Data Challenge ENS
Learning radiological and oncological anatomy with few shots learning - by Raidium

Mathias Vigouroux, Lucas Haubert

École Normale Supérieure, Master MVA
<mathias.vigouroux, lucas.haubert>@ens-paris-saclay.fr

March 18th 2024

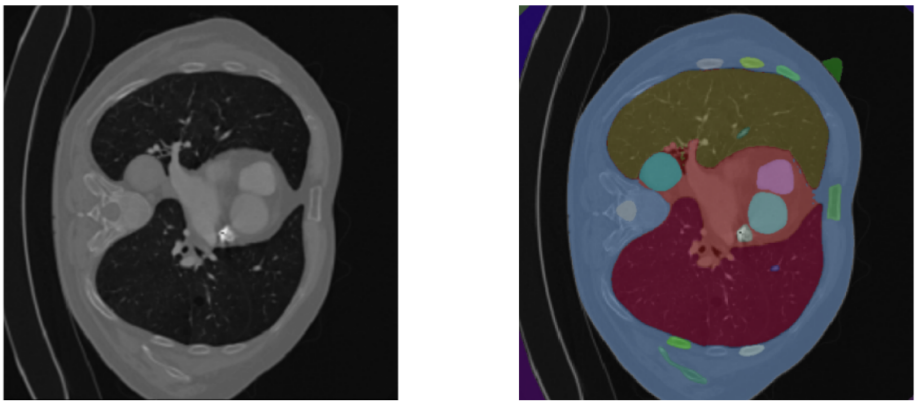


Figure 1: An example of segmentation on a CT-Scan slice obtained with SAM [2]

Abstract

This project is relative to the Data Challenge[1] for the course "Learning and generation by probabilistic sampling" given par S. Mallat, Collège de France. It explores the image segmentation task, aiming to split an image into multiple segment regions, often to identify and locate objects and boundaries within the image. In the context of few-shots learning, integrating a relatively small number of labeled data, i.e. provided with segmentation masks, constitutes a relevant data challenge, close to the real-world constraints. Several approaches based on deep learning architectures, such as U-Net, Detectron, and SAM, are considered in this paper. Additionally, an original learning pipeline for segmentation is also designed and implemented, to get as close as possible to the underlying medical expertise, but also to meet the requirements imposed by few-shots learning. This work results in segmentation scores higher than those of the Challenge baseline, as well as reflections on the relevance of using each of the approaches considered. For replication purposes, the code for this project is stored in our public Google Colab repository[2].

[1]Data Challenge ENS: https://challengedata.ens.fr/participants/challenges/150/
[2]Code: https://drive.google.com/drive/folders/1j8b1UhqwQ3stTe9c6qFddghocIMpd8Sv?usp=sharing

1 Task Definition

CT-scans provide highly accurate 3D images of the human body, enabling us to grasp the human anatomy. This challenge aims to automatically segment the anatomical structures of the human body, as well as tumors, on a CT scan, without semantically identifying the associated organs. This object discovery problem is an intuitive and straightforward one for a human, who will be able to identify new objects on a scene very simply, and even on a CT-scan, even if it has never seen the object before.

While supervised segmentation algorithms for these individual structures are now considered solved, it is not possible to use supervised learning to generalize to new, previously unseen anatomical structures. In this context, the provided training data are made of two groups: images with segmentation masks associated, and raw images which can be used in an unsupervised setting. Finally, the Rand index [8], implemented in Scikit-Learn[3] is adopted as the basis of the evaluation metric. The final metric is calculated by averaging the Rand index between each label and its associated prediction while excluding background pixels (i.e. 0 in the label). This clustering metric is invariant to the inter-frame permutation of the structure number.

2 Methodology

The methodology can be introduced as follows: We are first using the provided baseline as a reference model. It is a gradient-based method, meaning that the segmentation is built based on the gradients between pixels. This approach is simple, but not sufficient according to the chosen metric Rand-based metric. Then, pre-processing steps, as well as deep learning methods are leveraged, to capture the relevant frontiers between the areas of the image.

2.1 Baseline model

A source code, serving as a reference, is given as an entry point into our project and obtains a score of around 13% in the validation set, made of 200 training data. The objective of this challenge will be to maximize this score by modifying the given baseline. This baseline is based on a gradient approach, meaning that the difference, in terms of intensity, between neighbor pixels, will help to draw the segmentation frontiers.

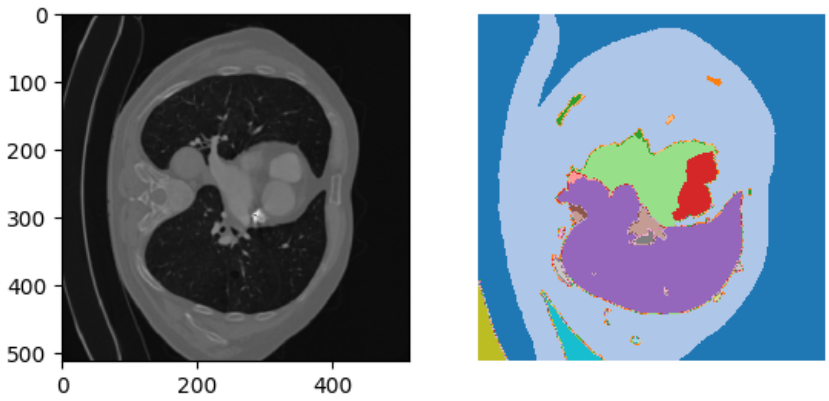


Figure 2: One segmented slice with the baseline model

[3]Implementation of Adjusted Rand Score: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

The entry image is first passed through a Sobel filter [11] to detect sharp variations of intensities in the image. It uses two convolution matrices of size 3×3 to approximate horizontal and vertical gradients in the image, to store strong variations, then possible frontiers, in the image. This step is followed by a median-based denoising of size 2. Then, segmentation markers are computed, to find homogeneous areas. Finally, the watershed segmentation algorithm [4] is applied. This method uses detected edges and previously generated markers to segment the image. Edges help define where the boundaries between segments should be, while markers help initialize growth regions. The compactness parameter adjusts how the segments are shaped, with a low value favoring more compact segments. Details are given in Appendix A.1

Despite the ingenious design of this approach, the quality of the segmentation leaves much to be desired. Some areas of the organ, such as the cavity at the bottom of the image, appear quite well segmented. However, the second cavity was not detected at all. Also, some specific areas, such as the two central glands, are too coarsely delineated to be of assistance to a medical professional. This result implies a significant margin for improvement, which is primarily due to the simplicity of the proposed baseline model. For these reasons, it is relevant to consider other approaches, which are introduced in what follows.

2.2 Pre-processing

A first possible improvement to our approach is to introduce a data pre-processing step, to make the data cleaner and more suitable for segmentation. Two steps are explored here: enhancing the contrast of the image and denoising it. Increasing image contrast will subsequently help to differentiate the slice zones from one another. Secondly, denoising allows us to get rid of details at the microscale that are not information-bearing, and may therefore hinder the clustering task.

CLAHE (Contrast Limited Adaptive Histogram Equalization) [12] is used to increase the contrast. Unlike traditional histogram equalization that applies a global contrast transformation based on the overall image histogram, the model works by dividing the image into small blocks called tiles and then applying histogram equalization to each of these blocks separately. This approach allows for local contrast enhancement, making it effective in improving the visibility of features across different regions of the image.

In a second time, the denoising step is done on the enhanced image. Two filters, implemented in Scikit-Learn[4] are considered. The first one is the Bilateral filter [18]. It maintains edges while averaging pixels, taking into account their spatial proximity and similarity in color intensity. The measure of spatial proximity is determined by the Gaussian function applied to the Euclidean distance between two pixels, with a predefined standard deviation. Importantly, the calculation of weights takes into account both the Euclidean distance between pixels and the radiometric variations (for instance, differences in color intensity, depth, etc.). This method ensures that sharp edges are maintained.

On the other hand, the Total Variation Chambolle filter [5] is based on the principle of total variation (TV), a concept that measures the sum of absolute gradients in the image. The key idea behind total variation is that in natural images, variations in pixel intensity are generally small except where they correspond to the edges of objects. So, by minimizing total variation, the image is smoothed while the important edges are preserved. Note that this filter is parameterized by a denoising weight $\frac{1}{\lambda}$ being proportional to the denoising power. In other words, $\lambda > 0$ weights fidelity data, then as $\lambda \to 0$, the total variation dominates the objective function

[4]Denoising filters in Scikit-Learn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

of the underlying optimization problem, then leads to a smaller total variation of the solution, this latter looking less like the input data.

A visualization of the filter effects can be made, to evaluate the effect of each pre-processing step at a glance. Figure 3 displays the evolution of an entry, before being segmented. It is first passed through the contrast enhancer CLAHE, before being denoised either by Bilateral or TV Chambolle filter. First, the contrast is drastically increased, making it easier to differentiate the different areas. Then, the slice is denoised, leading to different results in terms of details and texture. Also, notice that the filters are parameterized, then the results can be slightly changed.

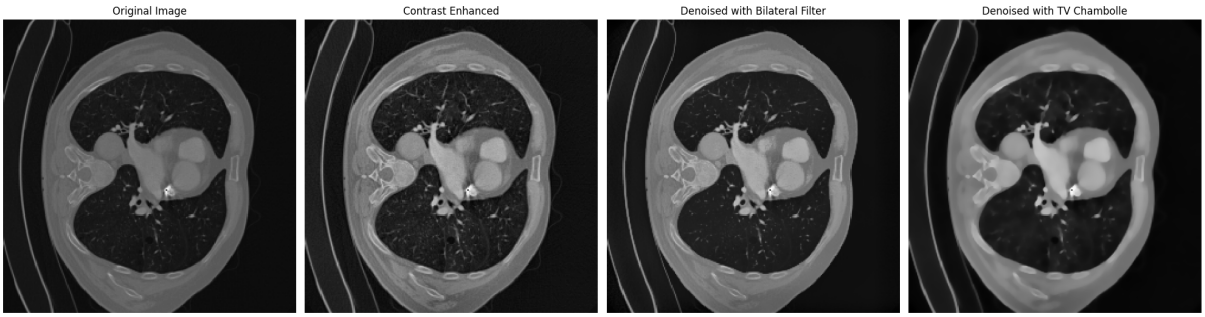


Figure 3: An overview of the pre-processing pipeline. Bilateral filter with $\sigma_{spatial} = 15$ and TV Chambolle filter with $\frac{1}{\lambda} = 0.1$.

Before making a quantitative comparison of the performance of the two filters, it is first possible to illustrate some algorithmic considerations. Despite its simple construction, the Bilateral filter can be particularly slow when input images are large. For each pixel, we need to calculate weights based on intensity differences and spatial distances. This, multiplied by the number of pixels, and the number of images to be processed, can result in a very long calculation time. On the other hand, the TV Chambolle filter provides more guarantees in terms of complexity. Fewer iterative loops are involved in solving the system for minimizing the variation function. Indeed, the pixels only are iterated, while the Bilateral approach involves multiple additional windows. Note that the described approaches are implemented in Scikit-Learn[5]. For these reasons, as well as the experiments presented in Section 3.2, the TV Chambolle filter is adopted in the segmentation pipeline.

2.3 U-Net

To propose a new method, the idea of neural networks quickly emerged, as these models are effective in many computer vision tasks. To this end, one of the most cited references is U-Net [13]. This convolutional architecture is powerful when it comes to segment images, especially biomedical ones.

U-Net architecture is characterized by its U-shape, made up of two main parts: the contraction path and the expansive path. The contracting path utilizes a conventional convolutional network structure, applying two 3×3 convolutions without padding in succession. Each convolution is followed by the activation function rectified linear unit (ReLU) and a 2×2 max pooling operation with a stride of 2 to reduce the spatial dimensions. With each step of downsampling, the quantity of feature channels is increased twofold. Conversely, each stage in the expansive path begins with upsampling the feature map, followed by a 2×2 convolution that reduces the feature channels by half. This is followed by merging it with the appropriately trimmed feature

[5]skimage.restoration: https://github.com/scikit-image/scikit-image/blob/v0.22.0/skimage/restoration/

map from the contracting path and then performing two 3×3 convolutions, each succeeded by a ReLU. Cropping is required to compensate for the border pixels lost during each convolution. To allocate each 64-component feature vector to the required number of classes, a 1×1 convolution is employed in the ultimate layer. Altogether, the network comprises 23 convolutional layers. Figure 4 gives a visual representation of the network.

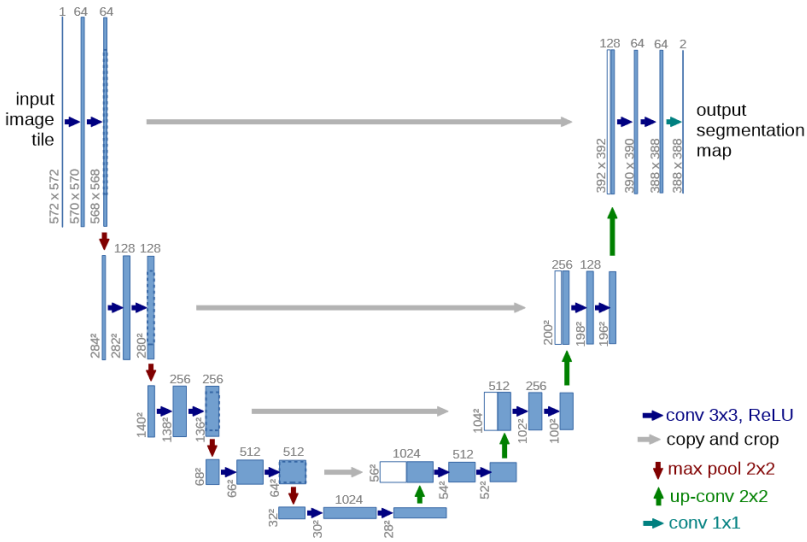


Figure 4: U-Net architecture, taken from [13]

The key feature of this network that caught our attention is that it can be trained with little labeled data (i.e. with an associated ground truth segmentation mask), and at the same time perform very well on biomedical data. This is made by data augmentation, which consists of generating, using Gaussian distributions and bicubic interpolations, new images from the training dataset, and making the segmentation robust to shifts, rotations, and elastic deformations. This paradigm leads to less training data needed, which corresponds to the challenge's framework, but also good performances on biomedical data, since the underlying transformations are inspired from biology.

However, the network configuration for the data challenge may not be optimal. U-Net learns to segment images based on the consistency of labels across the training dataset. If a structure is represented by different label numbers from image to image (for example, given in the challenge's description, the liver can be labeled as 4 in one image and 1 in another), this can confuse the network, reducing learning efficiency and segmentation accuracy. This discrepancy between the nature of the problem and the construction of the model leads us to consider alternative approaches. Among them, it may be possible to consider only one class to detect and segment, that represents organic components, by opposition with the background. The challenge is then to perform segmentation among two labels. Due to technical considerations concerning the nature of the network, and others like the ones of Detectron2 in Section 2.5, U-Net is not considered as the optimal choice, since we discovered this point of view only later when playing around with Detectron2.

2.4 SAM

Other neural networks can meet our needs, depending on the nature of the data involved. Among them, the Segment Anything Model (SAM) [2] appears to be a relevant solution. The model seeks to overcome some of the limitations of traditional segmentation approaches, notably the need for large amounts of specifically annotated training data and the difficulty of generalizing

to new objects or scenes.

Pre-trained on massive datasets from the web, large language models are transforming the field of natural language processing with their robust capabilities in both zero-shots and few-shots learning. When these models are scaled up and trained on extensive web-based text corpora, their performance in zero and few-shots learning scenarios is surprisingly competitive, even reaching parity with models that have been fine-tuned in some instances. Segment Anything is inspired by this idea. SAM's paper [2] begins by establishing a segmentation task that is broad enough to serve as an effective pre-training goal and to facilitate a variety of applications downstream. The idea of SAM is then to return a valid image segmentation given any prompt, prompting methods being a parameter that depends on the user's query. As in the case of Large Language Models, the goal is to provide an ambiguity-robust solution, given a prompt that can be textual, or visual (points, boxes, etc.).

A straightforward design meets all constraints of the Segment Anything task: an effective image encoder generates an image embedding, a prompt encoder processes prompts, and then these two sources of information are merged in a streamlined mask decoder that produces segmentation masks. Figure 5 highlights the architecture of the model.

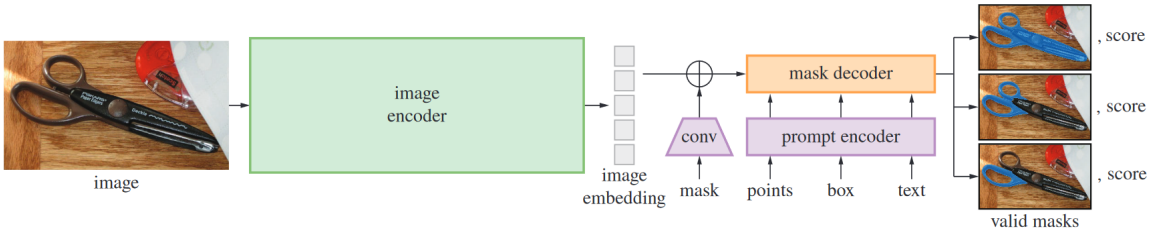


Figure 5: SAM architecture, taken from [2]

Driven by the need for scalability and effective pre-training approaches, the authors opt for a Vision Transformer [1] pre-trained using Masked Autoencoder [10], with minimal adjustments to handle high-resolution inputs. The image encoder is executed once per image and can be utilized before prompting the model. Prompts are split into two groups: sparse (including points, boxes, and text) and dense (masks). Points and boxes are encoded through positional encodings combined with learned embeddings specific to each type of prompt, while free-form text is processed using a pre-built text encoder from CLIP [3]. Dense prompts, namely masks, are encoded via convolutions and then integrated with the image embedding through element-wise summation.

The mask decoder adeptly transforms the image embedding, prompt embeddings, and an output token into a mask. This setup modifies a Transformer decoder block, supplemented by a dynamic mask prediction head. The decoder block employs self-attention for prompts and cross-attention in both directions (from prompt to image embedding and back) to refine all embeddings. Following the execution of two blocks, the image embedding is upsampled, and a Multilayer Perceptron (MLP) converts the output token into a dynamic linear classifier. This classifier then determines the mask's foreground likelihood at each location in the image.

When presented with an ambiguous prompt, the model, if restricted to a single output, will blend several valid masks together. To counteract this issue, the model is adapted to generate multiple output masks in response to one prompt. In fact, producing three mask outputs typically suffices to cover the majority of scenarios (considering that nested masks rarely exceed three levels: whole, part, and subpart). During the training phase, backpropagation is applied to only the minimum loss [7] across the masks. Additionally, to order the masks, the model assigns a

confidence score (i.e., predicted Intersection over Union IoU) to each mask. The mask prediction process is trained using a linear combination of focal loss [17] and dice loss [6].

Experiments were carried out in Section 3.3 on the pre-trained SAM model. This resulted in significant computation time, with results similar overall to the pre-processed baseline method. The underlying explanation is that this approach lacks relevance, given the datasets of labeled, i.e. segmented, images provided in the Challenge. Then, the fine-tuning of SAM is considered, but faces multiple practical limitations, due to the source code and the ergonomics of the implementation. For this reason, another approach is explored in the next subsections.

2.5 Detectron2

Detectron2 is a PyTorch-based modular object detection library. It was developed by Facebook AI Research, before SAM, and aims at giving multiple image processing tools, of which segmentation. Detectron2 is an extension of Detectron that is easy to handle and provides fast training on single or multiple GPU servers. In particular, it contains an implementation of a segmentation algorithm, which is the reason we are considering the library. The underlying models also bring high ergonomics, making them more suitable in the Challenge's context.

The dataset COCO [16] have been used to pre-train underlying segmentation models in Detectron2. It comprises more than 330,000 images, each labeled with 80 categories of objects and accompanied by 5 descriptive captions of the scene. It is extensively utilized in the field of object detection and segmentation. The images are annotated, and the labels contain object classes, but also bounding box coordinates and segmentation masks (in polygon or Run-Length Encoding (RLE) format). This dataset also came in a certain format, and it was necessary to make the dataset of the challenge in this format too.

Various models are hosted in the Detectron framework. Versions of R_50, R_101 and X_101, inspired from ResNet [9] (ResNet_50, ResNet_101 and ResNeXt-_01 backbone), are implemented. The strengths of individual models can complement each other to achieve better overall performance. Each model may capture different aspects of the data, which can be relevant, as explained in Section 2.6.

Given an input image, the Detectron2 environment returns an image segmentation with instance recognition and match probability. More precisely, for a threshold set by the user, the model returns segmented instances whose inference is safe at a level above this threshold. It is important to note that the trivial use of this approach does not correspond to the context of the Challenge. The aim here is to cut out the various identifiable organic components in a radio image, without necessarily labeling them. An adaptation of the Detectron2 approach is therefore necessary and was realized during our experimentation with the model. Hence, we decided to artificially consider two classes: biological component and background. By construction, the action of Detectron2 then relates to the task imposed by the challenge, i.e. segmenting biological components between themselves, without differentiating them from a semantic point of view. Figure 6 illustrates this approach.

An additional point is the choice of the segmentation paradigm, which could be either a polygon approach or a binary mask approach. Unlike polygon segmentation, which uses contours defined by points, mask segmentation assigns each image pixel a label indicating whether it belongs to an object or the background. Thus, masks can be efficiently processed by convolution-based networks for instance segmentation. In this context, and regarding the expectations of the Challenge, the mask approach is adopted.

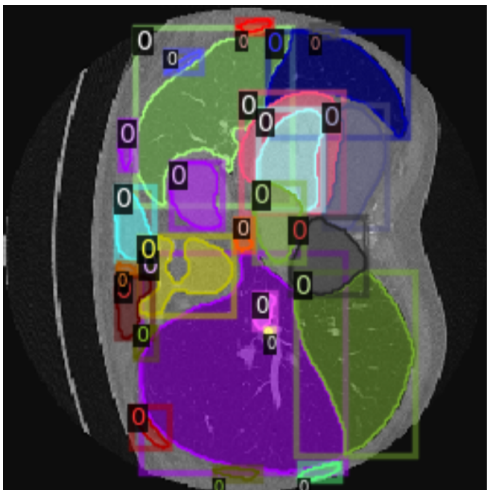


Figure 6: Adapted format of segmentation for Detectron2: Bounding boxes are added to each of the organs, and the label of each organ i set to 0 while it was before a randomly variable number.

Finally, we choose to use models based on Detectron2 in order to solve the Challenge. Due to the technical issues encountered with SAM, one had to adopt U-Net or Detectron to focus on the submissions. In the context of few-shots learning explained in 2.6, it may be important to deal with different models in the same framework. Because of the variety of methods implemented in Detectron2, and their efficiency, we have ruled out U-Net networks. Then the experiments in the context of few-shots learning are an on Detectron2 in Section 3.

2.6 Few-Shots Learning

The context and constraints posed by the Data Challenge prompt us to use the few-shots learning paradigm. Indeed, only a small proportion of the supplied data is labeled, i.e. segmented. This situation is realistic in a radiology context. Indeed, given the large number of radio recordings that scanners can provide, it is difficult to conceive of total intervention by radiologists to label the data. The task would be too time-consuming and not sufficiently accurate over time if a single radiologist were assigned to label the data, while the intervention of several human actors could introduce biases into the annotation stage.

A data processing pipeline must be considered, depending on whether the data is labeled, and also on the freedoms offered by the Detectron2 framework. The idea is inspired by the Pattern-Exploitation Training paradigm in NLP[14], which consists of leveraging the efficiency of pre-trained models and then dealing with a few numbers of training data, for which the task was simplified. Also note that the simplification of classes paradigm is similar, in that only two classes are considered. Using the Challenge's specifications, we designed and implemented the following pipeline. One important finding related to this pipeline is that it is possible to leverage small "models" and make them good few-shots learners [15]. Due to our limited computational resources (we only had access to Google Colab Graphical Processing Units) we had the intuition that this approach might yield good results. We consider three intermediate models that will be combined to label all unlabeled data. In this way, we will have a greater number of soft-labeled data than initially available. This will enable a further segmentation model to train for the Challenge task. Figure 7 highlights the learning architecture that is used to deal with the few numbers of labeled data.

We start with the current data, i.e. there are 400 labeled and 1600 unlabeled. This represents two different groups of utilities in the pipeline. First, we split the annotated data into training

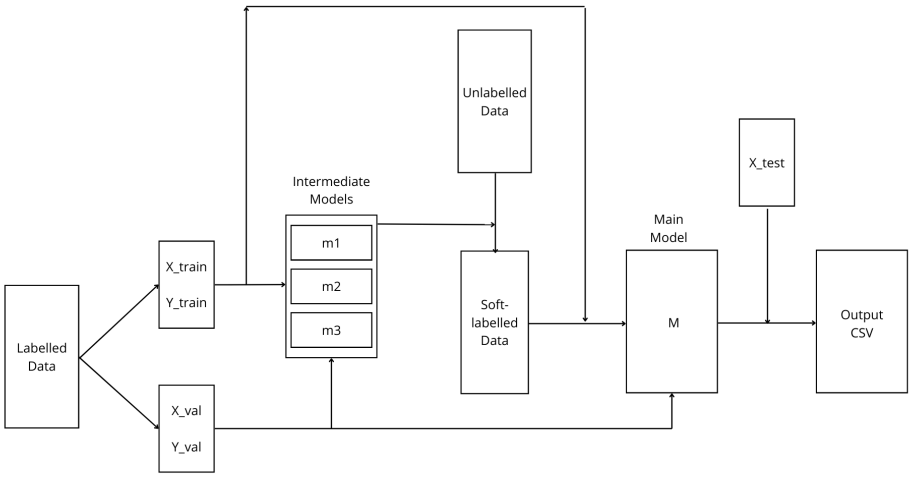


Figure 7: Own Pipeline: Few-Shots Learning setting, from the initial datasets to the CSV submission file

data and validation data. The validation dataset, which represents 20% of the annotated images, will be involved as test data to evaluate the models m1, m2, m3, and M during the learning process.

Intermediate segmentation models m1, m2, m3 are here to work on unlabeled data. In particular, the idea is to simulate the work of three radiologists, trained in different ways, to label unlabeled data. To do so, m1, m2, and m3 are trained with the annotated dataset X_train and Y_train. Different epochs and learning rates are considered, to build various models (radiologists with different approaches). Then, their performances are evaluated with the validation dataset X_val and Y_val. The evaluation metric is the Average Precision (AP) over the predictions. It consists of averaging the precision across multiple Intersections over Unions (IoU) thresholds. Then the AP of the three different models are stored and converted as normalized weights. It is a way to compare the segmentation performances between the three intermediate radiologist models.

Unlabeled data are then considered, and treated split into three groups of which the size is proportional to the intermediate model's weights. Thus, each group 1, resp 2, resp 3 is passed through the models m1, resp m2, resp m3. We obtain three sets of predictions, each one being made by its related intermediate model. Finally, the whole set of predictions is merged, and stored as soft-labeled data. This way of proceeding allows us to give importance to relevant intermediate models - for example, if the weight of m2 is high, then a lot of unlabeled data are passed through m2. At this time, we then have two classes of data: the initial labeled data, and the soft-labeled data. As only 400 over 2000 images were already segmented, we could end with 5 more times labeled data. Yet, this is in the idyllic situation where every resulting image was annotated at least once by the associated intermediate model. However, the threshold, as well as the model's efficiency, may lead to empty output masks. Anyway, the majority of the unlabeled data are annotated with such an approach. This provides much more data for training a final segmentation model, which aim is to work on the Challenge's test data X_test.

Finally, the final segmentation model M is trained, as the intermediate ones, but with different training data. Here, there may be up to 5 more labeled data, made of the initially annotated ones, and the others, made using the intermediate model predictions. The same validation dataset X_val and Y_val is used. Now that model M is trained, we can pass the test data

X_test through the architecture, and then construct the CSV file to be submitted. We therefore implemented a few-shots learning method adapted to the context of the challenge, to be able to train a segmentation model from a large and consistent dataset. Experiments relating to this approach are presented in Section 3.4.

3 Experiments

3.1 Data

The input is a list of grayscale 2D images (i.e. a 3D numpy array), each corresponding to a CT-scan slice (in the transverse plane) of size 512×512 pixels, and the output has the same shape with integer values (uint8). Each position (w, h) of each matrix $Y_{i,w,h}$ identifies a structure. If 23 areas are segmented, 23 related colors are plotted, so each pixel label $Y_{i,w,h}$ at position (w, h) has values in the interval $[|0, 23|]$. 0 is a special value meaning that this pixel is not part of a structure and is therefore part of the background.

Notice that the slices are mixed, so there is no 3D information. Hence, this problem can be seen as a frame-by-frame pixel clustering problem, where each structure is a cluster within the image. For example, the structure associated with the liver may be mapped to label 4 in one image and 1 in another.

In practice, the output is encoded as a CSV file that encodes the transpose of the flattened label matrix. Note: The transpose is used here for performance reasons: Pandas is very slow to load CSV files with many columns, but is very fast to load CSV files with many rows. Thus, the CSV is made up of $262144 = 512 \times 512$ columns, each corresponding to one image pixel, and 500 rows, each corresponding to one image.

The training set consists of 2,000 images, divided into two groups: 400 with fully segmented structures. For these images, the corresponding output is a 2D matrix with pixel labels for segmented structures and tumors, and the other pixels are set to 0. 1600 of them have no annotation at all. For these images, the corresponding output is filled with zeros. Structure segmentations, in the form of images of the training set, are also given in the supplementary materials. Finally, the test set consists of 500 images with segmented structures and is similar to the unlabeled training data. Considering that an individual image with its label matrix is around 400KB and that we have 1500 images, we then have a dataset of around 600MB in total.

3.2 Pre-processing

The purpose of this section is to highlight the performances of the different pre-processing approaches, to choose the most relevant. This choice is also backed up by the theoretical guarantees of Section 2.2.

The approach is the following: First, outputs are generated from the same image, using the baseline. We then display the results of the proposed segmentations, to demonstrate the effectiveness of pre-processing, starting from a quasi-trivial segmentation method. Finally, we compare the corresponding performances, to select the denoising filter to be adopted.

As a recall, Figure 2 gives the segmentation of one slice, from the baseline, without any pre-processing step before. As detailed in the related section, this segmentation lacks accuracy and is not suitable to be an efficient tool. Then, by selecting appropriate filter parameters, the Bilateral and TV Chambolle methods provide the segmentations of Figures 8 and 9, that better

than the first one.

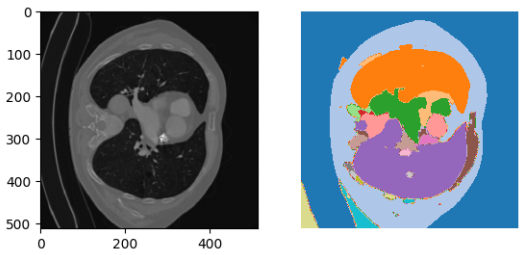


Figure 8: TV Chambolle ($\frac{1}{\lambda} = 0.08$)

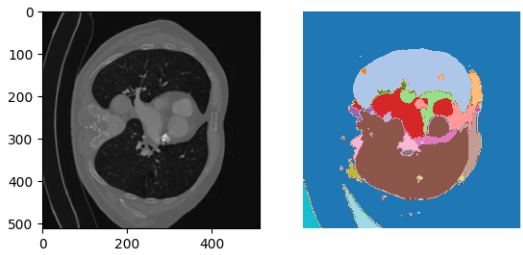


Figure 9: Bilateral ($\sigma_{spatial} = 15$)

In the images tested with these two filters, we note that the same pattern repeats itself, i.e. TV Chambolle, coupled with the baseline, produces better quality segmentations. In addition to showing that a denoising step is relevant to the task posed by the Challenge, this seems to demonstrate the superiority of this filter. To confirm this, the Rand-based evaluation metric is computed for both types of segmentation.

On a few parts of the training data (the 200 first ones), the Rand-based metric is computed, and leads to the results presented in Table 1:

Filter	**Parameters**	**Computation time**	**Score**
Baseline	NaN	0:53	0.137
Baseline + Bilateral	$\sigma_{spatial} = 15$	9:51	0.175
Baseline + TV Chambolle	$\frac{1}{\lambda} = 0.08$	1.42	0.182

Table 1: Rand-based metric for different denoising filters

Remarks can be made about these results. First, the denoising step has be proven to be efficient, not only concerning the visual representation of the images in Figures 8 and 9, but also when it is about the scoring metric. About 0.05 points are gained by only pre-processing the data (contrast enhancing + denoising). Also, computation time is consequently better with the TV Chambolle filter. This echoes the reasoning of Section 2.2 about the complexities of the two approaches.

An additional idea was also tested. The purpose was to aggregate the results of the filters, obtained with different weights, by taking the statistical mode of the predictions for each pixel. The idea behind this is to strengthen the power of segmentation and not forget relevant pixels. Despite its engineering character, this method proved too demanding from a computational point of view, as calculation time is linear with the number of parameters considered. We therefore rejected this approach.

3.3 SAM

To explore more efficient segmentation techniques, we considered the Segment Anything Model in our pipeline. The model is imported from FAIR's repository on GitHub[6]. Due to its ability to generalize segmentations to unknown situations, we first used the pre-trained model, without fine-tuning. To this end, `SamAutomaticMaskGenerator` generates automatic segmentation masks for a given input image. Then we can merge the masks to visualize the automatic segmentation. Figure 10 provides a visualization of the automatic segmentation. The pre-trained SAM appears

[6]SAM repository: https://github.com/facebookresearch/segment-anything.git

to give good results, at least at a glance.

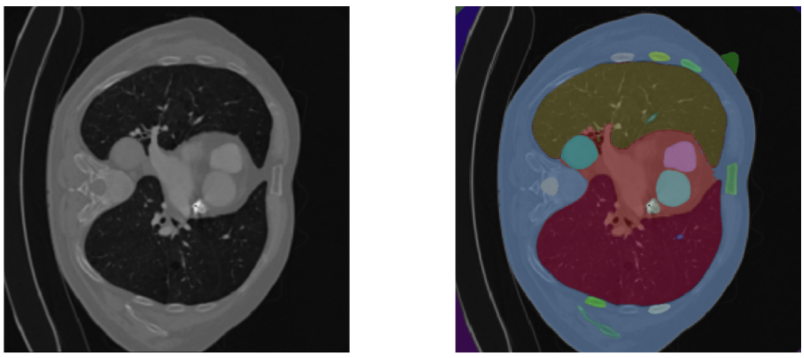


Figure 10: An example of automatic segmentation with SAM

In the same way as for the evaluation of the baseline method, we evaluate the performance of automatic pre-trained SAM on the same validation dataset. Surprisingly, given a pre-processed or raw dataset, the Rand-based evaluation score is about 16%, which is not drastically better than the baseline score, and even worse than the pre-processed baseline. Several explanations can be given. First, we were content to use only a pre-trained model, and therefore to ignore the dataset of labeled training data. Using the training dataset as a whole, i.e. considering a fine-tuning step, seems more relevant to the objectives of the challenge. It should also be noted that biomedical image segmentation requires a high degree of precision, given the organic components to be detected and segmented. Another limitation of this approach is relative to the computational complexity. Indeed, these results, really close to the baseline, were obtained after 6 minutes of running time, on a V100 GPU, which is drastically slower than the baseline. Moreover, an interesting finding was that the pre-processing which helped the baseline worsened the capacities of SAM. One possible explanation could be that SAM might have encountered more, not pre-processed data.

Due to the limitations of the pre-trained approach, and the availability of labeled training data, it was considered to fine-tune SAM, to accurately segment our validation and test data. However, the implementation of SAM does not make the task of fine-tuning easy, since the `SamAutomaticMaskGenerator()` class performs extensive processing internally and operates without utilizing gradients. Moreover, its output is binary and not subject to differentiation, so it is not possible to implement a classic training pipeline (`.train()`). The solution would therefore be to work on the model architecture and directly modify the training weights by adjusting the wrapper model and the loss function, for the mask decoder and so on. We did try this approach and were able to perform a pre-training of Sam, which however yielded bad results and was too long to compute. Given the lack of ergonomics of this solution, it was decided to consider another segmentation model for the future.

3.4 Few-Shots Learning with Detectron2

Several key elements must be specified, to settle the experiments' framework. The first one is the recall of the exact approach used with Detectron2. As the underlying models may be built according to different classes to detect and segment, it was specified in Section 2.5 that we consider only two classes - organs and background/irrelevant zones - to perform the instance segmentation. Yet, one could have chosen to consider multiple classes, corresponding to different organs to segment. Indeed, we can consider that elongated organs are similar to each other, as are circular organs. This leads to a sort of proximity between the labels that would have been

assigned to each class of organ. The idea is inspired by the radiologist's approach, since they may be more efficient when searching for a liver to segment it, than searching for a random organ. Unfortunately, this approach was not suitable in this Challenge, because the data labels lack consistency among the different images (the same organ can be labeled 1 in one image, and 4 in the other). This is the reason why we perform instance segmentation, built from two classes to feed the models, that point to a difference between an organic component and other parts of the image.

Detectron2 provides a variety of model configurations for instance segmentation, each with different backbones, feature extractors, and training schedules. The names of the configuration files typically provide information about the architecture and settings used. More details about the architectures provided in Detectron2 can be found in Annex A.4.

3.4.1 A toy model to adjust the hyper-parameters

To begin with, we decided to start by playing around with the code given by the official code of Detectron2 as the beginners' notebook[7]. The parameters given in this notebook were used as our very first parameters. Especially, we started by using R_50_FPN_3x[8]. Considering the learning rate and the number of epochs, we also started with the values provided in the official tutorial. We then diverged from these values by trying to empirically augment the maximum learning rate and the number of epochs.

3.4.2 Realizing what we should do

Until now, we were not able to use the labeled dataset because we had not figured out what it represents. Whether it was with the U-Net or SAM, we always tried to predict multiple classes. This always yielded in the explosion of the loss. However, stubbing upon Detectron2 and the previous notebook had a significant impact because we realized that the model was always predicting one class, see Figure 11. At first, we thought that there was an imbalance in the dataset, but that was not the case. Ultimately, we realized that we should really consider all organs as identical and just do an object, or instance, detection task without trying to do semantic learning on it. Retrospectively, we realized that we held the assumption that the labels were varying 'not really randomly' for too long. Once we decided to change our point of view on the labels, we could perform better. Indeed, the loss started to decrease. From this, we decided to start and build the entire self-supervised pipeline.

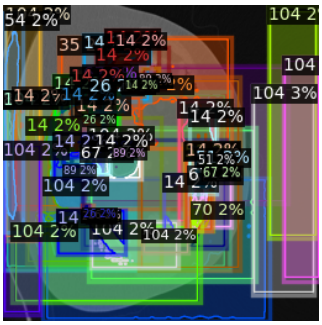


Figure 11: The model was always predicting the same class, not due to an imbalance in the dataset, but rather by the own task itself. We do not care about predicting the class number. This explains the low prediction score also (2 percent for each label).

[7]Detectron2 notebook: https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5

[8]R_50_FPN_3x: https://github.com/facebookresearch/detectron2/blob/main/configs/COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml

3.4.3 Trade-off: who should be more trained ?

To introduce some differences between the models, we played on the maximum learning rate and the number of iterations. For the intermediate models an easy strategy was quickly adopted: find the optimal parameter for the learning rate and the number of epochs, and then make one model that learns quicker, and for a shorter period (this is a radiologist that might be interested in rough features of the organs), and then do the opposite for the third one, smaller learning rate and larger number of iterations. This model is a detail specialist, but might learn too many details!

For the self-supervised pipeline there is one central consideration which is the trade-off between the intermediary models and the final model. The first intuition was just to **follow the PET** Pattern-Exploiting Training paradigm. If we just followed blindly the PET algorithm, one would be tempted to train the large model more than the three intermediary models. However, this yields limited results, as it is depicted in Figure 12. We made the hypothesis that the small models were therefore poisoning the data, making the final model unable to learn anything.

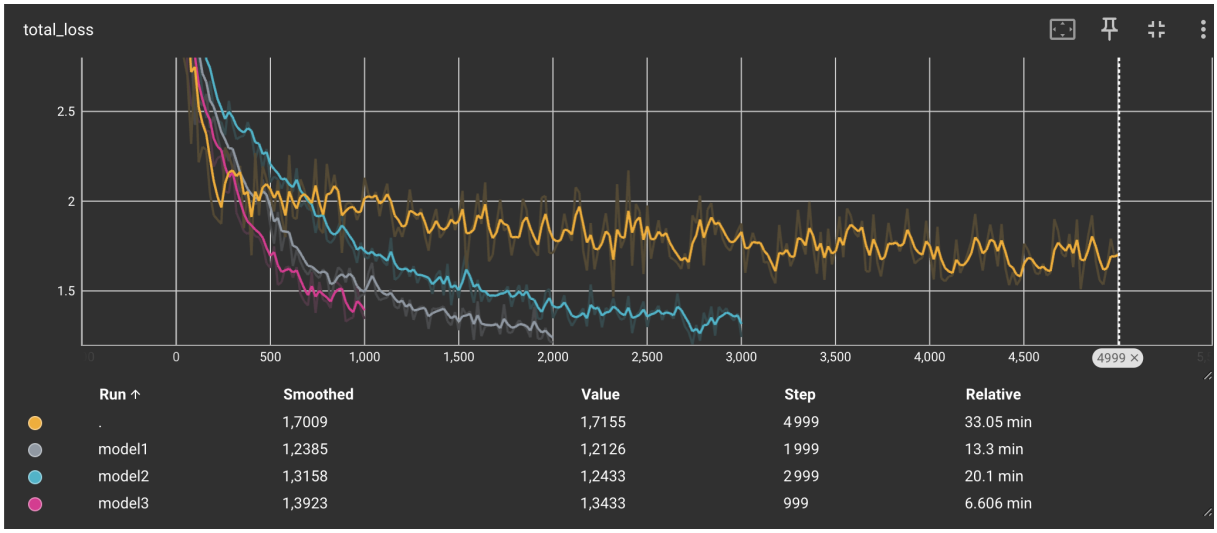
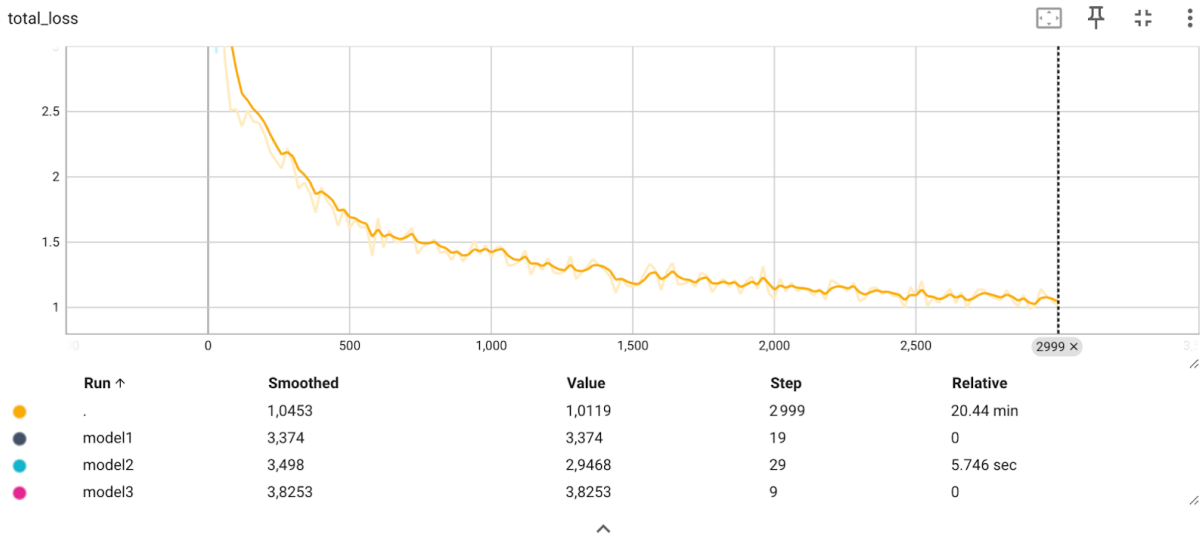


Figure 12: The orange model (the final one) is doing worse than the small models trained on the true data. The small models might be poisoned with the badly soft-labeled data.

Rejection of the small models These findings motivated us to completely suppress the small intermediary models. This could be done artificially through very short training for the intermediary models and then rejecting all their predictions by putting a very high threshold for the labeling. This yielded a very good loss for the final model, see Figure 13.



Run ↑	Smoothed	Value	Step	Relative
.	1,0453	1,0119	2999	20.44 min
model1	3,374	3,374	19	0
model2	3,498	2,9468	29	5.746 sec
model3	3,8253	3,8253	9	0

Figure 13: The intermediary models can be artificially removed by training them on a few epochs and putting the threshold used for soft-labeling the data very high

However, it was having a big problem, which could be detected only by visual inspection. Indeed, the total loss only (which is depicted), also takes into account the bounding box quality of the model (i.e. is it making good bounding boxes). Having a lot of bound boxes (since there are a lot of organs in one scan), a simple, yet deceptive strategy can be implemented by the models which was to only focus on the bounding box and put them everywhere. And this is what happened, as it can be seen in the left panel of Figure 14. On the other hand, small, less trained models would predict some little and irregular shapes, as can be seen in the right panel of Figure 14. This motivated us to create a pipeline where all the models would play a similar role.

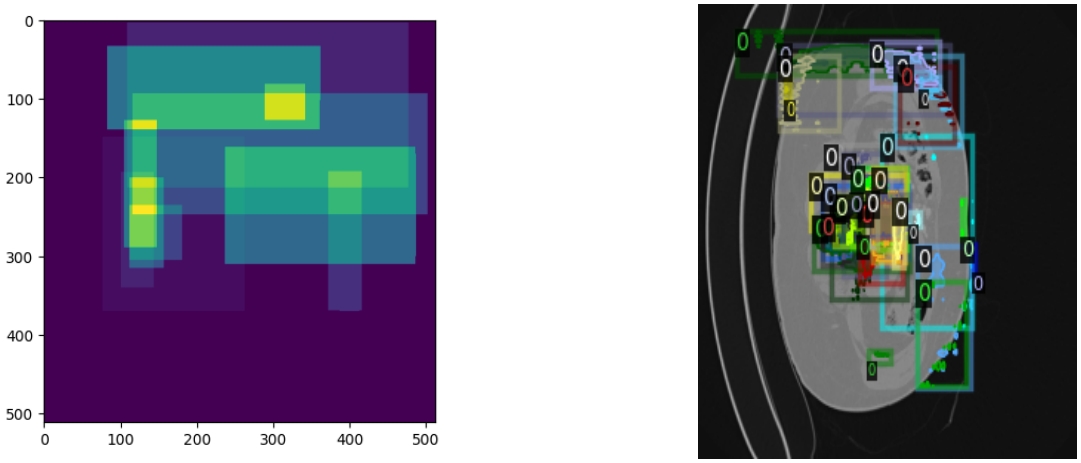


Figure 14: Left: When the big model is left alone, it overfits the bounding boxes. Right: On the contrary, the intermediary models, less overfitted, can detect small irregular shapes and create complementary soft-labeled data. The small models introduce a little, but necessary, noise in the soft-labeled dataset.

All models of different the sizes complement each others. We found a trade-off between the two models and the hyperparameters which yields the best performance that we had with these small models (R_50). Both in terms of loss, see Figure 15, and in terms of predictions, visually inspected in Figure 16.

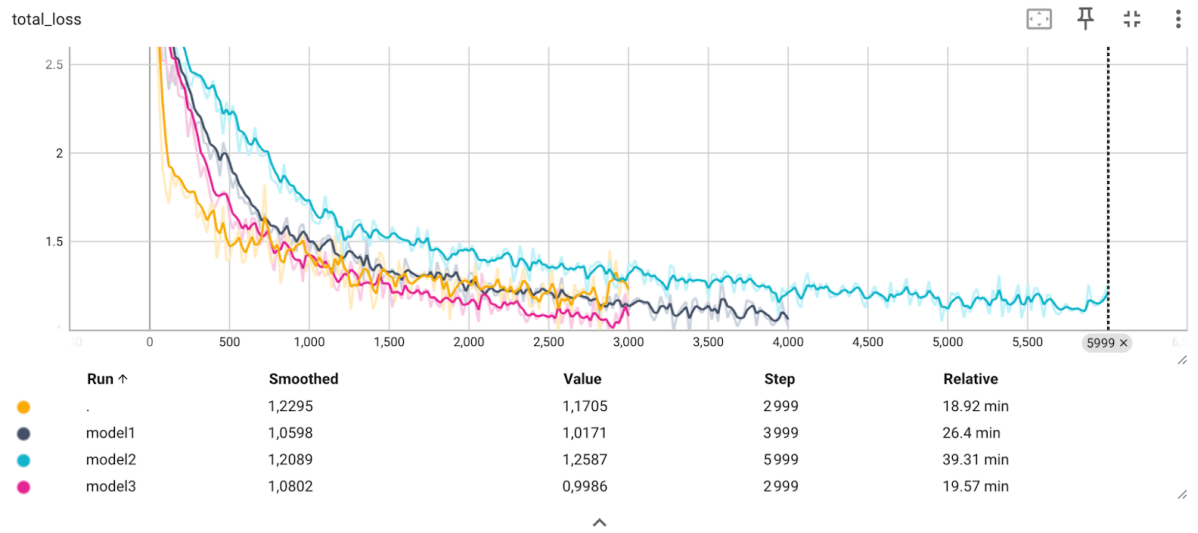


Figure 15: Training "almost" equally the intermediary model and the large model. This yields a good loss, but especially the visual inspection shows us the best prediction so far.

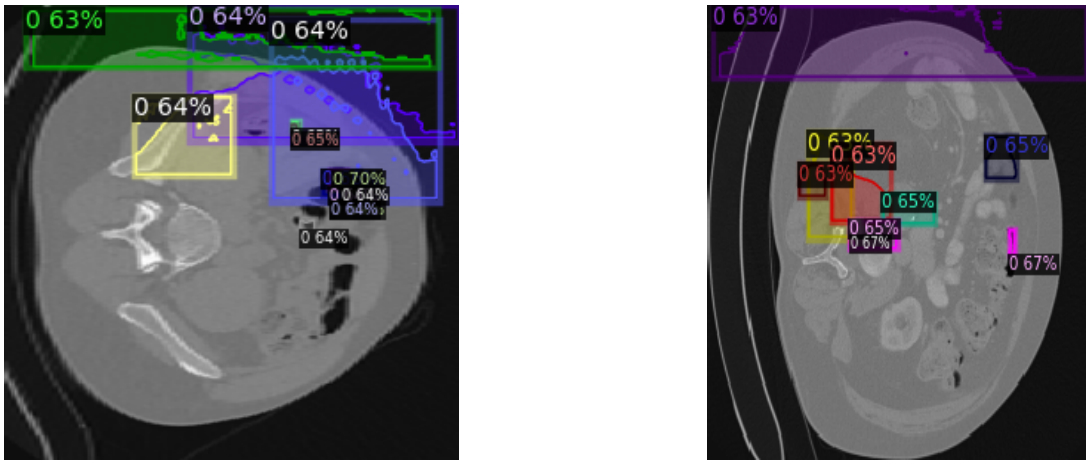


Figure 16: Outputs of the models depicts both the ability to find large areas and not just overfitting on the bounding boxes. However, the segmentations are far from being perfect

Then, we arrived at the limit of the model. In summary, this pipeline enables us to play with 12 hyperparameters. For each model, on top of the learning rate and the number of epochs, the detection threshold had also to be precisely fine-tuned. However, the confidence of the models increased when they were overfitting, especially on the bounding boxes. Therefore, in a final attempt playing with the R_50 models, played with overtrained the small models, and this resulted in the whole pipeline always predicting squares (see Figure 17). Moreover, the final model became disoriented, as can be seen in Figure 18, depicting a final model unable to learn.

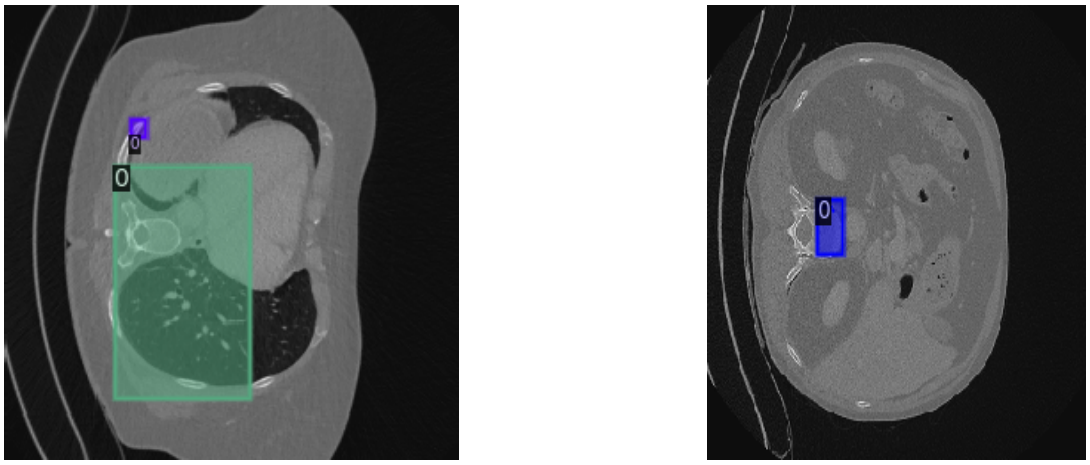


Figure 17: Outputs of the models carry out the overfitting of the intermediary models, as we can see that predictions are mainly in the shape of a square.

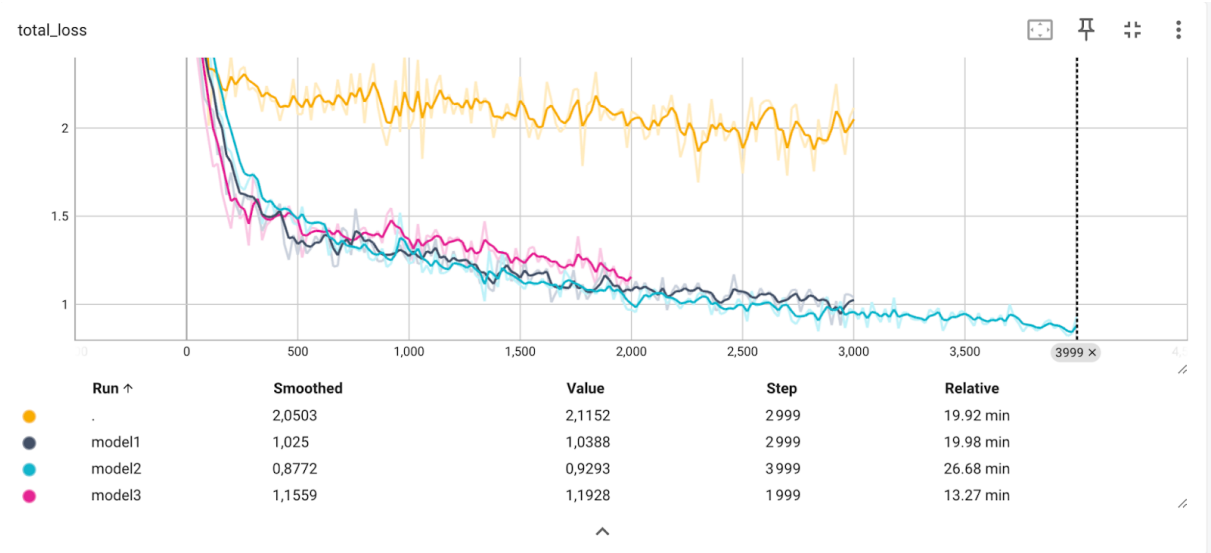


Figure 18: In a final attempt with R_50 we decided to train longer and with a larger learning rate, all the models. However, this yielded the intermediary models to overfit and only predict some bounding boxes. This made the final model unable to learn a real segmentation or anything at all...

This motivated us to completely change the base model, we felt as if we had arrived at the limits of the capacities of the pipeline using these simple R_50 models.

3.5 Using X_101

As it was said before, we felt that we arrived at the limits of the capacities of the R_50. We thus decided to take one of the most complex models present in the Detectron2 library: X_101_32x8d_FPN_3x[9].

Its architecture is detailed in Appendix A.4. However, some key lessons could be drawn from our experimentation with the small R_50 models :

- The intermediary models are as important as the final model;
- Each model should train almost equally. Indeed, the intermediary models add a necessary little noise ;
- The intermediate s can easily poison the data, they should not overfit;
- When overfitting (for instance on the bounding boxes), models exhibit trust in themselves;
- The score threshold for intermediate models should be higher than the one of the final model. It is better to have a few, well-annotated data than to have poisoned data

First, it is possible to see that from the start, using X_101 models helped a lot in the soft-labelling tasks.

The intermediate models can both find large areas and not just square shapes and use that to soft-label the unlabeled data, as can be seen in Figure 19.

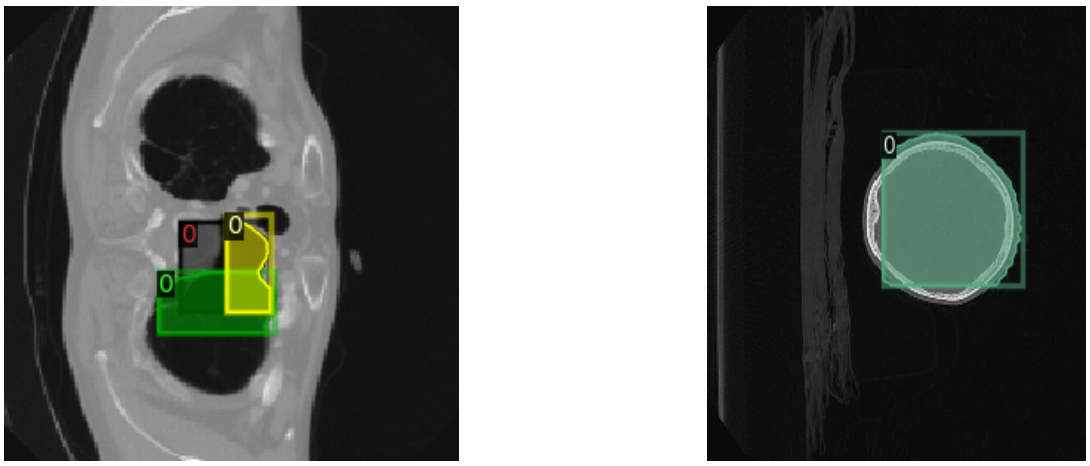


Figure 19: Soft-labelled data by the intermediary X_101 models. Not a lot of annotation remained with our high threshold (0.65) (only 177 data were labeled).

Then we decided to train all models, and due to these good annotations we were confident enough to train a little longer the final model. However, in terms of training time, the X101 is two times longer than the R50 and its loss is better, but not twice better. See Figure 20.

However, the loss is a proxy that can be used badly by our models (notably overfitting by the R_50). Yet, visual inspection gave us a good surprise since it was possible to see that the model was making some "meaningful" segmentations, see Figure 21. We can see that sometimes it still only segments a few elements. Moreover, out of the 500 test data, 102 did not receive any segmentations. Yet, considering that even this good segmentation was not perfect, we

[9]X_101_32x8d_FPN_3x: https://github.com/facebookresearch/detectron2/blob/main/configs/COCO-InstanceSegmentation/mask_rcnn_X_101_32x8d_FPN_3x.yaml

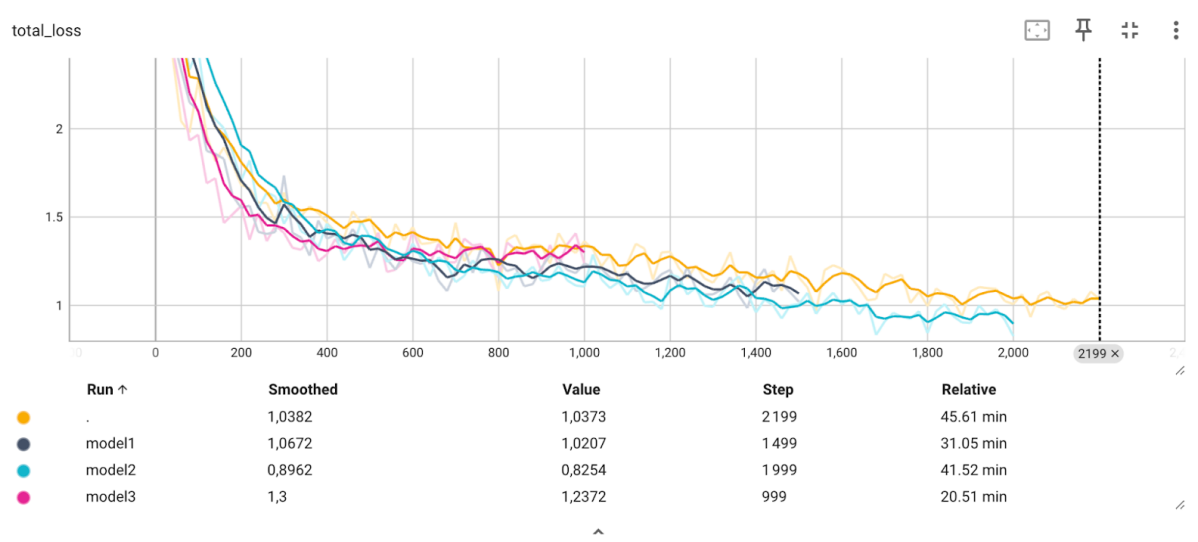


Figure 20: Because we had limited computational units, we decided to decrease all the epochs by half compared to R_50. This yielded the same computational time and provided a better loss. The X_101 approach is very promising.

decided to keep the threshold this high. Moreover, putting it a bit below created some aberrant segmentation, and the signal (some meaningful segmentation) was lost in the noise.

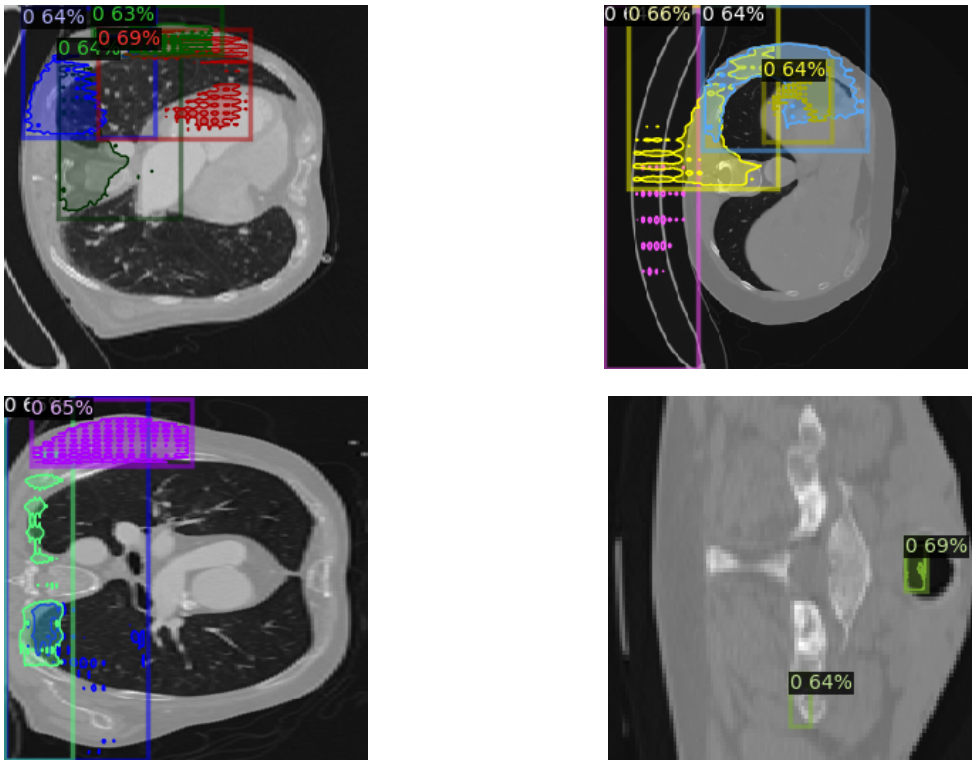


Figure 21: Selected segmentation after the final model. The quality of these segmentations is part of the normal, even good, segmentations made by the model. The threshold was set to 0.63 after a visual inspection leveraging the validation dataset.

3.6 The last challenge of the Challenge

However, there was still something missing. Indeed, what is possible to see is that these segmentations were in the COCO format, different from the CSV output required by the challenge. If with the baseline it was not possible to overlap masks, it can be seen in the previous predictions that it was the case. The question was, then, how do we deal with the overlapping data? Our very first approach was just to add to the pixel contained in the mask a unique identifier (nothing more than just an increasing number for each mask). Indeed, the evaluation metric of the challenge was the rand index, which did not care about permutation or the label of each class. However, just adding the masks like that gave rise to a new problem, which was the creation of a new class for each of the pixels at the intersection of the two masks. This was artificially creating new organs and segmentations, which is something that we did not wish for.

The option that we finally chose was to keep, at the place of intersection between two masks, only the mask which had a higher score, preventing us from creating a new label and artificial segmentations. A prediction on the test dataset with the masks coming from the final .csv file can be visualized in Figure 22.

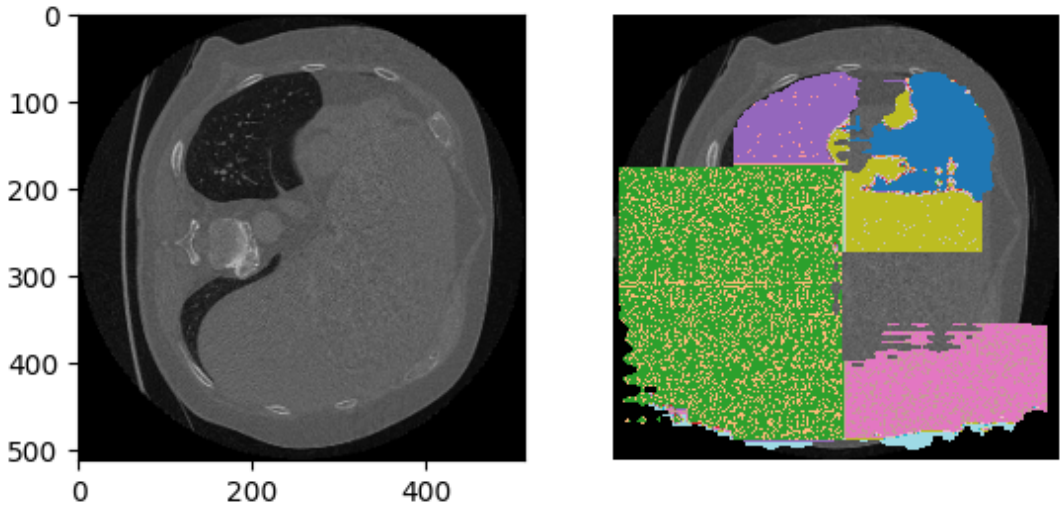


Figure 22: Masks could not be overlapped in the final .csv file of submission. When segmentations were overlapping each other, we only kept the one with a higher score. However, as can be seen, this yields some remaining problems. Indeed, the 'green' segmentation, which resembles a square, the type of shape that appeared also when overfitting, and with high confidence, did crop the more organically shaped purple segmentation.

4 Conclusion

During this challenge, we had thus the opportunity to learn more about image pre-processing techniques and showed how much they can improve the performance. Then we tried to train both the canonical Neural Network of medical segmentation (U-Net) and the State-of-the-Art algorithm, SAM. However, it was not before using a more handy model, Detection and Detectron2, which had a lot of documentation, that we realized that we were doing the task completely wrong. If we were trying to do a label detection task, it was necessary to completely change the data and one should consider the challenge as a foreground-background detection task, not as a classic label prediction task. Once we realized that, we decided to continue with the library, which we found quite handy given the large amount of different pre-trained models implemented in it.

However, in total, this challenge cost us 30€, which is already a big sum. However, the last architecture that we were able to create and that we made from scratch is very promising since it has a lot of hyperparameters (12) (learning rate, iterations, thresholds for each of the four models) to tune and is flexible to a lot of models. By changing just one sentence in our notebook, it is possible to change the architecture of the model, this means 330 different architectures can be implemented by just changing a few lines in our code. On top of that, it might be possible to use different predefined model weights, on some different datasets than the one given by the initial Facebook release.

Finally, we are taking a lot of positives from the Challenge experience. None of us (Mathias and Lucas) had ever taken a course or carried out a project in computer vision before. The Data Challenge offered by Raidium seemed to us a good opportunity to explore this field, but also to learn a lot about the strengths and limitations of the models used, as well as the good practices to adopt in image processing. In this context, we faced both technical and practical difficulties. Nevertheless, we put all our energy into not only coming up with a better solution than the baseline, but also proposing relevant solutions and taking a step back from them. This experience and the lessons learned will undoubtedly serve us well in the future.

References

[1] A. Kolesnikov D. Weissenborn X. Zhai T. Unterthiner A. Dosovitskiy, L. Beyer and G. Heigold S. Gelly J. Uszkoreit N. Houlsby M. Dehghani, M. Minderer. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[2] N. Ravi H. Mao C. Rolland L. Gustafson T. Xiao S. Whitehead A. C. Berg W-Y Lo P. Dollár R Girshick A. Kirillov, E. Mintun. Segment anything, 2023.

[3] C. Hallacy A. Ramesh G. Goh S. Agarwal G. Sastry A. Askell P. Mishkin J. Clark G. Krueger I. Sutskever A. Radford, J. Wook Kim. Learning transferable visual models from natural language supervision, 2021.

[4] Serge Beucher and Centre Mathmatique. The watershed transformation applied to image segmentation. *Scanning. Microsc.*, 6, 2000.

[5] A. Chambolle. An algorithm for total variation minimization and application. *Journal of Mathematical Imaging and Vision*, 20:89–97, 2004.

[6] S.-A. Ahmadi F. Milletari, N. Navab. V-net: Fully convolutional neural networks for volumetric medical image segmentation, 2016.

[7] Abner Guzmán-rivera, Dhruv Batra, and Pushmeet Kohli. Multiple choice learning: Learning to produce multiple structured outputs. In L. Bottou K.Q. Weinberger F. Pereira, C.J. Burges, editor, *Advances in Neural Information Processing Systems*, volume 25, 2012.

[8] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 1985.

[9] S. Ren J. Sun K. He, X. Zhang. Deep residual learning for image recognition, 2015.

[10] S. Xie Y. Li P. Dollár R. Girshick K. He, X. Chen. Masked autoencoders are scalable vision learners, 2021.

[11] N. Kanopoulos, N. Vasanthavada, and R.L. Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of Solid-State Circuits*, 23(2):358–367, 1988.

[12] S.M. Pizer, R.E. Johnston, J.P. Ericksen, B.C. Yankaskas, and K.E. Muller. Contrast-limited adaptive histogram equalization: speed and effectiveness. *Proceedings of the First Conference on Visualization in Biomedical Computing*, pages 337–345, 1990.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[14] Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, 2021.

[15] Timo Schick and Hinrich Schütze. It's not just size that matters: Small language models are also few-shot learners, 2021.

[16] S. Belongie L. Bourdev R. Girshick J. Hays P. Perona D. Ramanan C. Lawrence Zitnick P. Dollár T. Lin, M. Maire. Microsoft coco: Common objects in context, 2015.

[17] R. Girshick K. He P. Dollár T.-Y. Lin, P. Goyal. Focal loss for dense object detection, 2018.

[18] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, 1998.

A Theoretical details of the tools

The purpose of this annex is to provide the mathematical details about the construction of the different tools involved in our work. While the content of the report focuses on their use case for segmentation, they are here introduced and explained.

A.1 Sobel filter

Sobel filter is built utilizing Sobel operator [11]. Sobel operator calculates the intensity gradient of each pixel. This indicates the direction of greatest change from light to dark, as well as the rate of change in this direction. We then know the points of sudden change in brightness, probably corresponding to edges, as well as the orientation of these edges.

The operator uses convolution matrices. The 3×3 matrix is convoluted with the image to calculate approximations of the horizontal and vertical derivatives. Consider the source image A, and two images G_x and G_y which at each point contain approximations of the horizontal and vertical derivative of each point respectively. These images are calculated as follows:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

At each point, the approximations of the horizontal and vertical gradients can be combined as follows to obtain an approximation of the gradient norm:

$$G = \sqrt{G_x^2 + G_y^2}$$

The direction of the gradient can also be calculated as follows:

$$\Theta = atan2(G_y, G_x)$$

where, for example, Θ is 0 for a darker vertical contour on the left.

A.2 Bilateral filter

Bilateral filter [18] is a denoising filter. It maintains edges while averaging pixels, taking into account their spatial proximity and similarity in color intensity. The measure of spatial proximity is determined by the Gaussian function applied to the Euclidean distance between two pixels, with a predefined standard deviation. Importantly, the calculation of weights takes into account both the Euclidean distance between pixels and the radiometric variations (for instance, differences in color intensity, depth, etc.). This method ensures that sharp edges are maintained.

The filter can be defined as:

$$I_{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

with the normalization term:

$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

where I^{filtered} is the filtered image, I is the original input image to be filtered, x are the coordinates of the current pixel to be filtered, Ω is the window-centered in x, f_r is the range kernel for smoothing differences in intensities and g_s is the spatial kernel for smoothing differences in

coordinates (f_r and g_s can be Gaussian functions).

The weight W_p is determined by considering both the proximity of pixels (calculated using the spatial kernel g_s) and the similarity in their intensity values (calculated using the range kernel f_r). For instance, when aiming to denoise a pixel at location (i, j) in the image, the contribution of a neighboring pixel at location (k, l) is weighted. If Gaussian functions are used for both the range and spatial kernels, the weight for the pixel (k, l) in the process of denoising the pixel (i, j) is calculated accordingly with:

$$w(i, j, k, l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|I(i, j) - I(k, l)\|^2}{2\sigma_r^2}\right)$$

where $I(k, l)$ are pixel intensities and σ_d, σ_r are smoothing parameters.

Finally, the denoised intensity of the pixel (i, j) is given by:

$$I_D(i, j) = \frac{\sum_{k,l} I(k, l) w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)}$$

A.3 TV Chambolle filter

The TV Chambolle filter [5], of Total Variation filter, is based on the principle that signals containing too much or unwarranted detail exhibit a high total variation, meaning the sum of the absolute gradients across the image is large. In line with this concept, lessening the signal's total variation—while ensuring it still closely resembles the initial signal—eliminates unnecessary detail yet retains crucial features like edges. As the bilateral one, TV filter aims at reducing the noise, while keeping the sharp edges of the image.

Considering f as a noisy image (the input data), the objective of total variation denoising, which is also referred to as total variation regularization, is to identify an image u that possesses a lower total variation than f, while being constrained to bear a resemblance to f. This objective is encapsulated by the discrete Rudin–Osher–Fatemi (ROF) minimization problem:

$$\min_u \sum_{i=0}^{N-1} \left(|\nabla u_i| + \frac{\lambda}{2}(f_i - u_i)^2\right)$$

where λ is a positive parameter. The first portion of this cost function denotes the total variation, while the second part signifies data fidelity. When λ increases towards infinity, the total variation component prevails, compelling the solution to exhibit a reduced total variation, which may lead to a result that does not as closely resemble the original input data.

One way to solve this problem is implemented in Scitkit-Learn and respects Chambolle's approach. At each stage of the iterative loop, pixels are stored, along with their associated gradients and divergences. Then the gradient vectors are optimized, so that the cost function decreases until the difference reaches a pre-defined threshold.

A.4 The different models in Detectron2

- 'R' or 'X': This indicates the type of ResNet used. 'R' stands for ResNet, while 'X' stands for ResNeXt, which is a more advanced version of ResNet with grouped convolutions for increased model capacity.

- ‘101‘, ‘50‘, etc.: These numbers indicate the depth of the backbone network, specifically the number of layers. For example, ‘R_101‘ means ResNet-101 with 101 layers, while ‘R_50‘ means ResNet_50 with 50 layers.

- ‘C4‘, ‘DC5‘, ‘FPN‘: These refer to the feature extractor used in conjunction with the backbone. - ‘C4‘ means that the 4th stage of the ResNet is used as the feature extractor, without any FPN (Feature Pyramid Network). - ‘DC5‘ means that dilated convolutions are used in the 5th stage of the ResNet, which increases the receptive field. - ‘FPN‘ indicates that a Feature Pyramid Network is used, which combines low-resolution, semantically strong features with high-resolution, semantically weak features to generate a rich multiscale feature pyramid.

- ‘3x‘, ‘1x‘, etc.: This usually refers to the training schedule. ‘3x‘ means the model is trained for approximately 3x longer than the default 1x schedule. A longer training schedule can potentially lead to better performance as the model has more iterations to learn from the data.

- ‘giou‘: This indicates the use of Generalized Intersection over Union (GIoU) as a loss function for bounding box regression, which can lead to better performance compared to the traditional IoU loss, especially in cases where the predicted and target boxes do not overlap.

- ‘regnetx‘, ‘regnety‘: These are names of different backbone networks from the RegNet family, which are designed using a design space exploration approach to find network architectures that provide a good trade-off between efficiency and accuracy.

- ‘32x8d‘: In the case of ResNeXt backbones (indicated by ‘X‘), this refers to the configuration of the network in terms of the number of groups (32) and the width of each group (8d).

- ‘4gf‘: This is related to RegNet backbones and stands for 4 Giga Flops, indicating the computational complexity of the model.