Alroobaea and Mayhew [2] for usability studies intended for statistical significance and performance metric analysis.

We instructed the participants on how to use the software and asked them to annotate leaves in two distinct images. For the first image, we chose a leaf that is more regular in shape and hence easier to annotate. For the second, we chose a leaf with a wavy surface

and a cut on the left side, making it more difficult to annotate. These images are displayed on Figure 9. Later in this work, we reference the first image as leaf 1 and the second as leaf 2.

(a) First image (leaf 1). **(b) Second image (leaf 2).**

Figure 9: Images used in the software evaluation.

The participants annotated the same two images from scratch twice: once completely manually and once only fixing the AI tool predictions. To diminish bias coming from variations on the AI prompts, we asked participants to utilize a single point prompt around the leaf's center for the first image and a box prompt for the second. We measured completion times and saved the annotations to compare later to the dataset's ground truth.

After finishing the annotations, the participants responded to a questionnaire about their user experience. Most of the questions used the 5-point Likert scale [16] to ask about how hard it was using the features in the application. Also, there was a question about whether the participant had already used other annotation tools, and an optional open field to give feedback about the system. More specifically, the questions were:

(1) Have you ever used other annotation software? (yes or no);
(2) How easy is to understand the user interface? (very easy, easy, neither easy nor hard, hard, or very hard);
(3) How easy is to use the polygon creation tool? (very easy, easy, neither easy nor hard, hard, or very hard);
(4) How easy is to add a class for the annotation polygon? (very easy, easy, neither easy nor hard, hard, or very hard);
(5) How easy is to use the artificial intelligence tool with point prompts? (very easy, easy, neither easy nor hard, hard, or very hard);
(6) How easy is to use the artificial intelligence tool with box prompts? (very easy, easy, neither easy nor hard, hard, or very hard);
(7) How easy is to fix the annotations generated by artificial intelligence? (very easy, easy, neither easy nor hard, hard, or very hard);
(8) How easy is to export the files to the desired format after finishing the annotations? (very easy, easy, neither easy nor hard, hard, or very hard);
(9) In general, how useful is the artificial intelligence tool for annotation? (very useful, useful, neutral, useless, very useless)
(10) Open field for you to leave your comments, critics, and suggestions for the application.

It is worth noting that all evaluations were done online. We passed the instructions to each participant individually via Google Meet sessions and then sent them the URL to the software program. To ensure consistency, we asked participants to use the Google Chrome browser. For the questionnaire, we used the Google Forms. Also, we only started our research after receiving the approval of the Ethics Committee on Human Research under the CAAE 78839724.3.0000.5147, opinion number 6.846.107.

Regarding the questionnaire results, in the first question, only half of the participants reported prior experience with other annotation tools. In question 9, all participants said the AI tool was "very useful" for annotation. Figure 10 summarizes the results from the second to eighth questions, which are all about the system's ease of use. We highlight that in those questions, we kept a range of 70–90% of the participants answering "very easy," with the remainder responding "easy." The only exceptions were questions 4 and 7, which had one "neither easy nor difficult" answer each.

Furthermore, we received 13 answers to the open question asking for feedback. A few of them only contained compliments. There were also a handful of suggestions, which we list below:

- To change the default class color to increase contrast;
- To show files on the files explorer as currently it only shows folders;
- To add a loading spinner while changing from one image to other;
- To add an instructions pop-up, which may be opened by default when the user accesses the tool for the first time;
- To change the icons of the export button and the add class button so they are more intuitive.

Compared to the ground truth, the annotation accuracies using manual and AI tools were similar to each other. Using the mean IoU, the manual annotation accuracy was 98.8% ± 0.4%, whereas the AI-aided annotation accuracy was 98.8% ± 0.3%. Similarly, the mean Boundary IoU was 96.1% ± 1.2% for manual annotation and 96.3% ± 0.8% for AI-aided annotation.

Investigating the annotation times, we observe in Figure 11 an overall reduction to finish the task while using AI. The mean time for manual annotation was 755 ± 399 seconds, while it was 470 ± 318 seconds for AI-aided annotation. However, we note that our data contains outliers pushing the standard deviation up, and that the time reduction was unequal for the leaf 1 and leaf 2.

To summarize the results, we calculated the speedup of AI-aided annotation compared to manual annotation. The mean speedup was 1.9× ± 1.01×, but due to outliers and a high standard deviation, we believe the median is a less biased metric. The boxplots in Figure 12 show us that while the overall median speedup was 1.5×, the median speedup for the leaf 1 was 1.3× and for the leaf 2 it was 1.7×. One possible explanation for such a difference may lie on the fact that the leaf 2 is harder to annotate, so the AI tool is more helpful. Furthermore, the prompts may have had an impact on this difference, as the box prompts used in leaf 2 carry more information and provide better mask predictions than the point prompts used in leaf 1.

Given the high accuracy of HQ-SAM predictions and the model's inference time being significantly faster than human annotation, speedup values may appear to be lower than expected. However, the

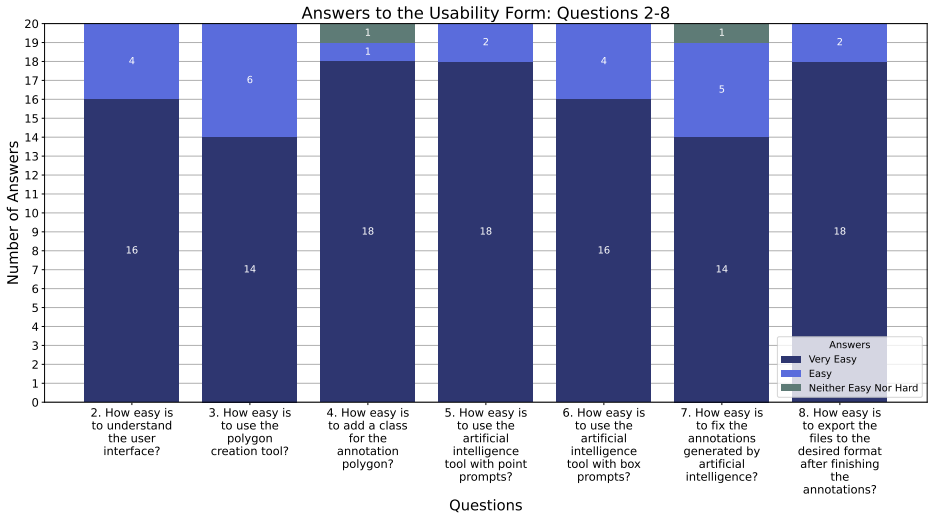


Figure 10: Answers to questions 2-8.

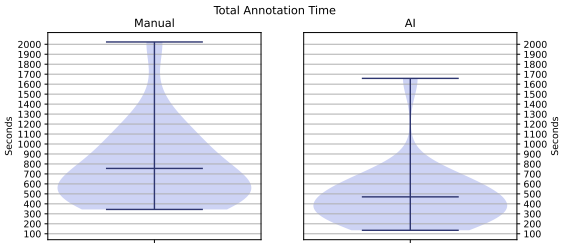


Figure 11: Summed up annotation times for the two leaves.

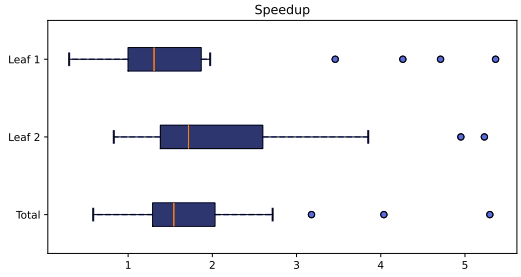


Figure 12: AI-aided time speedup compared to manual annotation.

possible reason for those results is how participants corrected the AI's annotations. Some participants passed over most of the points in the annotation polygon, making minor changes that increased their overall annotation time.

Another interesting finding is that, on average, participants who reported not having used other annotation software completed the task faster. Figure 13 shows that the median speedup for the non-experienced was 1.8×, while for the experienced it was 1.4×. We interpret the greater speedup for non-experienced users as an indication of the proposed software's ease of use and learning, which is consistent with the usability form's results. Therefore, we consider the proposed system to have completed its primary objectives satisfactorily, reducing the annotation time and being easy to use.

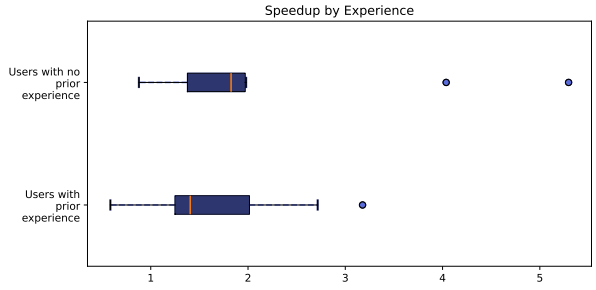


Figure 13: Comparing total speedup between participants with prior annotation experience with other tools and participants with no prior experience.

6 Conclusion

This work looked into how to diminish annotation time when creating image datasets. Our proposal was to semi-automate the annotation task. We followed the socio-technical theory to build a human-centered software that makes users and AI work together in efficiently annotating images. The software makes use of a previously trained promptable segmentation model and converts the binary mask predictions to editable polygons. By doing so, we reduced annotation to prompting and correcting prediction errors, with the expectation that humans would take less time to do it than they take in manual annotation.

We evaluated our software tool with 20 participants and verified a mean speedup of 1.9× ± 1.1× and a median speedup of 1.5× using our proposed AI tools over manual annotation. Therefore we conclude that this work was successful in its main objective

of reducing the amount of labor (measured in time) required for annotating images.

The other objectives for our tool included user-friendliness, a low learning curve, and the inclusion of an exportation feature. Considering the experiences reported by participants through our user experience questionnaire, we conclude that these objectives were also met. Furthermore, we observed that participants with no experience in annotation annotated faster than participants with experience in other tools, while maintaining comparable accuracy. This observation provides additional evidence of our proposed tool's ease of use and low learning curve.

During the development phase, our proposed software assisted in the annotation of a dataset consisting of images of bean leaves. However, the software can be applied in the annotation of other datasets for computer vision application. Therefore, our application provide a valuable contribution to computer vision researchers by speeding up the annotation process. As future work, we will attend the feedback provided by the evaluation participants and keep improving the software. We intend to demonstrate our software capabilities on other datasets, as well as extend some currently available features, such as making other deep learning models available in addition to HQ-SAM and supporting other annotation formats. Besides that, we judge it would be beneficial to add support to multiple simultaneous users annotating the same image, i.e, make the software a collaborative tool.

Acknowledgments

This work received financial support from the FAPEMIG grant number APQ-00603-21. The authors thank the Coordination for the Improvement of Higher Education Personnel (CAPES) and National Council for Scientific and Technological Development (CNPq) for their financial support.

References

[1] H Abbas and G Katina. 2023. Socio-Technical Theory. *Trist & Bamforth* 1, 2 (2023), 01–16.

[2] Roobaea Alroobaea and Pam J Mayhew. 2014. How many participants are really enough for usability studies?. In *2014 Science and Information Conference*. IEEE, 48–56.

[3] Renata Araújo and Rita Suzana. 2017. Grand research challenges in information systems in brazil 2016–2026. *Brazilian Computer Society. Clodis Boscarioli Renata Araujo and Rita Suzana* 5, 1 (2017), 2016–2026.

[4] Prachya Boonsri and Yachai Limpiyakorn. 2023. Semi-Automated Image Annotation for Cannabis Seed Gender Detection Model. In *2023 IEEE 3rd International Conference on Software Engineering and Artificial Intelligence (SEAI)*. IEEE, 189–193.

[5] Lluis Castrejon, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. 2017. Annotating object instances with a polygon-rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5230–5238.

[6] Bowen Cheng, Ross Girshick, Piotr Dollár, Alexander C Berg, and Alexander Kirillov. 2021. Boundary IoU: Improving object-centric image segmentation evaluation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 15334–15342.

[7] Karla Gabriele Florentino da Silva, Paulo Victor de Magalhães Rozatto, et al. 2025. Bean leaf image dataset annotated with leaf dimensions, segmentation masks, and camera calibration. *Data in Brief* (2025), 111328.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

[9] David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization* 10, 2 (1973), 112–122.

[10] Ionuţ Ficiu, Radu Stîlpeanu, Anca Petre, Carmen Pătraşcu, Mihai Ciuc, et al. 2018. Automatic Annotation of Object Instances by Region-Based Recurrent Neural Networks. In *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)*. IEEE, 287–291.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 2961–2969.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[13] Lei Ke et al. 2023. Segment Anything in High Quality. In *NeurIPS*.

[14] Alexander Kirillov et al. 2023. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4015–4026.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[16] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* (1932).

[17] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 740–755.

[18] Shervin Minaee, Yuri Boykov, et al. 2021. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence* 44, 7 (2021), 3523–3542.

[19] Kenneth A Philbrick et al. 2019. RIL-contour: a medical imaging dataset annotation tool for and with deep learning. *Journal of digital imaging* 32 (2019), 571–581.

[20] Krittaphat Pugdeethosapol, Morgan Bishop, Dennis Bowen, and Qinru Qiu. 2020. Automatic Image Labeling with Click Supervision on Aerial Images. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[21] Xuebin Qin, Shida He, Zichen Zhang, Masood Dehghan, and Martin Jagersand. 2018. Bylabel: A boundary based semi-automatic image annotation tool. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 1804–1813.

[22] Bhavani Sambaturu, Ashutosh Gupta, CV Jawahar, and Chetan Arora. 2023. ScribbleNet: Efficient interactive annotation of urban city scenes for semantic segmentation. *Pattern Recognition* 133 (2023), 109011.

[23] Melvyn L Smith, Lyndon N Smith, and Mark F Hansen. 2021. The quiet revolution in machine vision-a state-of-the-art survey paper, including historical review, perspectives, and future directions. *Computers in Industry* 130 (2021), 103472.

[24] Michael Sony and Subhash Naik. 2020. Industry 4.0 integration with socio-technical systems theory: A systematic review and proposed theoretical model. *Technology in society* 61 (2020), 101248.

[25] Satoshi Suzuki et al. 1985. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing* 30, 1 (1985), 32–46.

[26] Cihan Topal and Cuneyt Akinlar. 2012. Edge drawing: a combined real-time edge and segment detector. *Journal of Visual Communication and Image Representation* 23, 6 (2012), 862–872.

[27] Antonio Torralba, Bryan C Russell, and Jenny Yuen. 2010. Labelme: Online image annotation and applications. *Proc. IEEE* 98, 8 (2010), 1467–1484.

[28] Ashish Vaswani, Noam Shazeer, et al. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[29] Xue Ying. 2019. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, Vol. 1168. IOP Publishing, 022022.

[30] Jiaxin Yu, Florian Wellmann, et al. 2023. Superpixel segmentations for thin sections: Evaluation of methods to enable the generation of machine learning training data sets. *Computers & Geosciences* 170 (2023), 105232.

[31] Zhong-Qiu Zhao, Peng Zheng, et al. 2019. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems* 30, 11 (2019), 3212–3232.