Performances and Limitations Regarding GUI Elements Segmentation

Winter Semester 2023/2024

Tinouert Alexandre
University of Luxembourg
Email: alexandre.tinouert.001@student.uni.lu

This report has been produced under the supervision of:
PhD Stdt. Parvin Emami
University of Luxembourg
Email: parvin.emami@uni.lu

Abstract

In the wake of generalized Image Segmentation, models have evolved to cover an increasing number of use cases. Models have amassed billions of images to train, in order to detect all possible cases. However, as seen per the Segment Anything 1 Billion (SA-1B), Meta's dataset of 1 billion images used to train Segment Anything, a significant number of the images were left unlabeled. Moreover, the proportion of those images represents the general proportion of observable objects that exist. Since quantity is not quality, biases might exist for more specialized use cases. As such, Segment Anything can properly segment images of vast cities and landscapes with a plethora of objects, but in more specialized fields such as GUIs, there might be room for improvement. Since those models weren't specifically trained to segment GUIs, discrepancies might occur on boundaries or annotations. This BSP will mainly focus on mobile applications GUIs, as they are deemed consistent [8], with the presence of menu and status bars, as well as other design choices that make the User Experience more seamless for casual users, rather than desktop and laptop User Interfaces. This paper evaluates the current limitations of those supermodels regarding GUI segmentation and proposes improvement, either on training schemes or on fine-tuning.

1. Main Objectives

This report presents the third Bachelor Semester Project performed by Tinouert Alexandre during the Winter Semester of 2023. It informs about the developed scientific and technical deliverables, meaning a mathematical and scientific demonstration and developed software to answer the main question that this BSP raised: "What are the limitations and challenges faced when applying pre-existing models and APIs to GUI segmentation tasks?".

As GUIs are becoming increasingly complex, recurring patterns of elements can be identified within them. While the possibilities for arranging these elements are limitless, they are constructed using universally recognized elementary components. However, considering the diversity of GUI parts and the use of such elementary components, it raises the question of whether segmentation models are sufficiently efficient for this task.

Hence, this project aims to improve the visual saliency prediction of GUIs. As GUIs become increasingly complex over time with a multitude of elements in their HTML backbone, the objective is to provide an overview of the GUI designs without having to look into said backbones, but only visually. Additionally, the project aims to analyze and improve GUIs segmentation across a selection of mobile UIs, paving improvements for desktop UIs as well.

This BSP will articulate around three different parts. The first will be a general assessment of different pre-existing models of segmentation, mainly from Amazon's API and Meta's models. From those results and with the help of the manually established Ground Truth (GT), comparative data about accuracy for SAM and Amazon Rekognition can be determined. Finally, if the first assessment part is successful, a second part of the BSP will occur with the development of a fine-tuned model meeting higher requirements and accuracy than the pre-existing models for segmenting GUIs.

As Design for the Scientific Deliverable, this BSP aims to assess the efficiency of pre-existing models. First, broad tests have to be done to describe a Ground Truth using the aforementioned models. Having this Ground Truth will help calculate "efficiency" for a recurrence of different elements found, and "accuracy" for the models. Having those acquired Design definitions will help in the creation of the fine-tuned model to segment GUIs. By recalling the prerequisite knowledge, it is expected to have a general model that can attain better accuracy than the previously tested ones.

Technically, using the Scientific setup, the purpose would be to develop a model using the acquired knowledge. Using Python and the obtained database of segmented GUIs, and parameters from the Scientific Deliverable, obtaining the model is straightforward. A platform such as Roboflow allows the training of a personalized model with many pre-

processing and augmentation steps that would help for fine-tuning, and comparing the results between it and traditional general segmenting services such as SAM (barebone) and Amazon Rekognition.

To sum up the deliverables:

- Scientific Deliverable:
  - "What are the limitations and challenges faced when applying pre-existing models and APIs to GUI segmentation tasks?"
  - "How effective are pre-existing models and APIs in performing semantic segmentation on GUI?" (elaboration of a Ground Truth)
  - "Can the performance of semantic segmentation on GUI be improved by incorporating customized labels using available segmentation tools (fine-tuning pre-trained models)?".
- Technical Deliverable:
  - API Calls and Existing Models
  - Ground Truth creation
  - Development of a fine-tuned model for segmenting GUIs.

2. Targeted Competencies

2.1. Scientific Deliverable

To assess the encountered possible limitations, it is expected that multiple outputs from different pre-trained models have to be obtained. Mainly, APIs and models such as "Amazon rekognition" and "Segment Anything" from Meta are going to be used to segment different GUI based on either web parts or singular elements. Determining their efficiency is a must, but finding recurrent misses or findings is crucial, as it allows us to draw a table of efficiency for the elements themselves. At this point, it is possible to either pinpoint the accuracy of a model, tweak parameters from other models or create an entirely new model. This new model, based on previously obtained accuracy from already existing models, could surpass them in accuracy thanks to fine-tuning a dataset of manually segmented GUIs.

2.2. Technical Deliverable

From the main question, the Technical Deliverable is firstly comprised of the elaboration of API calls (Python program, or other regarding support) from the main service providers (Amazon, etc). Secondly, if the main Scientific question is answered, it is expected that a database of segmented GUI would be created (OpenCV, CVAT, etc.). Finally, if every condition is met, the creation of a Segmentation model on Python and/or Roboflow that, thanks to fine-tuning, will use the aforementioned database to verify the last optional part of the Scientific Deliverable, and conclude the Technical Deliverable itself with a model more accurate than pre-existing ones.

3. Pre-requisites

For the Scientific Deliverable, definitions of Machine Learning, and Neural Networks are needed. Every basic definition regarding the elaboration of a Neural Network and the overall process of Machine Learning will not be substantiated.

Next, for the Technical Deliverable, medium knowledge of Python is needed, with only knowledge of the specified libraries to be mentioned (PyTorch). Principles regarding the elaboration of the database will be disregarded, with only a minor overview in the complete report.

4. Performances and Limitations Regarding GUI Segmenting

4.1. Context

Image Recognition and Segmentation stands at the forefront of software advancement, continually evolving by harnessing vast datasets of images to train models that transcend various domains, serving an ever-expanding user base [2][3]. However, the emergence of such models necessitates the collection of a multitude of images (SA-1B). While these models excel at recognizing patterns, figures, and segmenting images, their inherent design leans towards generalization rather than specialization in a particular image domain. This Scientific Deliverable endeavors to unveil the current limitations of existing models, notably Segment Anything and Amazon Rekognition - two widely adopted models. The objective is to pave the way for constructing a more specialized model, specifically tailored for enhanced performance in GUI segmentations.

4.2. Segmentation Paradigms

In the context of GUI segmentation, Instance Segmentation becomes a crucial aspect. Unlike Semantic Segmentation, which focuses on classifying and labeling each pixel into predefined classes, Instance Segmentation aims to distinguish and delineate individual instances of objects within an image. This is particularly relevant in GUI analysis where identifying specific elements, their boundaries, and relationships is essential.

Metrics such as Mean Average Precision (mAP) play a pivotal role in evaluating the performance of Instance Segmentation models. mAP measures the accuracy of object detection and localization, providing insights into how well the model identifies and precisely delineates instances.

The choice of Instance Segmentation is motivated by its ability to handle overlapping elements in GUIs effectively. In scenarios where multiple UI components may overlap, Instance Segmentation ensures that each component is individually recognized and segmented, contributing to a more detailed and accurate analysis.

Understanding the trade-offs between metrics and losses in Instance Segmentation is crucial. While losses like the Mask R-CNN (Region-based Convolutional Neural Network) focus on both localization and segmentation accuracy, it's essential to balance these objectives with computational efficiency for real-time applications.

4.2.1. Object Classification.

The main goal of Image Segmentation in its entirety is to classify and segment an image according to its components.

The overall first step is Object Classification, which aims at training a Convolutional Neural Network (referred to as CNN), or other types of Machine Learning models to classify images. Typically, the phase is done over individual instances of classes covering the entire image in many variations (pose, angle, etc...) and/or augmentation steps (image rotation, brightness, color changes, etc...) which are intrinsically linked to respectively the object's possible variations, and the image's properties and alterations.

For example, this step might be where a model is trained to classify if an image is or contains a dog or a cat. Such models are then augmented to encompass as many objects as possible, being the case of SAM, or Amazon Rekognition.

4.2.2. Object Detection.

Object Detection is the natural sequel of Object Classification. A model trained with an Object Detection paradigm, with images only containing the classifiable images might struggle with bigger images containing multiple instances of the same object surrounded by asperities, or versions of an opposite item the model was supposed to classify with another object.

Segmentation is thus applied to portions of the image where the model has the highest efficiency at detecting an object. A bounding box can be drawn at the extremes of a detected item, or pixels of the object may be altered or colored.

However, there is a key difference in the main interpretation of those segmented portions containing the objects.

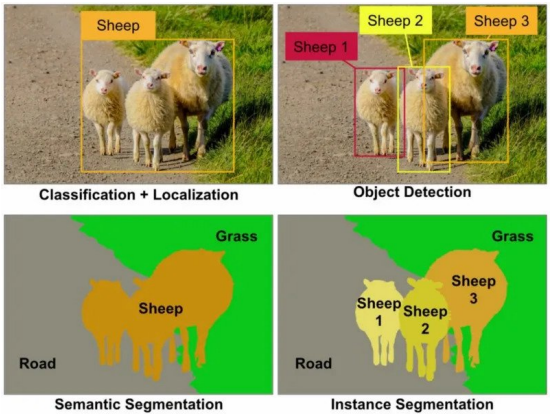
4.2.3. Semantic Segmentation.

Semantic Segmentation is often used to segment the image into different and unique classes. From the bounding boxes for each object, the selected areas will be grouped with their main classes. For unique instances of unique classes, this approach is fruitful as it gives a simple segmentation along those classes.

However, this paradigm does not fulfill the main needs of GUI Segmentation, as having multiple disjoint buttons labeled as one is not ideal for analyzing a website's visual saliency and structure only based on images.

4.2.4. Instance Segmentation.

Instance Segmentation, contrary to Semantic Segmentation, takes a more atomic approach. Rather than splitting the image according to all classes or types of objects to be found, Image Segmentation will keep the original bounding boxes made for each segmented portion of the image used to detect individual objects, and will label any individual object differently. Of course, those objects are not demoted from their original classes but are distinguishable from other objects from the same class. In the scope of GUI Segmentation, this approach is aligned with the main objectives. It is possible to distinguish multiple components of the same nature, whereas using Semantic Segmentation, would have required additional steps to convert this output into a more appropriate one, which would be in the form of an Instance Segmentation approach. All in all, those approaches have their own perks, but for the sake of GUI Segmentation, Instance Segmenting is the main paradigm of this project, as detecting a class is not equivalent to detecting an instance or an object in a GUI (determining the presence of a button in a GUI does not specify the different instances, or how many buttons are present in a GUI).



Semantic VS Instance Segmentation
Image provided by Nirmala Murali

4.3. Limitations & Challenges

4.3.1. Requirements.

To control the limitations of the models, here are the main requirements for the first sub-question:

- Filter the existing models, with the one that is deemed to be the most general in their field. It is also expected that the used models should use their most powerful or recent checkpoint.
- Use the "UEyes" dataset as a standing point for the models. A partition can be created around common specifications (mobile UIs, desktop UIs, websites).
- Segment the images using the models.
- Compare the observable flaws, either on boundaries or on annotations.

4.3.2. Design.

This sub-section serves as a repertoire for the main ideas regarding the first sub-question.

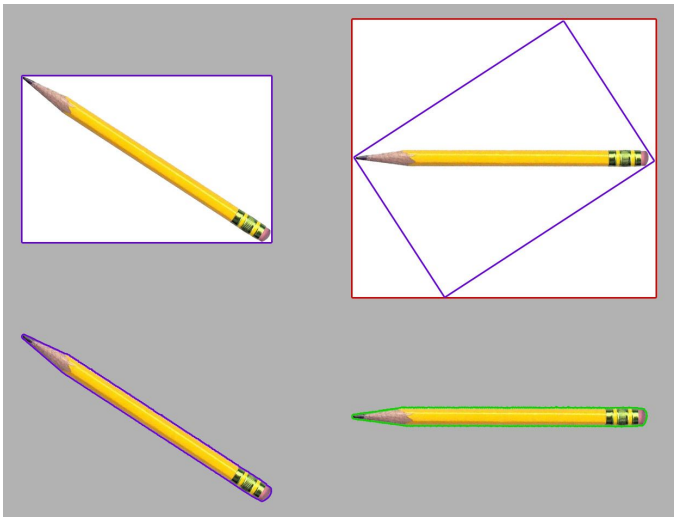
First of all, models are selected for their general segmenting competencies, in order to exclusively analyze their capacities for GUI segmentation and give possible ways of improvement in that field.

Second, regarding the ”UEyes” dataset, as it is large enough to encompass every GUI subtle element, must be partitioned. This is done for multiple reasons:

- For models that are still incomplete, or accessible by everyone and are trained through feedback, it is important to not base the only use of such models to be calibrated only for GUIs. This is different for models that require to be trained beforehand, such as Rekognition and its ”Custom Labeled” service.
- To better control and assess the general annotations among a selected few, but still coherent for GUIs.
- This approach can help in special design ways. For example, the ”Mobile First” design encourages the elaboration of a website to be optimized for mobile users first.
  A possible partition can be 10 images from phone GUIs (UEyes Dataset), to be segmented and used in the early stage to confirm the fundamental flaws through this sub-question. To increase the number of images in the process, pre-processing and augmentation steps will be applied when possible, as this helps determining if those steps help in fine-tuning a model for segmenting GUIs. Another partition can be made for the other sub-questions.

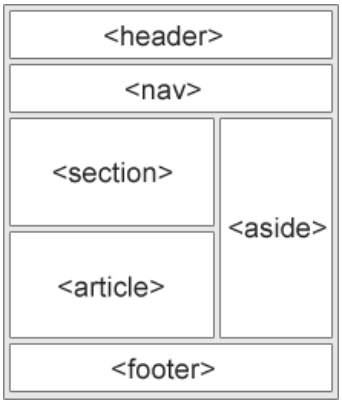
Third, one way to assess the efficiency of the models is by creating classes that represent elements commonly found in GUIs. These classes should accurately reflect the elements without overlapping or encompassing others. For instance, the class ”dog” would not be considered a valid class on its own, as dogs can appear in images or advertisements, which are common elements in GUIs. Therefore, it would be more appropriate to fit ”dog” into the ”image” or ”ads” class, which aligns better with the analysis of GUIs.

Finally, an important note should be made regarding the comparing method for training models. Segmentations and annotations will be considered as bounding boxes rather than polygons during the training process.



Differences between Bounding Boxes (up) and Polygon detection (down), image provided by Roboflow.

Although the Polygon detection method seems more accurate, in the scheme of GUIs, it is more beneficial and appropriate to consider bounding boxes, as it gives an approach similar to how HTML web applications are constructed (using blocks, inline or grid elements) as getting such structure without having to look at the intrinsic HTML layout is key for more advanced uses. Moreover, GUIs on mobiles are mostly adaptations of their counterpart on desktops [9].



An example of a HTML layout is mainly bounding boxes, or containers for their elements.

Image provided by w3schools.com

4.4. Current Models Efficiency

4.4.1. Requirements.

Using the last part’s results, it is thus possible to get the overall efficiency of the models regarding GUI segmentation. The main requirements are:

- Elaborate a Ground Truth. Using the ”UEyes” dataset, it is possible to use the same images previously segmented and elaborate the optimal segmentation.

- Compare the Ground Truth with the segmented images. A classification of the models can thus be ensured.

4.4.2. Design.

For this sub-question design, some criteria for the resolution are fixed.

The Ground Truth must be elaborated from the same classes used beforehand. However, the old pool of classes can be extended to cover a more general scope. However, the evaluation will still occur with the old pool of classes.

This database accounts for the correct expected boundaries. Those boundaries can be adjusted empirically, especially for text, or absolutely regarding the class' complexity. For example, a paragraph of text can be segmented for each letter, though this would be ludicrously hard, or it can be detected as a whole, accounting for a possible space, padding, and/or margin.

4.4.3. Elaboration of Ground Truth.

To correctly measure the accuracy of SAM and Amazon Rekognition, a Ground Truth must be made.

This Ground Truth, made with manually segmented images of GUIs on Roboflow, comprises images segmented with classes deemed important regarding GUI elements. It will be used to compare SAM and Amazon Rekognition's segmentations to evaluate their accuracy.

The main classes used as beacons for evaluation are:

- anchor (implicit link, hyperlinks as text)
- bar (visible separator)
- button (delimited action/link area)
- card (section of page or application, among a group of cards with similar structures)
- check (check mark input)
- fill-in (user text input)
- icon (various, could also be concealed, or conceived as buttons)
- image (not applicable to icons, more detailed pictures)
- link (explicit link, URL of a website)
- logo (more detailed and/or unique iconography different than icons and images)
- menu bar (Android's lower menu bar with home, return, and tabs functionalities)
- notification (implicit integration: text or number over a high-contrasted impacting pop-up)
- progress bar (explicit iconography of a colored bar over an uncolored bar to represent progression)
- status bar (Android's notification bar, to take apart due to numerous notifications. Implicit grouping of multiple notifications)
- text (explicit text outside of links, hyperlinks, etc...)

Those elements are deemed common in most Android GUIs used (especially for menu and status bars) [8][9], but also in common desktop/laptop/tablet GUIs as well (for all the other elements). The choice of those classes is determined by their presence in the chosen images to also be reproducible and common in the majority of GUIs. Moreover, most mobile applications are applets, or adaptations of website applications of their inspiration, and mainly adapted with those elements and design choices.

The notion of explicit or implicit explains the classes or subclasses of each part. For example, a hyperlink/anchor is a link, but the inverse might not be true. And those links are essentially made out of text. However, it is important to distinguish such differences, as the interaction a user will have depends on the semantics. A user might interact directly with a link or a hyperlink, whereas a simple text is made to be read.

4.4.4. Uniformity and Composition.

The Ground Truth will be elaborated with the classes above. It will be made out of 10 images from Android GUIs using those classes, to test the current models across all classes. Depending on the model, augmentation steps may be applied, to create more variations of those images.

Amazon Rekognition Custom Labels does not offer any augmentation step. Calibrating Amazon Rekognition will be done with the only purpose of manual segmentation of images without any external pre-processing or augmentation step.

One Roboflow process was done with minor pre-processing and augmentation steps:

- Flipping images horizontally.
- Rotation of images: between -45° and 45°.

The main augmentation and pre-processing steps were applied for the final Roboflow process:

- Flipping images horizontally.
- Rotation of images: between -45° and 45°.
- Exposure modifications: between -45% and +45%.
- Blur: for up to 15 pixels.
- Noise: for up to 25% of pixels blurred.
- Individual Bounding Boxes rotation: between -30° and +30°.
- Isolation of objects into additional individual images.

It is important to note that those augmentation and pre-processing steps create multiple variations of images in different forms, allowing the potential improvement of results by training more on individual objects.

4.4.5. Measuring Results of Bounding Boxes.

This section applies to the results of SAM, with no augmentation steps.

The main output of SAM is segmentations with no semantic applications. However, it is possible to extract the JSON coordinates of the bounding boxes (Technical Deliverable).

To measure the accuracy of a bounding box, it is essential to measure how much of a bounding box covers the ones from the Ground Truth. In the case of SAM, having no direct class output, obtaining the result will be done according to the bounding box's best accuracy among the Ground Truth's.

Here, using the superposition formula helps in determining the accuracy of one bounding box.

$$P(X,Y) = \frac{A(X \cap Y)}{A(X) + A(Y) - A(X \cap Y)} \tag{1}$$

P is the percentage of a rectangle X covering another rectangle Y, and A(X) is the area of X. In that case, X and Y are bounding boxes, respectively for SAM and the Ground Truth.

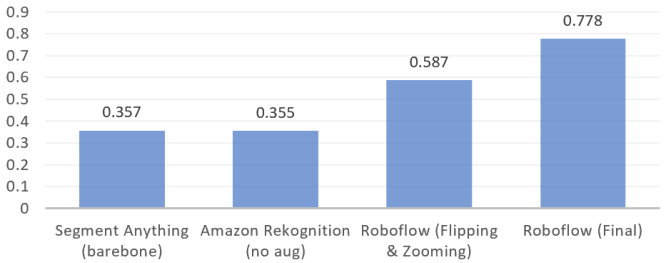
Having the accuracy of one bounding box allows for having the accuracy of all bounding boxes of one image.

As such, it is possible to calculate the Mean Average Precision (mAP) of SAM:

$$mAP = \frac{\sum_{i=1}^{n} \alpha_i}{n} \tag{2}$$

The mAP is thus the mean of all accuracies of all classes.

4.4.6. Results.



Results mAP for each Segmentation Model Service (Mobile GUIs Segmentations)

SAM, has no augmentation step. The Mean Average Precision over the selected dataset is approximately 0.357.

The first, yet limited Roboflow process using flipping and rotation as augmentation steps yielded a result of approximately 0.587.

For the final Roboflow process using all of the mentioned augmentation steps, with pre-processing of the classes into account (merging COCO with separated training over the dataset) gave a mAP of approximately 0.778.

Finally, Amazon Rekognition Custom Labels, having no integrated option enabling the use of augmentation and pre-processing steps does give a mAP of around 0.355.

Regarding SAM's results, details of the results for each class can be found in the appendix.

4.5. Improvements

4.5.1. Requirements.

Now that the main flaws or qualities of the models have been observed, optimization and improvements shall be made. To see if a fine-tuned model can outclass SAM or Rekognition regarding GUI segmentation, some requirements must be met:

- Mathematically design a Neural Network that can accommodate the images and classes. The model's development is seen in the Technical Deliverable.
- Observe the results from Technical Requirements.

4.5.2. Design.

How the model should be created shall meet some guidelines.

First, for the input itself, it should be remembered that images have a dynamic height and width. As such, adaptive input shall be examined as part of the process.

Second, other mathematical definitions not covered in previous projects will be detailed.

4.5.3. Interpretation of Results.

Taking on the previous results informs about the current accuracy of General Segmentation Models such as SAM, Roboflow's tool, and Amazon Rekognition Custom Labels.

Those models are often trained on general datasets (COCO, or SA-1B) with no specialization in any field, to stay for general use. However, those tools may not help predict the visual saliency of GUIs.

It is evident from the results that the performance of segmentation models is influenced by the extent of training data and the complexity of the classes involved. In the case of SAM, the absence of pre-processing and augmentation steps resulted in a Mean Average Precision (mAP) of approximately 0.357. However, as seen with Roboflow's iterative improvement, the introduction of more images and classes, along with

augmented data through flipping and rotation, significantly boosted the mAP to around 0.778. This underscores the importance of diverse training data and augmentation in enhancing the model's performance.

While augmenting the dataset improves the model's generalization, it is crucial to address the potential risk of overfitting. Overfitting occurs when a model becomes too specialized in the training data, leading to poor performance on unseen data. Therefore, a balanced approach, incorporating more images and classes, coupled with thoughtful augmentation and preprocessing steps, is essential. As the results suggest, finding the optimal balance can result in a segmentation model that excels in segmenting GUIs and, hence would help in getting the visual saliency of such GUIs.

As such, as Roboflow's results show, any fine-tuned model trained exclusively on GUIs may provide better results.

5. Models Implementation & Fine-Tuning

5.1. Context

This Technical Deliverable accounts for the implementations of the used models, namely Segment Anything from Meta and Amazon Rekognition. It also accounts for the creation of the fine-tuned model, based on the results of the Scientific Deliverable sub-questions.

Before implementing the models, here is a presentation of them, with motivated comments for their use.

5.1.1. Segment Anything.

Segment Anything (SAM), is a segmentation tool seconded by Meta. This model has been trained using the "SA-1B" comprising of more than 1 billion images. For now, the SAM model cannot be trained, but only adapted to needs, and has one specific model checkpoint.

5.1.2. Amazon Rekognition.

Rekognition is a service offered by Amazon as part of the Amazon Web Services. The proposed model is also constant, although there is an alternative sub-service, "Custom Labels" that allows to training of a model using a dataset of images and classes.

5.1.3. Roboflow.

Roboflow is a service offering computer vision models as well as personalized image segmentation and annotation models. Users can upload their images and manually segment them with classes to have a fully personalized model from a general detection model's checkpoint.

Since Roboflow features a tool that handles manual segmentation, with the possibility of exporting those segmentations in a JSON (COCO-formatted bounding boxes) format, Roboflow's manual segmentation also serves as Ground Truth (GT) for this BSP.

5.1.4. Overall Use and Motivations.

As such, SAM and Rekognition are widespread enough, but also general enough to be considered for implementation and comparisons. Rekognition Custom Labels also allows for further comparison between it and a fine-tuned model, as per the model's design rather than scope and starting dataset.

5.1.5. Fine-tuned Model's Requirements.

Here are the main functionalities that the fine-tuned model must encompass:

- Being able to take images as input, regardless of their size, as long as the format is conventional for images (at least for .jpg, .png images).
- Train on a partition of the dataset, and be able to transcript its efficiency while doing so.
- Testing it on another partition of the dataset and being ready to deploy as a full-on viable application.

Those functionalities will allow us to compare these fine-tuned models with the other models to fulfill the requirements of the Technical and Scientific Deliverables.

5.2. Ideas

For the established models, such as Meta, the highest-performing, most recent checkpoint is used to fulfill the Scientific Deliverable's corresponding requirement. No other specific ideas should be derived from this point.

The fine-tuned model will be elaborated on a SAM model basis and will be fine-tuned from there. The model will be divided into three separate programs: one for training, one for testing, and one for deployment/production. The weights and biases will be stored using either CSV or a conventional text file.

5.3. Model Implementation

5.3.1. Segment Anything.

Segment Anything can be deployed using Python. The most recent and developed checkpoint (vit-d) was chosen for the Scientific Deliverable. Its code implementation can be found in the Appendix.

5.3.2. Roboflow.

For the Scientific Deliverable, another model on a website interface, "Roboflow" was also chosen for more accurate point

and class accuracy and model training, and does not require any code implementation. Its personalized manual segmenting interface was also chosen for making the Ground Truth, no code is required as well.

5.3.3. Amazon Rekognition.

Amazon Rekognition does not require any code implementation, its use is web-based only.

5.4. Miscellaneous Properties

5.4.1. Python Libraries.

For the implementation and use of SAM (natively and for assessment), different libraries are needed since SAM uses them barebone.

Namely, those Python libraries are:

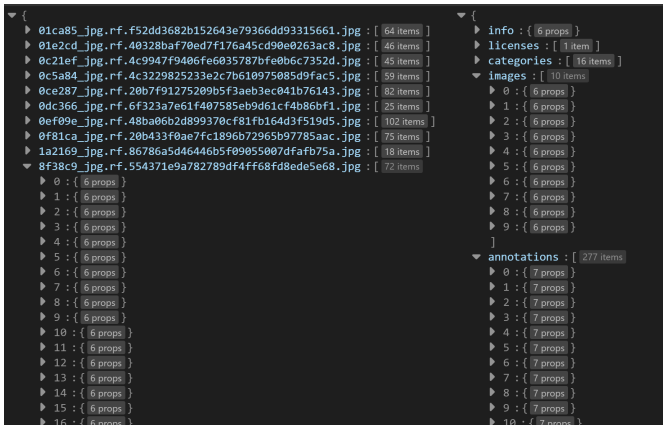
- NumPy, mainly used for enhanced calculations such as matrices multiplications and, handling the main Convolutional Networks that SAM uses.
- Matplotlib, and its sub-library Pyplot, to map the obtained results from convolution (NumPy) onto the image as a mask and display it to the user.
- OpenCV, a key library for image visualization. Here, OpenCV is used to get the images as readable inputs (binding generators) to be processed.
- JSON, for additional implementation read the Ground Truth segmentation as dictionaries, as well as write the SAM masks in JSON format.
- Codecs, helps to encode data with separators, useful for parsing dictionary to JSON.
- OS and SYS, are mainly used to process file paths of different components, such as the SAM checkpoint, the Ground Truth segmentations to analyze, and the images to segment.

5.5. Assessing the Efficiency

5.5.1. Ground Truth Format.

The main format used to handle the Ground Truth and the obtained segmentations from SAM is the JSON data format, for 'JavaScript Object Notation'.

However, the structure of the Ground Truth's JSON is different from the SAM segmentation's JSON.



Structures of SAM segmentations (Left) VS Ground Truth (Right)

The annotations, respectively each sub-object of each image in SAM's JSON and each sub-object of the 'annotations' class in the Ground Truth's JSON contain the Bounding Boxes coordinates.

5.5.2. JSON and Coordinates Processing.

Each 'bbox' (bounding box) attribute leads to a COCO-formatted array of processing coordinates as such: $[x, y, w, h]$, as for the X and Y position of the top-left corner of the bounding box, its width, and its height respectfully. Processing on the JSON has been applied to directly get the bounding boxes' second coordinate point, by simply adding the width to the X, and height to the Y.

5.5.3. Getting SAM Masks in a Readable Format.

This section addresses the design choice of having the current structure and format of the SAM segmentations' saves (in contrast to 5.5.1 Ground Truth's format)

To get the accuracy for SAM, it is crucial to obtain the generated masks in a comprehensible format, as SAM's mains direct outputs are images with their masks on. However, it is possible to extract an image's masks stored in the $masks$ variable when implementing SAM. Since those masks are not annotated, but can be processed using the instance segmentation paradigm (as well as semantic segmentation, though this approach will not be used as stated in the Scientific Deliverable).

Since the Ground Truth segmentations are ordered by categories, then by images, whereas SAM's segmentations do not feature classes or categories, classifying the segmentations with their bounding boxes for each image (see above image in 5.5.1, left) to be easily usable and readable for the final processes of this Deliverable.

Here is the main highlight for this subsection, responsible for getting masks and ' jsonifying' (returning a JSON formatted file):

Listing 1: Masks Obtention and Jsonifying.

```
import codecs, json

masks2 = {}

for file in os.scandir(dir):

    print("Current file: " + file.path)
    print("Segmentation to: " + finaldir + file.path[n:])

    image = cv2.imread(file.path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    masks = mask_generator.generate(image)

    seglist = []

    for i in range(len(masks)):
        toapp = {}
        for key in masks[i].keys():
            print("toapp: " + str(toapp))
            if key != 'segmentation':
                toapp[key] = masks[i][key]
        seglist.append(toapp)

    masks2[file.path[n:]] = seglist

    print(seglist)

    plt.figure(figsize=(20,20))
    plt.imshow(image)
    show_anns(masks)
    plt.axis('off')
    plt.savefig(finaldir + file.path[n:], bbox_inches='tight',
        pad_inches=0)

json.dump(masks2, codecs.open("sam_annotations.json", 'w',
    encoding='utf-8'),
        separators=(',', ':'),
        sort_keys=True,
        indent=4)
```

This code is applied for each image in an input file. Each images are passed through SAM and their masks ($masks$) is handled. First, a placeholder $masks2$ is created, where each attribute of $masks$ of each image is placed, into a dictionary having the structure. In short, $seglist$ contains the information of all the segmentations of an image, then $toapp$ are all the individual properties of each segmentation (including the bounding boxes as 'bbox'). Images are then shown to the users using PyPlot with the overlapping mask over the images. Finally, $masks2$ is appended to the $sam\_annotations.json$ file comprising of a single unique JSON file. $json.dumps()$ writes data into a JSON file while $codecs.open()$ with the help of separators, encodes the date (here, $masks2$) into a JSON-readable format, as $masks2$ was a dictionary

5.5.4. Dimensions Inconsistencies.

The exported Roboflow Ground Truth automatically resizes any parsed image. As a result, the Ground Truth cannot be assimilated on the same level as regular segmented images. As such, images from the Ground Truth were taken as basis, and images manually parsed into SAM had their size harmonized to better filter and process the images without any discrepancy. This resulted in a new trial of a separated batch of the same 10 images, which had their sizes meet the constraint, as this helped in comparing results afterward.

5.5.5. Data Parsing.

Now that the segmentation of the Ground Truth and SAM are obtained, before measuring the intersection, some parsing and pre-processing must be done to harmonize the formats of those (5.5.1 structure).

This subsection's implementation is straightforward, as both files are ordered slightly differently due to SAM not having any category filter (as not have classes and semantic annotations) whereas Roboflow's tool orders them automatically by category.

The following code was used to harmonize the files:

Listing 2: Segmentations Parsing and Harmonizing.

```
import json

fGT = open('...')
fSAM = open('...')

AnnsGroundTruth = json.load(fGT)
AnnsSAM = json.load(fSAM)

GTimdict = {}

for i in range(len(AnnsGroundTruth['images'])):
    GTimdict[AnnsGroundTruth['images'][i]['file_name']] =
        AnnsGroundTruth['images'][i]['id']
```

As SAM's segmentations are classified per images, which is convenient for processing, the Ground Truth's segmentations must be ordered by image so that no discrepancy can be made for the Mean Average Precision (mAP) as each class' efficiency must be calculated, and each segmentation comparison must be coherent, as segmentations of SAM and the Ground Truth must be compared within their respective image.

5.5.6. Intersection Accuracy Measurement.

As said in the Scientific Deliverable, overlaying the segmentations and measuring the intersections allows for precision and accuracy measurement.

Here is a Python highlight of the method depicted in 4.4.5:

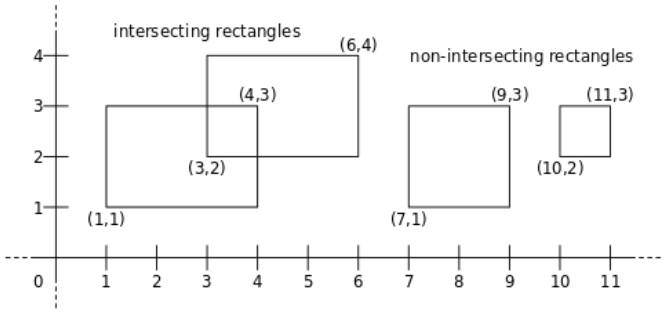
Listing 3: Intersection Accuracy Measurement.

```
def getIntsAcc(x1, y1, x2, y2, x3, y3, x4, y4):

    # Rect1 = (x1, y1), (x2, y2) the SAM segmentation
    # Rect2 = (x3, y3), (x4, y4) the Ground Truth segmentation
```

```
# Rect3 = (x5, y5), (x6, y6) the Intersection of Rect1 and
      Rect2

if x2 <= x1 or y2 <= y1:
    return 0.0

x5 = max(x1, x3)
y5 = max(y1, y3)
x6 = min(x2, x4)
y6 = min(y2, y4)

wBase = x4 – x3
hBase = y4 – y3
aBase = wBase * hBase

wInter = max(0, x6 – x5)
hInter = max(0, y6 – y5)
aInter = wInter * hInter

wSAM = x2 – x1
hSAM = y2 – y1
aSAM = wSAM * hSAM

return aInter / (aBase + aSAM – aInter)
```

This code assumes that the input values are already formatted with (x1,y1), (x2,y2) for the SAM segmentation and (x3,y3), (x4,y4) for the Ground Truth segmentation. As such, the COCO-formatted bounding boxes were preprocessed before to accommodate for the JSON structure.

It must be noted that the width and height of the intersection are either positive (images have an intersection respectively on the x's and y's) or null/negative (no intersection on x's and y's).



Intersecting Rectangles (Case 1) VS Non-intersecting Rectangles (Case 2)
Image provided by cscx.org

For example, in Case 1:

$$(x1, y1), (x2, y2) = (1, 1), (4, 3)$$
$$(x3, y3), (x4, y4) = (3, 2), (6, 4)$$
$$(x5, y5), (x6, y6) = (3, 2), (4, 3)$$
$$wInter = 4 - 3 = 1$$
$$hInter = 3 - 2 = 1$$

Hence here, in Case 1, the rectangles are well overlapping over a 1x1 rectangle.

However, for Case 2, the width and height should be adapted.

$$(x1, y1), (x2, y2) = (7, 1), (9, 3)$$
$$(x3, y3), (x4, y4) = (10, 2), (11, 3)$$
$$(x5, y5), (x6, y6) = (10, 2), (9, 3)$$
$$wInter = 9 - 10 = -1 \implies wInter := 0$$
$$hInter = 3 - 2 = 1$$

Here, in Case 2, the rectangles have at least some parts that have the same height. However, they do not overlap in the x's, which is what can be observed as the supposed intersection width is negative, which must then be given a value of 0 as the rectangles do not overlap (and later on, the area of the intersection will be 0).

Therefore, from those coordinates, areas can be calculated, then intersections, and finally the accuracy of the SAM segmentation over the Ground Truth segmentation.

5.5.7. Mean Average Precision.

The final part of the processes done in this Technical Deliverable is about the obtainment of the Mean Average Precision as described in the Scientific Deliverable. Using the previously developed parts (Data Parsing, Intersection Accuracy Measurement), by looping through the harmonized data from the JSON files of the segmentation, it is thus possible to get the mAP of SAM's results regarding GUI segmentation. Although the loop is trivial, here is the main decomposition and links between the Data Parsing part and the Intersection Accuracy Measurement processes, to extract the x's and y's from the bounding boxes (formatted $[x, y, w, h]$ with x and y the left-hand corner):

Listing 4: Main Loop Component: Parsing the x's and y's.

```
if segGT['image_id'] == GTimdict[imkey]:

    x1 = segSAM['bbox'][0]
    y1 = segSAM['bbox'][1]
    x2 = segSAM['bbox'][0] + segSAM['bbox'][2]
    y2 = segSAM['bbox'][1] + segSAM['bbox'][3]

    x3 = segGT['bbox'][0]
    y3 = segGT['bbox'][1]
    x4 = segGT['bbox'][0] + segGT['bbox'][2]
    y4 = segGT['bbox'][1] + segGT['bbox'][3]

    localAcc = getIntsAcc(x1, y1, x2, y2, x3, y3, x4, y4)

    if localAcc > bestAcc:
        bestAcc = localAcc
        bestSeg = segGT
```

The overall simple parsing comes from the harmonization done in the last parts of Data Parsing, although the structure was different beforehand when obtained directly from SAM (5.5.1).

Finally, as SAM segmentations do not have semantic annotations to the masks, contrary to the Ground Truth, getting the most appropriate segmentations equates to getting the segmentation in SAM that has the best score in Intersection Accuracy Measurement, explaining the filtering of accuracies to keep the most appropriate one while looping through SAM's segmentations in the last 3 lines. The highest accuracy obtained is thus the best fitting bounding box of a segmentation that SAM provided for one of the Ground Truth's elements.

The appropriate intersection accuracies are then compiled by class, and the process developed in the Scientific Deliverable for Mean Average Precision (means for each class) is thus applied, hence the result of SAM being approximately 0.357.

6. Conclusion

6.1. Current State and Results

In the fast-evolving realm of image recognition and segmentation, the nuanced complexities of Graphical User Interface (GUI) analysis present both challenges and opportunities. This extensive BSP has delved into the intricacies of GUI segmentation, evaluating existing models, and proposing avenues for enhancement. This development has been one of uncovering limitations, understanding paradigms, and envisioning solutions for improving performance in GUI segmentations, thus predicting visual saliency.

The evaluation of widely adopted models, such as Segment Anything (SAM) and Amazon Rekognition, revealed inherent limitations that are from their generalization-focused design. While these models showcase commendable proficiency in recognizing patterns and segmenting images across diverse domains, their inadequacies become apparent when applied to the specialized domain of GUIs. The need for extensive datasets, like SA-1B, raises questions about the scalability and adaptability of these models to specific image domains.

The delineation of segmentation paradigms, particularly the emphasis on Instance Segmentation, brought forth crucial insights. Unlike Semantic Segmentation, which classifies pixels into predefined classes, Instance Segmentation proves pivotal in GUI analysis. The ability to distinguish individual instances of objects within an image becomes paramount, especially when dealing with getting the accuracy of UI components segmentations.

The stages of Object Classification, Object Detection, Semantic Segmentation, and Instance Segmentation were dissected to emphasize their roles in GUI analysis. Object Classification, as the initial step, sets the foundation by training models to classify images based on their components. Object Detection follows suit but introduces challenges when confronted with larger images containing multiple instances of the same object. Semantic Segmentation, while effective in categorizing images into distinct classes, falls short in capturing the intricacies of GUIs, where the need for individual distinction among UI elements is paramount. In this context, Instance Segmentation emerges as the main paradigm for GUI Segmentation, aligning with the project's objectives.

The requirements and design considerations for evaluating existing models underscore the need for a meticulous approach. The emphasis on selecting the most general models, partitioning datasets, and creating classes representative of GUI elements serves as a guideline for meaningful evaluations. The exploration of challenges and constraints reinforces the importance of fine-tuned models specifically designed for GUI segmentation, rather than relying on general-purpose models trained on diverse datasets. Seeing the Scientific Deliverable's demonstration of getting classes and segmentation's accuracies is on par with its requirements, and its completion.

The final phase of the BSP opened doors to potential enhancements in GUI segmentation. The proposal of a mathematical design for a general approach to measuring segmentations' accuracy to GUI images and classes sets the stage for a coherent method. The consideration of classes, augmentation, and pre-processing steps as well as the obtention of a Ground Truth over common elements of GUIs (particularly in mobile but not excluding the scalability of desktop GUIs) showed interesting results. The promising results from fine-tuned models exclusively trained on GUIs, as seen by Roboflow's tool, hint at a future where tailored solutions would improve their more generalized counterparts by fine-tuning as well (SAM) as a follow-up.

This comprehensive Scientific Deliverable traversed the landscape of GUI segmentation, its characteristics, complexities, and envisioning solutions for getting accuracy and Mean Average Precision (mAP) of existing models over GUIs. The limitations of existing models, the significance of segmentation paradigms, and the challenges in evaluating GUI segmentations were dissected with precision. The proposed improvements, rooted in training design and considerations, present a roadmap for future advancements and follow-ups.

The elaborated Technical Deliverable also delivered primordial results, especially in the claims in the Scientific Deliverable ('generalized models fail in some precise fields'). Obtaining SAM's, Amazon's, and Roboflow's results show that, the more generated images, the more training, preprocessing, and augmentation steps there are, the best a fine-

tuned model will be in contrast to the generalized models (SAM and Amazon versus Roboflow with different training steps).

As we conclude this BSP, the development from limitations to possibilities becomes evident. The path forward involves not only refining existing models but also designing specialized solutions that resonate with the special intricacies of GUI segmentations. In the dynamic intersection of technology and user interfaces, the advancements for precision and efficiency in GUI segmentation continue, where GUI segmenting would continually improve, so that related works (predicting visual saliency) would benefit from more precise and accurate GUI structures overview without having to dive into the intrinsic structure.

6.1.1. Follow-up.

The evaluation of GUI segmentation models has revealed intriguing insights, with Roboflow emerging as a standout performer, showcasing remarkable results at approximately 77%. This substantial leap in performance compared to conventional segmentation services like SAM and Amazon Rekognition opens a gateway to further exploration and refinement in the realm of GUI segmentation. Particularly the use of a more extended portion of segmented GUIs coupled with a fine-tuned version of one of the showcased models (such as SAM) could show even better results.

The overall follow-up leads to more open uses of SAM and fine-tuning SAM to get more accurate results than the ones obtained, in future works or projects.

7. Plagiarism statement

I declare that I am aware of the following facts:

- I understand that in the following statement the term "person" represents a human or any automatic generation system.
- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:
  1) Not putting quotation marks around a quote from another person's work
  2) Pretending to paraphrase while in fact quoting
  3) Citing incorrectly or incompletely
  4) Failing to cite the source of a quoted or paraphrased work
  5) Copying/reproducing sections of another person's work without acknowledging the source
  6) Paraphrasing another person's work without acknowledging the source
  7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
  8) Using another person's unpublished work without attribution and permission ('stealing')
  9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

  Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

References

[1] Shidan Wang, Donghan M. Yang, Ruichen Rong, Xiaowei Zhan, Guanghua Xiao, Pathology Image Analysis Using Segmentation Deep Learning Algorithms, The American Journal of Pathology, Volume 189, Issue 9, 2019, Pages 1686-1698, ISSN 0002-9440, https://doi.org/10.1016/j.ajpath.2019.05.007.

[2] Segment Anything (2023), Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, Ross Girshick. https://doi.org/10.48550/arXiv.2304.02643.

[3] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 7, pp. 3523-3542, 1 July 2022, doi: 10.1109/TPAMI.2021.3059968.

[4] Yue Jiang, Luis A. Leiva, Hamed Rezazadegan Tavakoli, Paul R. B. Houssel, Julia Kylmälä, and Antti Oulasvirta. 2023. UEyes: Understanding Visual Saliency across User Interface Types. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23). Association for Computing Machinery, New York, NY, USA, Article 285, 1–21. https://doi.org/10.1145/3544548.3581096.

[5] MISHRA, Abhishek. Machine learning in the AWS cloud: Add intelligence to applications with Amazon Sagemaker and Amazon Rekognition. John Wiley & Sons, 2019.

[6] Luis A. Leiva, Xue Y., Bansal A., Tavakoli H. R., Körolu T., Du J., ... & Oulasvirta, A. (2020, October). Understanding visual saliency in mobile user interfaces. In 22nd International conference on human-computer interaction with mobile devices and services (pp. 1-12). https://arxiv.org/abs/2101.09176

[7] Hu, R., Chen, M., Cai, L., & Chen, W. (2020). Detection and segmentation of graphical elements on GUIs for mobile apps based on deep learning. In Mobile Computing, Applications, and Services: 11th EAI International Conference, MobiCASE 2020, Shanghai, China, September 12, 2020, Proceedings 11 (pp. 187-197).Springer International Publishing. http://dx.doi.org/10.1007/978-3-030-64214-3_13

[8] Lumpapun Punchoojit, Nuttanont Hongwarittorrn, "Usability Studies on Mobile User Interface Design Patterns: A Systematic Literature Review", Advances in Human-Computer Interaction, vol. 2017, Article ID 6787504, 22 pages, 2017. https://doi.org/10.1155/2017/6787504

[9] Sahami Shirazi, A., Henze, N., Schmidt, A., Goldberg, R., Schmidt, B., Schmauder, H. (2013, June). Insights into layout patterns of mobile user interfaces by an automatic analysis of android apps. In Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems (pp. 275-284).

8. Appendix

8.1. SAM Implementation

Listing 5: SAM Implementation.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os

dir = 'UEyes-DS/part1_mobile/'
finaldir = 'UEyes-DS/part1_SAM/'
n = len(dir)

def show_anns(anns):
    if len(anns) == 0:
        return
    sorted_anns = sorted(anns, key=(lambda x: x['area']),
        reverse=True)
    ax = plt.gca()
    ax.set_autoscale_on(False)

    img = np.ones((sorted_anns[0]['segmentation'].shape[0],
        sorted_anns[0]['segmentation'].shape[1], 4))
    img[:,:,3] = 0
    for ann in sorted_anns:
        m = ann['segmentation']
        color_mask = np.concatenate([np.random.random(3),
            [0.35]])
        img[m] = color_mask
    ax.imshow(img)


import sys
sys.path.append("..")
from segment_anything import sam_model_registry,
    SamAutomaticMaskGenerator, SamPredictor

sam_checkpoint = "SAM/sam_vit_h_4b8939.pth"
model_type = "vit_h"

device = "cuda"

sam =
    sam_model_registry[model_type](checkpoint=sam_checkpoint)
sam.to(device=device)

mask_generator = SamAutomaticMaskGenerator(sam)

for file in os.scandir(dir):
    print("Current file: " + file.path)
    print("Segmentation to: " + finaldir + file.path[n:])

    image = cv2.imread(file.path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    masks = mask_generator.generate(image)

    plt.figure(figsize=(20,20))
    plt.imshow(image)
    show_anns(masks)
    plt.axis('off')
    plt.savefig(finaldir + file.path[n:], bbox_inches='tight',
        pad_inches=0)
```

8.2. SAM Classes Results

```
Precision for anchor: 0.13
Precision for bar: 0.15
Precision for button: 0.65
Precision for card: 0.61
Precision for check: 0.71
Precision for fill-in: 0.74
Precision for icon: 0.4
Precision for image: 0.31
Precision for link: 0.02
Precision for logo: 0.09
Precision for menu bar: 0.59
Precision for notification: 0.71
Precision for progress bar: 0.05
Precision for status bar: 0.07
Precision for text: 0.11
```

Figure 1: SAM Results per classes