ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Computer Vision

CVAT MEETS TRANSFORMERS: ACCELERATING SEMANTIC SEGMENTATION LABELING IN INDUSTRIAL APPLICATIONS

CANDIDATE

Edoardo Procino

SUPERVISOR

Prof. Luigi Di Stefano

CO-SUPERVISOR

Dott. Giuseppe Casadio

Academic year 2022-2023

Session 5st

Contents

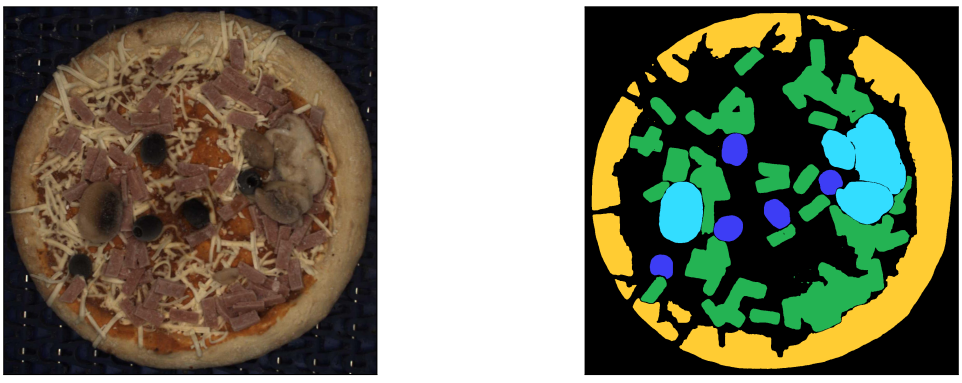
List of Figures

List of Tables

Introduction

This thesis is the description of a project developed at SpecialVideo [22] during my internship. SpecialVideo is a company based in Imola which is specialized in building industrial vision systems like for food product quality or to guide a robotic arm.

In many tasks required by SpecialVideo's clients, it is useful to perform the *semantic segmentation task*, namely the classification of each pixel into a class. This turns to be a powerful ally since, thanks to the classification of each pixel,it is then a lot easier perform the blob analysis of specific parts of the picture. To be more clear, let's do an example: consider the pizza in the figure 1 (a), if one wants to count the olives or to compute the area of the pizza covered by ham, the better way is to isolate the single ingredients to simplify the downstream computation. This can be done with the segmentation (In 1 (b) there are the ingredients divided with 3 different colours and the crust colored in yellow).



(a) (b)

Figure 1: Example of a pizza and its ground truth segmentation mask

The segmentation is quite hard to perform, especially it is difficult to obtain an algorithm that can generalize along different images. A solution could be the use of neural networks (§0.2) which permits to have a software that generalize well along different images, but they have the drawback that they need labeled images to learn, in particular in the case of

segmentation they need a segmented image (like the one in the figure 1 (b)) for each image used in the training phase. Normally the labels are constructed by humans, but as one can imagine it is a long process since each pixel of the interesting regions inside the image must be colored according to its label.

To speed up this process, before my internship, in SpecialVideo they used CVAT [4], an online tool designed to label images. Thanks to some useful technology like SAM (§1.2) the use of CVAT speeded up the labeling process, but not to an acceptable level: for the company the labeling was still an expensive task from both an economic and timely point of view.

0.1 Description of the problem

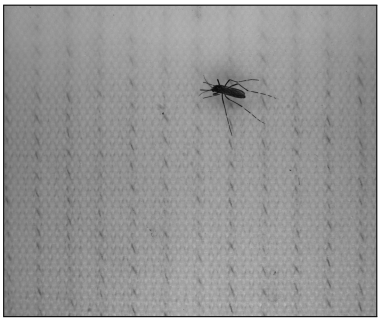
Given the premises in the former chapter, the aim of my internship, and thus of my thesis, was to build a semi automatic pipeline to further speed up the labeling process. Usually clients come to SpecialVideo with few, not labelled, images and the problem is the domain of those images: they represent very specific subjects like wheel rims in the middle of the production (figure 2) or the required task is very difficult like distinguish between male and female mosquitoes (figure 3). Those problems make impossible to directly use pre-trained neural networks to segment the images, since they do not have such domain specific knowledge, thus, fine-tuning is required. From those premises borned my project with the aims to speed up the labeling process using less human time and also with the idea to have a flexible system (i.e.: easy to transfer to new or similar tasks).

(a) (b)

Figure 2: Example of some images from the car alloy wheels task

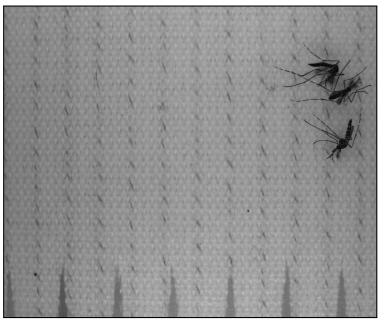
In particular, this work will focus on the segmentation of the toppings of frozen pizzas (like in figure 1); being more precise, we chose two different kind pizza: the first one

(a) Example of famale mosquito



(b) Example of famale mosquito

(c) Example of male mosquito



(d) Example of male mosquito

Figure 3: Example of some images from the mosquitoes task

with black olives and anchovies while the second one with salami, red peppers and yellow peppers. This choice was done for three different reasons:

1. When I started the project SpecialVideo already has the ground truth of those pizzas;
2. The combination of the two formats (i.e.: the one with olives and the one with salami) represents a challenging set; for example the black olives some times are similar to some burns in the pizza crust and also we have some red pepper which are camouflaged if placed over a slice of salami;
3. It is possible to test the transfer capacity of the network on new pizzas with a subset of the training ingredients since the company already have the segmentation of the ”capricciosa” pizza (diced ham, **black olives** and mushrooms).

As better described in chapter 1, we want to improve the segmentation pipeline using few labelled images to train a network which then can automatically segments the images leaving to the labeler only the task to refine the automatic segmentation. Furthermore, our aim is to make an auto-improving system: once the labeler has adjusted the automatically

labelled images, the network can be fine tuned another time to improve its performance and leaving to the labeler even less work for the next batch of images.

Before going in the deep with the details of my work in the following two chapters there will be a short introduction to the main techniques used in the field of machine learning for computer vision, since to my mind is very useful to better understand the following chapters without weigh them down.

0.2 A brief introduction to computer vision's neural networks

In the computer vision field, around ten years ago, thanks to AlexNet [11], the deep learning joined the game imposing itself as one of the main techniques, hitting state-of-the-art results in image classification, object detection and semantic segmentation.

0.2.1 Convolutional Neural Networks (CNN)

The first successful approach was the use of CNN. These networks, differently from the classical Fully Connected Networks (FCN), can process the images better in terms of both time and performances. The time improvement is given by the fact that in CNNs we process an image learning a kernel and sliding it across the whole image and this permits to use way few parameters. For example, if we have as input an image of size $224 * 224$ pixels, using a FCN with 1000 neurons in the firs layer, the network will have to learn $224 * 224 * 1000 = 50,176,000$ parameters, using a CNN with 1000 kernels of size 3x3, the parameters will be only 9000.

Writing about the performance improvement, the CNNs have some advantages which help them to be particularly good in the image related tasks:

- They do not require to flatten the image preserving so its spatial structure;
- Each output unit is correlated to local input units: they (rightly) suppose that neighbouring pixels are more correlated;
- Each output unit of a layer shares the kernel parameter with all the other output units, in this way, we have the same detector regardless the position.

Thanks to the last point, CNNs are also more *data efficient* if compared with linear layers since they do not need to see features in all locations to be able to learn how to detect them.

Furthermore, usually more than one kernel is used in each layer where each kernel has the same number of channels as its input and produces one output (figure 4). The outputs are then concatenated, in this way at each level the network can learn to detect different characteristics of the input making possible at the end to have a semantically rich output.

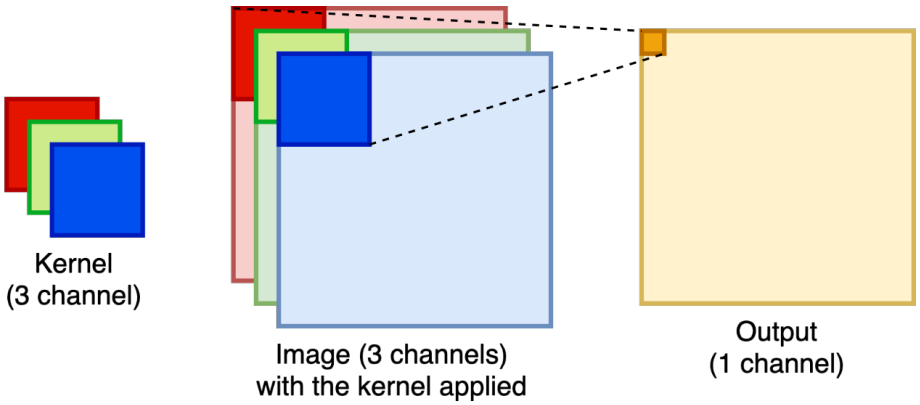


Figure 4: Example of a 3-channels kernel applied to a 3-channels image. The produced output has one dimension.

But, what is a convolution? In the field of image processing a convolution can be described as the process of making a weighted sum of the neighbours of a pixel, using the kernel's values as weights (a graphical representation is reported in the image 5).

$$\begin{bmatrix} 12 & 2 & 8 & 4 \\ 0 & 2 & 7 & 10 \\ 10 & 5 & 7 & 10 \\ 0 & 3 & 6 & 12 \end{bmatrix} \times \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 8 \\ 0 & 0 \end{bmatrix}$$

Concluding, a CNN can be described as a stack of layers spaced out by a non-linear activation function – which prevents the collapse of the layers in a linear classifier – where each layer detects multiple features thanks to the use of different kernels. At the end, the output – the so called *image embedding* – can be used in various way depending on the task:

- The raw embedding can be used to cluster the images based on the similarity, this technique is known as *metric learning*;
- The embedding can be projected in a vector subspace with a number of dimension

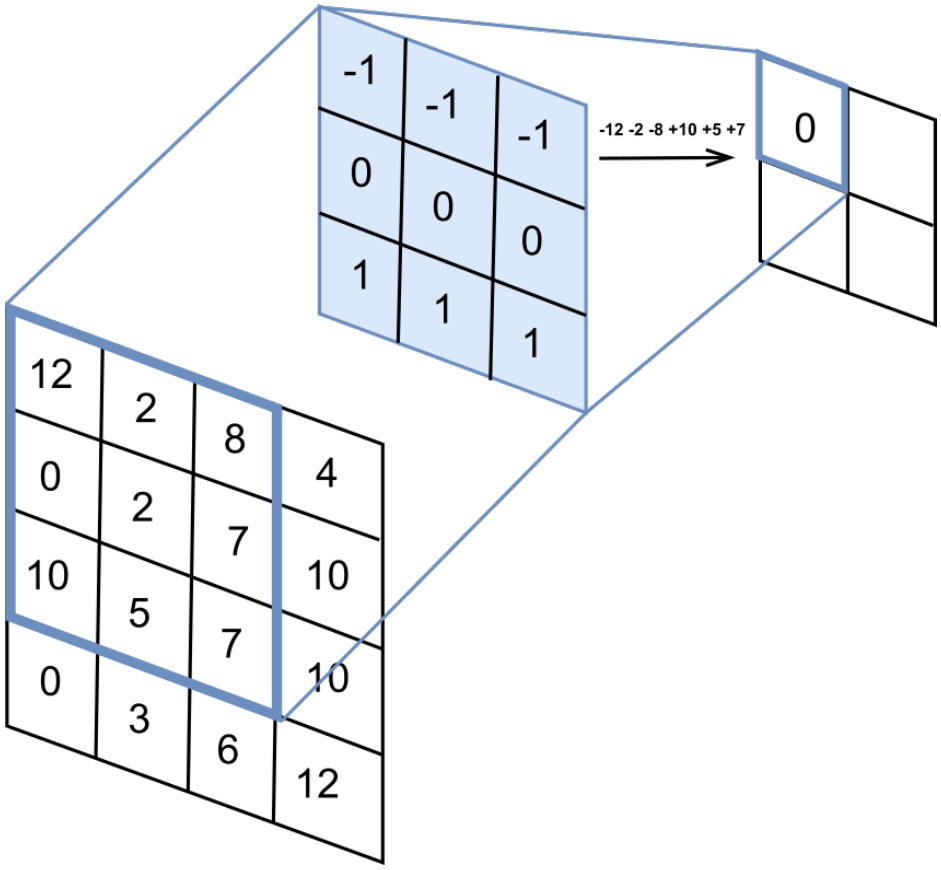


Figure 5: Convolution of a 4x4 matrix with a 3x3 kernel.

equal to the number of classes to be detected and through a Softmax function ($Probability(x_i) = \frac{e^{(x_i)}}{\sum_j e^{(x_j)}}$) a distribution of probability among the possible classes is computed;

- In addition to the classes, also some bounding box coordinate can be given in output for the object detection task;

- An image, the *segmentation mask*, will be the output in case of semantic segmentation.

0.2.2 Vision Transformers (ViT)

During the following years, a lot of improvements were done to AlexNet, for example it is worth mentioning like to mention Inception, by Google [23], the first network that implemented a stem layer (useful to shrink the image in a faster way reducing the cost of the subsequent processing), ResNet [9], which introduced the skip connections (this technique permits to the networks to be able to learn an approximation of the identity function making the training of deeper networks more stable and more effective) and YOLO [20], which is

a fast one-stage object detector (see 2.2.1 for a detailed description of YOLOv8). Then, in 2020, Dosovitskiy at all. [7] proposed the **Vi**sion **T**ransformer (ViT), a slightly modification of the original transformer architecture [25] used in Natural Language Processing, which achieved comparable results with the 2020's best CNNs while it is requirements in terms of resources for the training were fewer thanks to parallelization of the computations.

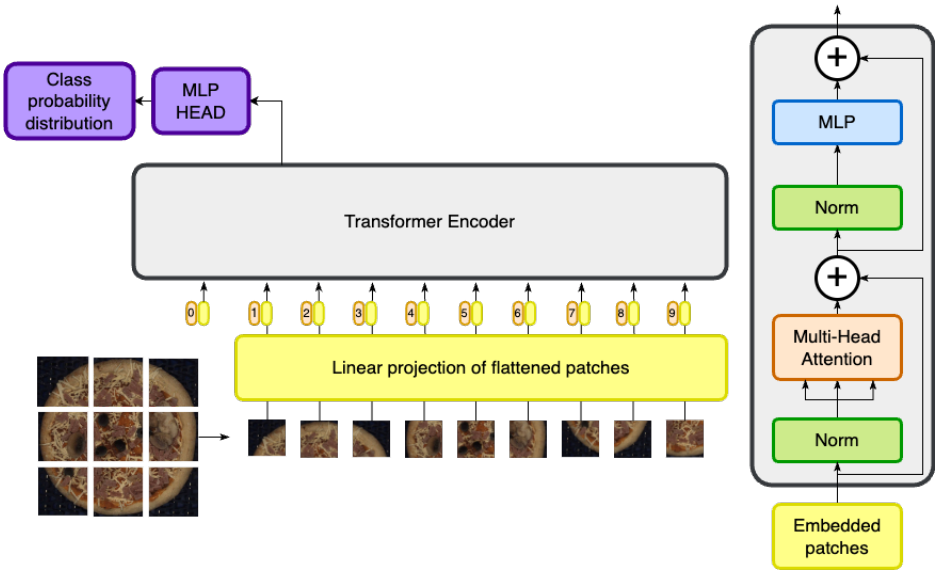


Figure 6: Schema of a Vision Transformer.

Writing about the original transformer architecture [25], they takes in input words – called tokens – and through an encoder-decoder architecture they produce the output. The detailed description of the original transformer block will not be explained here since it is slightly different from the ViT's one and it was not used in this work, but it was important to mention since without it likely nowadays not even ViT would exist. The Vision Transformer uses non-overlapping image patches as tokens, the choice to use single pixels was discarded due to time complexity problems, which are flattened and then projected in a, usually smaller, space and merged with a positional encoding to preserve spatial information, before being feeded to a series of transformer encoders which work exactly like a normal transformer.

The most fascinating module in the ViT's architecture is undoubtedly the *Multi-Head Attention*. This layer works as follow:

1. For each embedded patch three vectors are created multiplying the embeddings with three trainable matrices: a *query* matrix, a *key* matrix and a *value* matrix;

2. At the end of the first step the result consists in three vectors for each embedding and

those vectors are now used to compute a score between each patch and the others. In particular, the dot product between the query vector of the current patch with the key vector of each of the other patches is used (the attention with itself is also computed);

3. The scores are then divided by the square root of the length of the key vector to achieve a more stable gradient and the results are passed to a Softmax function to have a probability distribution which indicates how much the current patch is correlated with all the other patches;

4. At the end, each value vector is multiplied with the Softmax's output to highlight the patches to focus on and the weighted value vectors are summed together.

The thing described above is a normal *attention head*, but both ViT and text transformers work with *Multi-Head Attention*. Basically, to give to the architecture more ability in encoding relationships and little nuances, the Query, Key and Value matrices are logically splitted and passed independently to a different attention head. At the end, thanks to a reshaping of the output, the results of the various head are merged. Trying to explain in a simpler way the concept, the multi-head attention permit to an encoder to use separate sections of an embedding to learn different aspects of the meanings of each token, based on the correlation with the other tokens.

Chapter 1

Initial situation: semi automatic annotation using Segment Anything

In this first chapter the initial setup (the one that SpecialVideo used before my arrive in the company) will be described, while the following chapters contain an analysis of the improvements resulted from this work.

1.1 CVAT: Computer Vision Annotation Tool

As the name suggests, CVAT [4] is an online tool developed by OpenCV [15] with the aim to help people to create labels for datasets with images. The online tool provide various way to annotate a dataset:

- Manual annotation: The user have to manually annotate all the objects through a ”brush” and a ”rubber” (semantic segmentation), building polygons using points (semantic segmentation) or drawing a rectangle (object detection);
- Semi-automatic annotation: The user can use the ***S**egment **A**nything **M**odel (SAM)* (§1.2) with points as prompt to segment each object in a faster way;
- Automatic annotation: The user just select the model and CVAT perform an inference on all the selected images annotating them.

Unfortunately, automatic annotation have a lot of problems: CVAT gives only yolov5 for object detection and since it is not fine tunable the possible labels are only the ones on

which YOLO was trained on and – especially for very specific task – this limitation makes impossible to use the automatic annotation in industrial tasks.

For this reason SpecialVideo usually paid a person (or more) – the *labeler(s)* – only to label images using SAM on CVAT which is very expensive in terms of money and time. The aim of this work now should be more clear: the fine tune of some models and their upload on CVAT can help the labeler in his/her task: instead of starting from scratch each time, the neural network could provide a very good starting point and so the work of the label will be only a check and correct task. Obviously, once the labeler have checked the inferred masks, they can be used to further improve the network.

1.2 Segment Anything Model

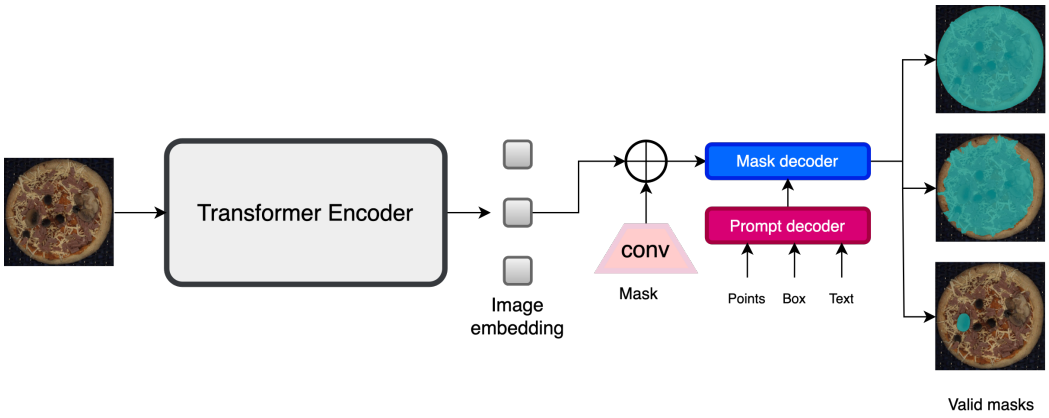


Figure 1.1: Structure of SAM.

For the comprehension of this work, it is worth to spend few pages writing about SAM since it is our starting point and also it will be used in the next chapter.

SAM [10] (figure 1.1) is a deep neural network designed to segment objects with the possibility to use different prompts based on the user preference. In particular one can choose between:

- Points: the user can use some points to select the interested object. A point can be ”positive” which tell to the network to focus on the specified area, or ”negative”, to tell to the network to avoid the area around;
- Bounding box: if a bounding box is passed to SAM the network will focus to the object inside the box;

- Text: thanks to a deep semantic knowledge the user can specify the object s/he wants to segment using words (note: as reported in the official web page ”Text prompts are explored in our paper but the capability is not released” [21]);
- Mask: using a mask as input along with other prompt type permits to generate a better mask which is useful to refine a dataset;
- Fully automatic annotation: the modalities described previously require an active participation of the user, but SAM can also segment the whole image without interactions. Once the user have chose the number of points (e.g.: 32), the model equally distribute them along the surface of the image and segmenting all the objects (figure 1.2).

Even if the last point seems very useful it has some non-trivial drawbacks:

- It does not assign labels (no version of SAM do that, but using key-points as prompt on CVAT permits the label pairing);
- It segments all objects like if each one was in a different class;
- Some unwanted objects could be detected, like the strips of mozzarella cheese.

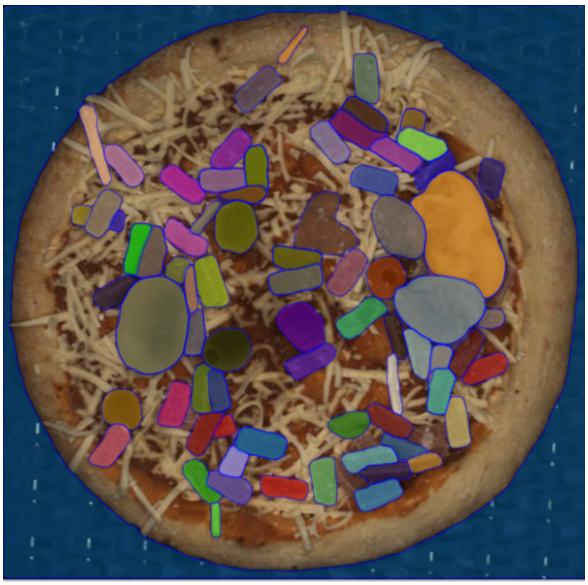


Figure 1.2: A pizza segmented using fully automatic SAM.

Besides those problems, SAM is a very powerful segmentation model and in this work has been used with both key-points and bounding boxes as prompts.

Following, I will explain how SAM is able – as the name suggests – to segment anything using different prompts going through the model structure and its training procedure.

1.2.1 An in-depth analysis of SAM structure

The idea beyond SAM was to build a *foundation model* for the segmentation task. To achieve this aim the idea was to slightly modify the problem into a promptable segmentation task. The prompt (point(s), bounding box or text) is the key to say to the model which part of the image one wants to segment, furthermore, to be sure that each prompt will generate a valid mask, the model returns three masks for each prompt (see the outputs shown in figure 1.1) helping also to avoid missed segmentations for ambiguous prompts.

The structure of SAM (reported in figure 1.1) is as simple as powerful, since it is composed by only three main parts:

1. Transformer encoder: It process the whole input image only once to save time and outputs the image embedding. It is based on the MAE pre-trained vision transformer which is a modification of the original ViT (figure 6) which mask some patches of the input image with the aim to reconstruct the input achieving a self-supervised model that achieve high generalization in downstream tasks and can be trained very fast if compared with normal approaches [8].

2. Prompt encoder: The prompts are of two kind:

   - **Sparse prompts**, like bounding box, points and text. While the points and the boxes are used like *positional encodings* and thus summed with the learned embeddings of the corresponding prompt-type, the text is processed by CLIP [19] which is trained on couples (image, image_description) and thus it has a powerful knowledge of the semantics that connect images and text.
   - **Dense prompts**, the masks, which are processed by convolutions and then the generated embedding is summed up with the image is one.

3. Mask decoder: Finally, the decoder uses both the image and prompt embeddings to generate a mask, which is the output of the model.

Training losses: focal loss and dice loss

Writing about the training, SAM was trained using both a *focal loss* and a *dice loss*.

Focal Loss The focal loss was introduced for the first time by Facebook AI Research team in the paper *Focal loss for dense object detection* [12] to develop a powerful one stage object detector called *RetinaNet*. The aim of this loss is to down weight the loss value for easy examples to make the network focus on hard examples: we want to avoid the network to focus on examples that it already can recognize well but, since they are a lot, the sum of their loss can be greater than the sum of the (high value) losses of hard and rarer examples.

To explain how the focal loss works, first of all it is necessary to recall the formula of the *Binary Cross Entropy loss (BCE)*:

$$BCE(p, y) = \begin{cases} -\ln(p) \text{ if } y = 1 \text{ where p is the probability, given by the model, to have label y=1} \\ -\ln(1-p) \text{ otherwise} \end{cases}$$

Now, for simplicity, we can define p_t as the probability of the true class:

$$p_t = \begin{cases} p \text{ if } y = 1 \\ 1 - p \text{ otherwise} \end{cases} \quad \text{Thus, we can write } BCE(p, y) = BCE(p_t) = -\ln(p_t)$$

Finally we can define the (binary) focal loss as follow:
$BFL(p_t) = -(1 - p_t)^{\gamma} \ln(p_t)$

The parameter γ is a focusing parameter that usually is setted to 2, but can have every value. If γ is equal to 0, we obtain the classic BCE, while as γ increases, less and less importance will be given to easy examples.

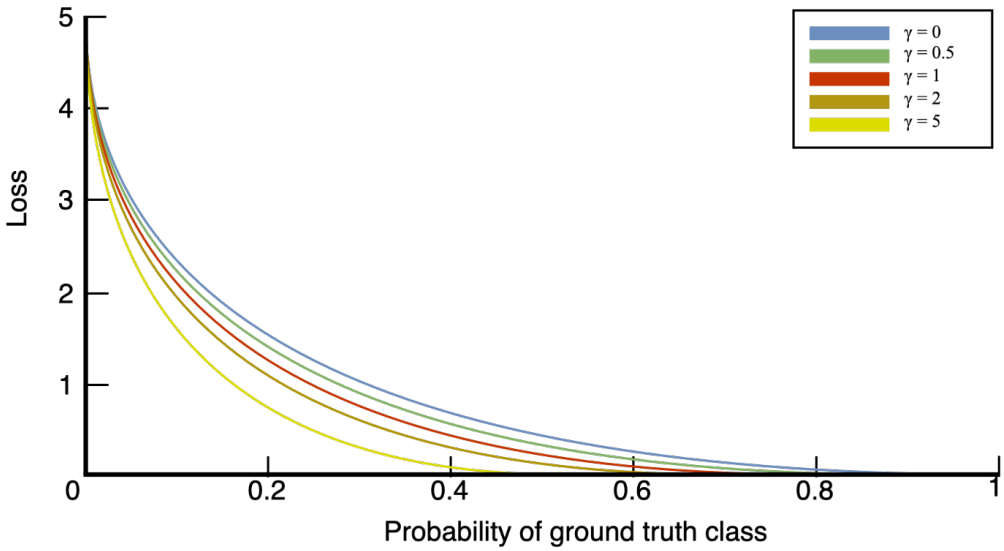


Figure 1.3: Graph representing the change of the loss value in relation with the parameter γ

Dice Loss If the focal loss is useful to make the network focuses more on difficult and rare examples, the dice loss [14] permits to deal with situations where the number of foreground pixels is much lower than the number of background pixels (.e.: when there is an imbalance between the two classes). Let define P as the area of the predicted segmentation mask and G the area of the ground truth mask, we can define the dice loss as:

$$Dice\ Loss = 1 - \frac{2*P \cap G}{P+G}$$

If compared with the more classical intersection over union, the dice loss give more importance to the intersection between the ground truth and the predicted label, thus the result will be a more precise segmentation.

Training of SAM with SA-1B dataset: the largest segmentation dataset

The used dataset was a new one called *SA-1B* which is formed by 11 million images with 1.1 billion masks. This dataset was built in an incremental way. First of all, a group of professional labelers used a browser-based version of SAM with also other tools like ”brush” and ”eraser” to perform semantic segmentation on objects (at that time SAM was trained using public available segmentation datasets). After a quite big dataset was created, SAM was retrained with the new data improving the performances and permitting to the labelers

to segment more objects in the same amount of time (note that this step is very related to the work proposed in this thesis!), thus the process of labeling/training was iterated six times. After that, two other steps were used:

- Semi-automatic stage: SAM automatically detected high confidence masks and the labelers had to segment only non already segmented things;
- Fully-automatic-stage: the model was prompted with a grid of 32x32 points and each of them outputted more than one mask. Then, IoU (Intersection over union metric) was used to select confident masks and NMS (non-maxima suppression) was used to delete duplicates.

1.3 Initial pipeline: Semi-automatic annotation with SAM and Key Points

Before my internship, to annotate their datasets, SpecialVideo used an online version of SAM on CVAT. The tool permits to upload images and annotate them choosing appropriate labels (which the labeler has to manually select since SAM does not perform classification). To perform the labeling the user can leverage on SAM with key points prompt (in particular foreground key points to specify which region consider in the mask and background key points to exclude parts that the model can confuse as part of the current object) to do the biggest part of the work and on other tools like a ”brush” and an ”eraser” to refine the masks.

This pipeline (reported in figure 2.1), even if significantly faster than using only the brush and the eraser, has a lot of drawbacks:

- Since it is an online tool some tasks (like uploading/downloading the images, waiting the response of SAM after a click, backup the project and others) are very slow since they depend on the internet connection and on the speed of the servers of CVAT;
- The labelers (especially if s/he is new on a task) has no hint on what to do and has no intuition of how looks like a good mask;
- The time is not optimized since the knowledge acquired by annotating the images is not used to speed up the process.

In the following chapters all the points above will be touched and fixed resulting in an astonishing speed up of the process (see chapter 5 for in depth analysis of the results).

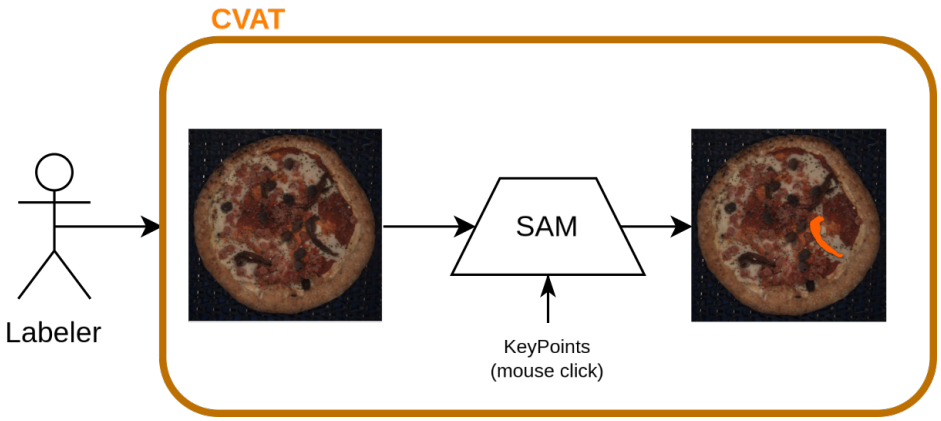


Figure 1.4: Structure of the first pipeline using only CVAT online

Chapter 2

A first improvement using YOLO and SAM

All the drawbacks highlighted in the last chapter, will be solved in this chapter, while in the following one we propose an even better approach using a more recent and powerful network.

2.1 Performance boost with local CVAT and Docker

The first task consisted into installing CVAT in a local machine (with a GeForce 4090 to run the models) to avoid the limitation of the free online version without paying, to be able to load in CVAT our own fine-tuned models and to speedup almost every operation deleting the dependencies from the servers of CVAT.

As the official documentation reports [5], to use CVAT in a local machine it is necessary to use Docker [6] which is a software very useful to set up customized programming environments and also, as in the case of CVAT, to develop applications which are divided in different modules – micro-services – and then put them together during the build. This permits to add features and to modify existing ones in a easier way. Moreover, Docker is very fast in Linux like systems since, differently from a virtual machine, the kernel of the virtualised operating system (each modules has the proper os) is shared with the one on the local machine.

Docker has two main concept to handle: the *images* and the *containers*. To make a comparison with the Object Oriented Programming (OOP) paradigm, the images are a general description of something and so can be viewed as a class; on the other hand, the containers are instances of an image which also contains data, thus they are equivalent to the objects in the OOP paradigm.

Already this simple operation boosted the performances since now everything is on one local machine. As reported in table 2.1, each operation done on the local machine has a time saving of at least 40%.

	Online counterpart
Project Backup - Offline	-50%
Export in CVAT1.1 - Offline	-40%
Export in segMask - Offline	-70.8%
Image Upload - Offline	-95%

Table 2.1: The table reports in percentage the time savad in various taks simply switching to CVAT offline. All the tests were done using 20 images.”CVAT1.1” and ”segMask” are two formats in which one can export a labelled dataset.

2.2 A first approach to automatic annotation using YOLO

A first step we tried was the use of YOLO as segmentor since it is very fast to run and also because the company already had a know-how on YOLO. The idea was to use few labeled pizza images to fine tune YOLO making it able to recognize the desired ingredients and then pass the YOLO-labelled images to the labeler. In such a way, after few raw pizzas, the labeler will have to do less work and thus, beeing able to label more pizzas in less time. The update pipeline is reported in figure 2.1

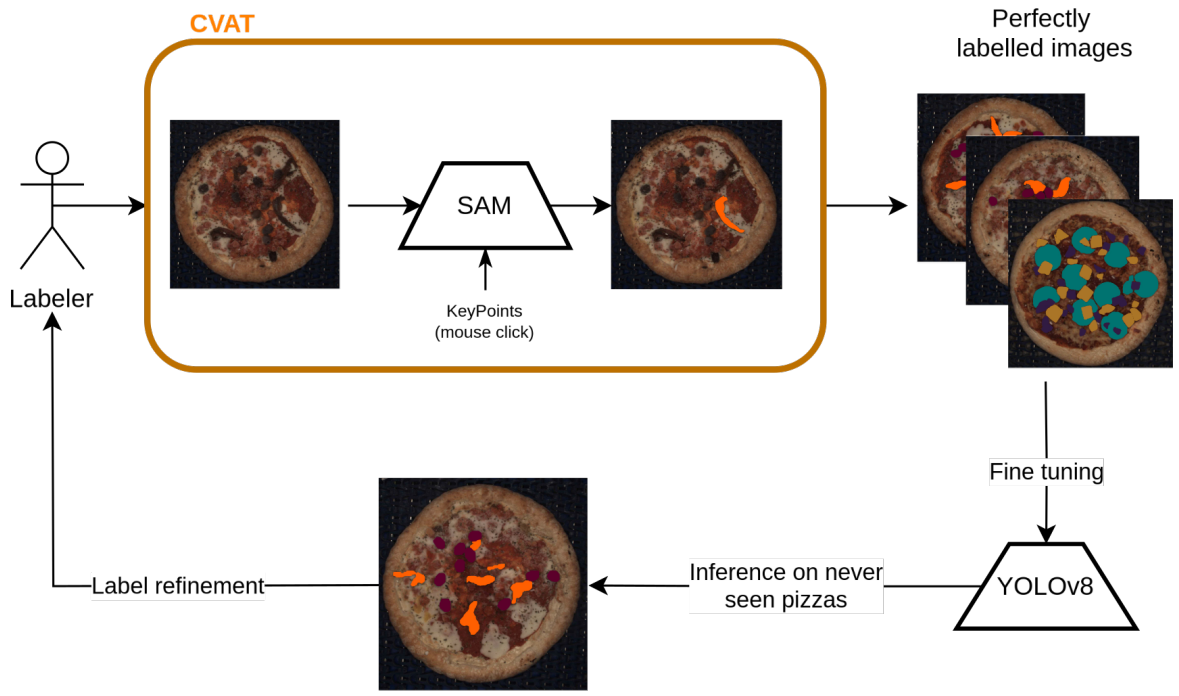


Figure 2.1: Structure of the improved pipeline using CVAT offline and YOLO as segmentor.

2.2.1 YOLOv8

YOLOv8 is the eighth version of the famous single shot detector YOLO. YOLOv8 is composed by three main parts:

1. Head: The head is basically a set of convolution layers, there is a stem layer followed by other 4 layers: the activation the last three layers are used in the following stage. The technique to use not only the last activation of a network is called *Feature pyramid network (FPN)*. Basically, since the deeper activations have a well semantic knowledge, while previous activations have more spatial information, the union of more activations (usually with some up-scaling followed by a sum or concatenation) permit to have bounding boxes or segmentation masks in output that are both precise and with the correct class;

2. Neck: the neck implements a PANet (path aggregation networks). Basically, it extend the concept of FPN: if in FPN there is only a path that up-scales deep activations to merge them with previous activations, in PANet there is also a second path which downscale high level activations to merge them with deeper ones.

3. Head: YOLOv8 implements 3 heads, each one is connected to a different part of the

neck to be able to produce different kinds of outputs.

2.2.2 The dataset

The dataset used was composed by 3 kinds of pizzas and the ground truths were created using CVAT with SAM (note that for margherita pizzas the ground truths were generated in python creating a black image since they represent background examples):

- Anchovy and black olives: Quite easy to segment since the two ingredients are very different;
- Salami, red pepper and yellow pepper: Very difficult to segment because we have the tomato sauce in the base, the red pepper and the salami which are all of the same color. Furthermore, both the yellow and red pepper are a lot;
- Margherita: They were mandatory to train YOLO since in the documentation is clearly state that YOLO needs a number of background examples to achieve good performances.

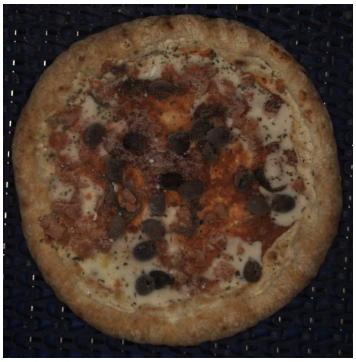
The choice of those formats was done because with one simple and one difficult formats we can have a complete idea of what the tested networks can do and which are their limits.

We created three different datasets: one with 40 images, one with 160 images and one with 320 pizzas, then each dataset was completed with margheritas to achieve 56, 176 and 352 sample each (see table 2.2). Moreover, to have reliable results for our pipeline idea, each dataset is a subset of the ones bigger than it ($Dataset\_20 \subset Dataset\_80 \subset Dataset\_160$). Note that we are referring to the training dataset, the validation dataset was made by a total of 40 pizzas (not including margheritas) while the test set was made by 20 pizzas of which 10 with anchovy and olives and 10 of the other format (the one with salami). The creation of more than one dataset was done to understand how many pizzas we need at least to start helping the labeler in its work.

Dataset	Number of olives and anchovies pizzas	Number of salami and peppers pizzas	Number of margherita pizzas	Total
Dataset_20	20	20	16	56
Dataset_80	80	80	16	176
Dataset_160	160	160	32	352

Table 2.2: The table contains the data split for the three dataset we created.

(a) Margherita pizza, used as background example



(b) Anchovies and black olives pizza

(c) Salami, red pepper and yellow pepper pizza

Figure 2.2: Pizzas included in the dataset

2.2.3 Fine tuning YOLOv8

To fine-tune YOLOv8 we used the python API from Ultralytics [28] and we trained three different models (YOLO nano, YOLO medium and YOLO extra large) to see which one performs best in our situation. All three models were first trained on the data set with 100 images and then on the one with 200 images (we did not try the 20 images dataset since they were too few for YOLO).

After the training, Ultralytics automatically generates some images with useful metrics. The confusion matrix reported in figure 2.3, already tells us that there are some problems:

- The biggest problems are the peppers, in particular it seems that both red and yellow peppers are recognised as background (i.e: not identified) around the 60% of the times while for the 33/32% of the times they are classified of the opposite color.
- The second problem, even if less serious, regards the anchovies which are not classified in the 31% of the cases, while are correct classified in the other case (except for a 2% of miss classification with the olives).

Looking at the images, a possible explanation for the errors reported above is the fact that the peppers are small and both yellow and red peppers have the same shape. Furthermore, the red peppers have a color which is very similar to the one of the salami and also to the color of the pizza's sauce. Writing about the anchovies, they are quite thin and difficult to see even for a human. Moreover, if compared with the olives (on which the network performs very well), there are way less instances of anchovies, since in a pizza usually there are more

olives than anchovies. As example, in figure 2.6a, there is a pizza segmented by YOLO and we have a not labelled anchovy, an anchovy labelled as olive and also an anchovy correctly identified but with a mask way larger then the actual area of the fish.

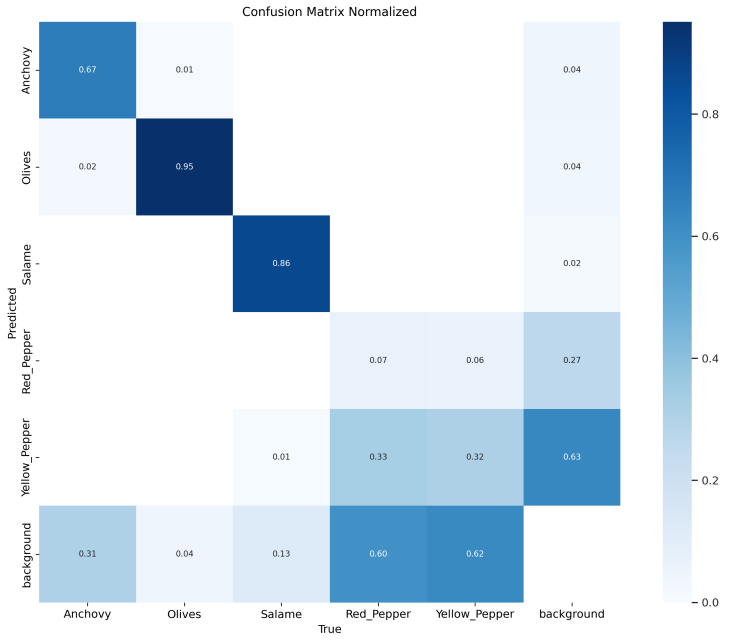


Figure 2.3: Normalized confusion matrix of the training of YOLO nano on 100 images

(a) Potential difficult part of olives and anchovies pizza

(b) Potential difficult part of salami and peppers pizza

Figure 2.4: Example of difficult parts to segment. On the left some olives cover the anchovies, while on the right there are some red peppers on a salami slice which are difficult to see even for a human.

The observations done with the confusion matrix are confirmed also by the ingredient-wise F1 curve reported in the figure 2.5. It is quite clear that even if the overall F1 score is not so bad (0.66), this quite positive result is trained only by the classification of Olives and Salami which the network can recognise very well, while the curves of the pepper reach a

peak of only 0.4. Another important thing to notice is that the network is not confident about the predictions since the peaks of the peppers are a the extreme left of the chart.

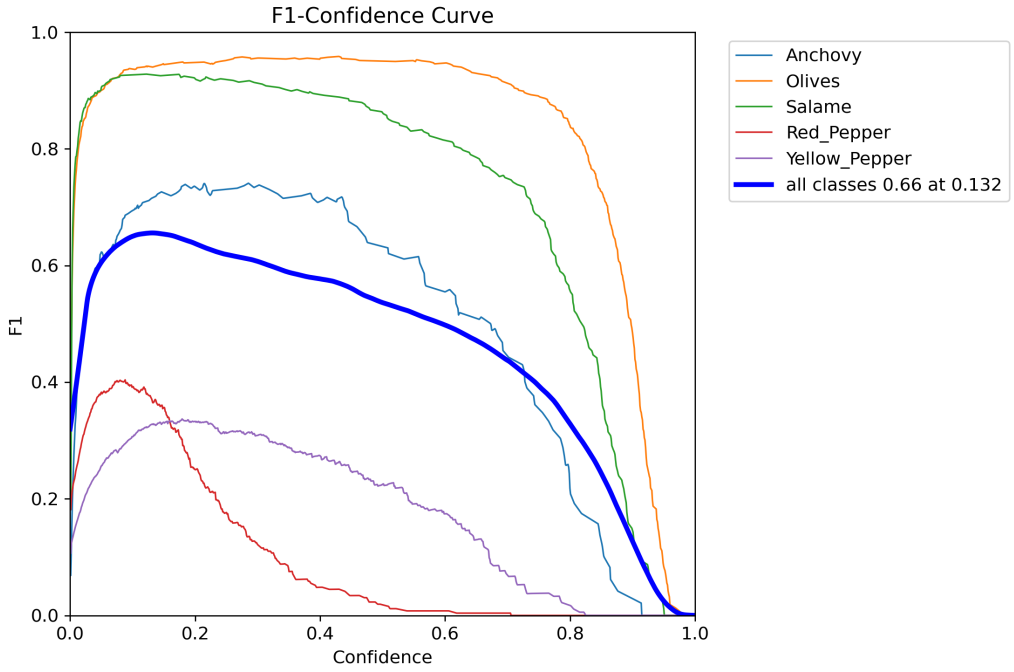


Figure 2.5: f1 curve of the training of YOLO nano on 100 images

2.2.4 Performances

To verify the performances of YOLO we used a particular metric: the multi-label mean intersection over union. The reason is that YOLO failed a lot with some predictions, in particular some pepper were labeled as both red and yellow and we want to consider the fact that one of the two is correct and at the same time we want to leave the decision on which label to keep to the labeler during the refining step. The idea behind this metric is the same as the normal mean intersection over union, but an object could have one or more labels associated.

The results achieved by YOLO were not very good – especially for the problem on the peppers –, but, as reported in the chapter 5 (§5) it still be able to help the labeler. The results achieved by YOLO are summed-up in the table 2.3

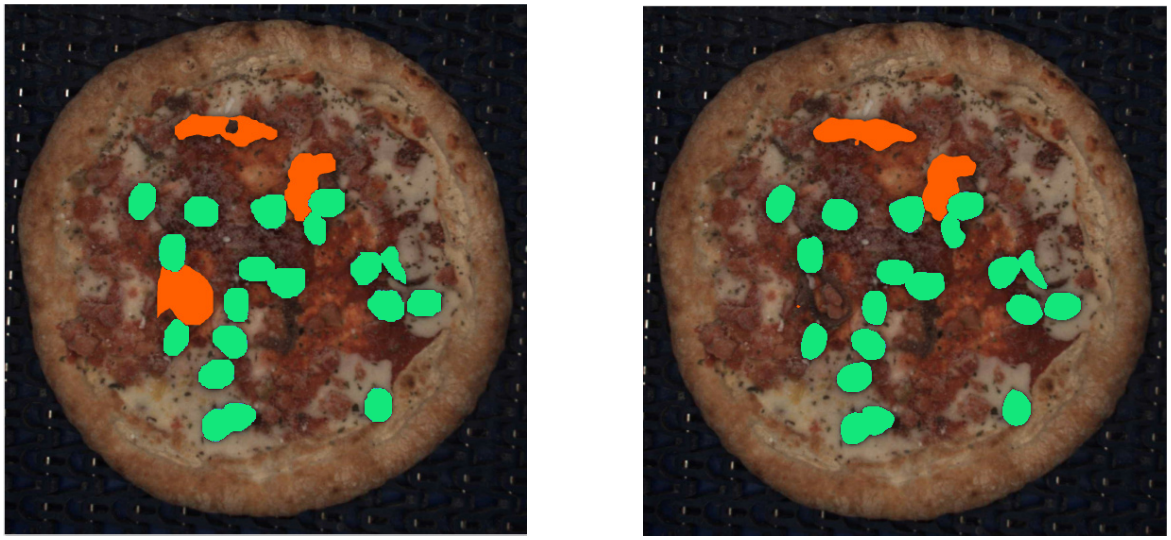
As the table 2.3 shows, the three different YOLOv8 models: nano (3.2 Millions parameters), medium (78.9 Millions parameter) and extra large (257.8 Millions parameters); have

Model	Inference time (seconds) on 20 images	multi label IoU Salami pizza	multi label IoU Anchovy pizza	Mean IoU
n_100	1.6	39%	67%	53%
n_200	2.3	40%	69%	54%
m_100	2.5	47%	27%	37%
m_200	2.7	31%	67%	49%
x_100	7.2	49%	63%	56%
x_200	7.1	44%	55%	50%

Table 2.3: The table shows the mean multi label IoU for the two tested formats of pizza. The percentages are a mean over 10 pizzas for each kind. the number in the model name indicate the number of samples in the train set.

similar results, probably due to the lack of more images. The best results came from the extra large models, but due the long inference time and also because on the 200 images tests the nano model beat the extra large, we chose to do the tests reported in 5 with the nano model.

2.3 Merging the labeling capabilities of YOLO with the performances of SAM



(a) Pizza labelled with YOLO

(b) Pizza labelled with YOLO + SAM

Figure 2.6: The same pizza segmented with YOLO and with YOLO + SAM.

After having tested YOLO, we noticed that, other the miss classification problem, we had also some masks with a quite wrong shape or with not perfect borders. To resolve this problem we tried to merge YOLOv8 with SAM. The idea, reported in the figure 2.7 is to use YOLO only as simple object detector and than feed all the bounding boxes to SAM in order to perform semantic segmentation.

At the cost of a very slow inference (but this is not a big problem for SpecialVideo since the inference part can be done earlier, in every moment before the work of the labeler), we achieved a final model with the better characteristics of both: the model is able to locate only the interesting objects, to label them and to segment them in a way better manner if compared with the plain YOLO pipeline. To have a more precise idea of the improvements see chapter 5.

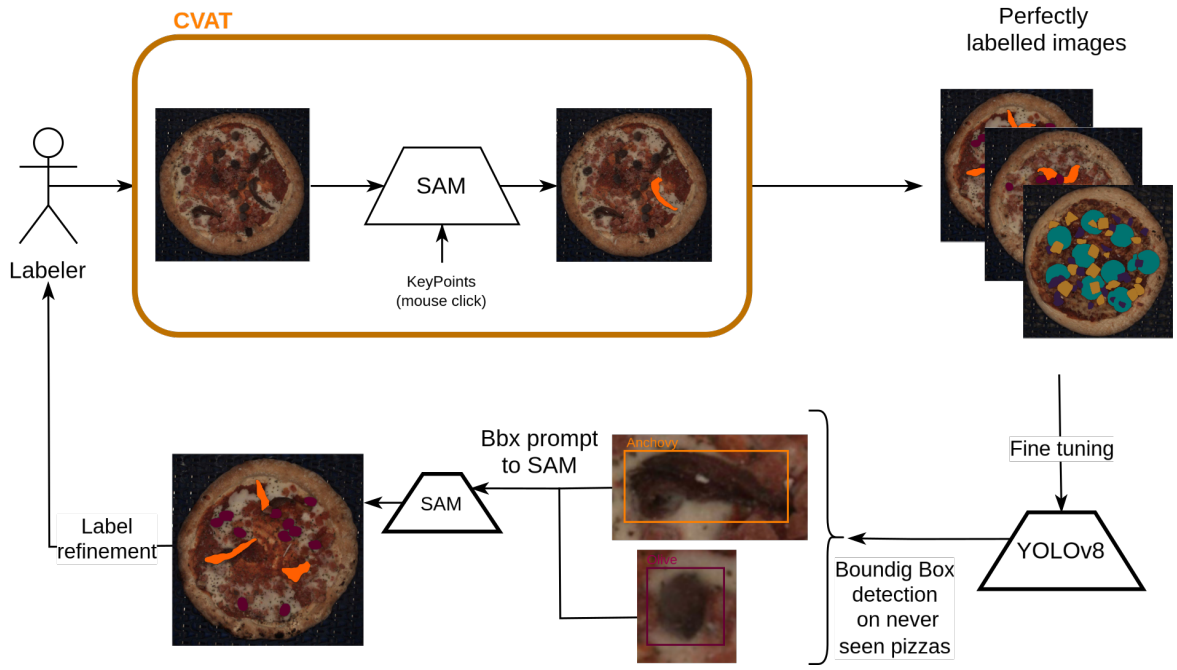


Figure 2.7: Structure of the improved pipeline using CVAT offline and SAM with bounding boxes as prompt.

Chapter 3

Data efficient segmentation with SegFormer

After the tests done with the two previous architectures our desire was to improve the results on both the quality of the segmentation masks and the ingredients localization capabilities. To achieve our goal, we sought a transformer architecture designed to enhance performance while also ensuring rapid prediction capabilities. This attribute is critical for streamlining the labeling process, saving valuable time, and enabling the deployment of the selected network in a production environment for SpecialVideo's clients.

We founded a network with such specifications and it is SegFormer a "simple, efficient, yet powerful semantic segmentation framework which unifies Transformers with lightweight multi-layer perceptron (MLP) decoders" [27].

Following, the structure of SegFormer will be discussed as well as the modality we used for fine tuning it.

3.1 SegFormer: a Segmentation specific Vision Transformer

Before SegFormer other transformer architectures for image tasks – including semantic segmentation – were proposed, like Swin transformer [13] which uses a sliding-windows attention to improve the efficiency and also redesigns the encoder of a classic ViT to have a multi-scale output which sets up the network for techniques like FPN. These architectures modified only the encoder part of the network, while SegFormer modified both the encoders

and the decoder. The complete schema is reported in figure 3.1.

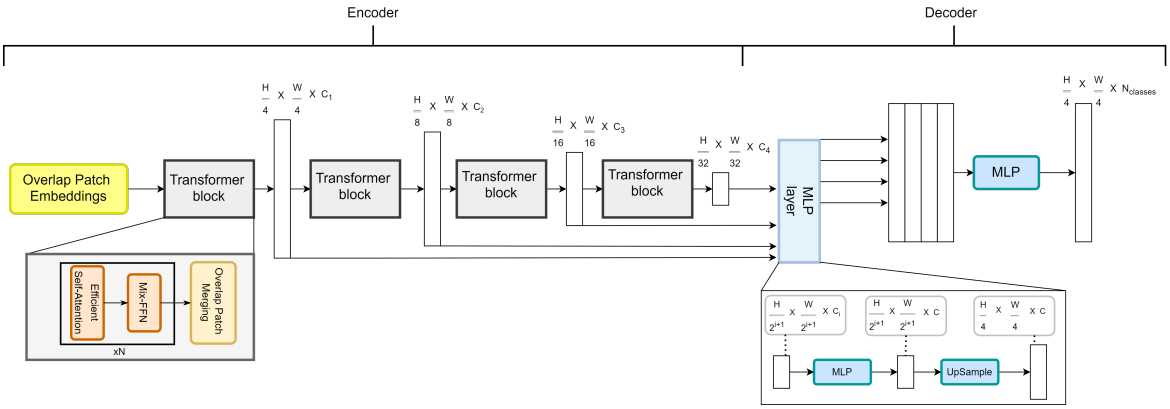


Figure 3.1: Structure of SegFormer.

3.1.1 Hierarchical transformer encoder

As mentioned above, one of the key features of SegFormer is that it outputs more activations which are then passed to the decoder. The resolution of that outputs is $\frac{H}{2^{i+1}}$ x $\frac{W}{2^{i+1}}$ x C_i where i is the output level of the feature map. This technique permits to have both semantically rich information, from the deeper activations, and spatially rich information. The latters, permit to have a more pixel-precise segmentation since the network have more spatial information to reason on.

Another major contribution from this paper is the efficient computation of self attention (§0.2), which usually is $O(n^2)$ (time complexity), here it reaches a complexity of $O(\frac{N^2}{R})$ where R $\in [64, 64, 4, 1]$, based on the current layer. In the standard ViT the Query (Q), Key (K) and Value (V) matrices have dimension of HxWxC and the computation is:

$$Attention(Q, K, V) = Softmax(\frac{Q * K^T}{\sqrt{d_{head}}}) * V$$

To achieve this complexity reduction, the length of the sequence to be processed (K) is reduced as follow:

$$\hat{K} = Reshape(\frac{N}{R}, C * R)(K)$$

$$K = Linear(C * R, C)(\hat{K})$$

Thus, the final, new, *K* will have shape $\frac{N}{R}$xC

Last, but not least, it is worth to mention the fact that SegFormer uses a variation of the positional encoding. Usually, both the text transformer and the ViT use positional encoding to permit the network to know which was the original position of a given patch of the input. Writing about the ViT, the drawback of the more classical versions of the positional encoding is that its resolution is fixed and this could causes a drop in the performances if the network is tested with images with a different resolution than the training one. The solution adopted in SegFormer to leak the local information is to use a *mix-FFN* layer on the output of the attention module. Let x_{in} be the input of the *mix-FFN* layer, the output is computed as:

$$x_{out} = MLP(GELU(Conv_{3x3}(MLP(x_{in})))) + x_{in}$$

Depth-wise convolutions are used to improve efficiency.

3.1.2 MLP decoder

Writing about the decoder, SegFormer adopted an approach very different from previous state-of-the-art segmentation transformers. Instead of using convolutions as head, which are quite heavy in terms of computational time, they adopted an MLP decoder. This choice was possible thanks to the multiple outputs described in the previous pages; they permit to a simple head to have all the necessary information to perform the segmentation giving spacial and semantic cues to the decoder.

The process can be described in four steps:

1. First of all, the features (F_i) coming form the different layer pass through a linear layer to unify the number of dimensions to C channels $\rightarrow \hat{F}_i = Linear(C_i, C)(F_i), \forall i$;
2. The second step instead, unifies the spatial dimensions to $\frac{1}{4}th$ permitting then the concatenations of the features $\rightarrow \hat{F}_i = Upsample(InputDim, (\frac{W}{4}, \frac{H}{4}))(\hat{F}_i), \forall i$;
3. Now, the concatenated features can be merged using a simple linear layer $\rightarrow \hat{F} = Linear(4C, C)(Concat(\hat{F}_i)), \forall i$;
4. Finally, the merged features are projected in a $N_{classes}$-dimensional space which is the output mask that has $\frac{W}{4}$x$\frac{H}{4}$x$N_{classes}$ resolution $\rightarrow \hat{Mask} = Linear(C, N_{classes})(F)$;

The output mask then can be manipulated in three steps to obtain a segmentation mask with the same shape of the input image:

1. First, the mask has to be up sampled to match the height and the width of the input $\rightarrow$ $Mask = BiLinearUpsample((\frac{W}{4}, \frac{H}{4}), (W, H))(\hat{Mask})$;

2. Then, the Softmax function is used to have a probability distribution along the channels, in this way for each pixel the most likely class will be known;

3. At the end, a mono-channel mask in builded using the *Argmax* function along the channel dimension.

The researchers made available six different models differing for the number of parameters, fer example the model *b4* refers to one of the biggest models available, all the models are reported in the table 3.1 alongside their performances on the cityscape dataset.

Model	Params (Millions)	mIoU on Cityscapes
MiT-b0	3.8	76.2%
MiT-b1	13.7	78.5%
MiT-b2	27.5	81.0%
MiT-b3	47.3	81.7%
MiT-b4	64.1	82.3%
MiT-b5	84.7	82.4%

Table 3.1: Comparison of the different SegFormer models

3.2 Data efficiency using data augmentation techniques

To fine tune SegFormer we started with the simplest scenario: we used the *Dataset_80*, without margherita pizzas, (see table 2.2) with the *b4* SegFormer model.

The results of this first experiment were very encouraging since we obtained 83.7% of mIoU on the test set (note: the test set is always used with full resolution images). For what regards the training recipe we just tuned the learning rate to $4e^{-5}$, the weight decay to 0.005 and we used a photo-metric data augmentation as well as rotation data augmentation; in particular each pizza was rotated of 0, 90, 180 and 270 degree since a rotated pizza has a

different distribution of the ingredients but given its round shape it remains a valid training data. The epoches were only 5 since then the model starts to overfit. Giving the fact that the score was very high and since we want to optimise our pipe-line we tried also to fine tune the models *b1* and *b0* to have more velocity during the inference. The former model reached 83.4% of mIoU while the latter 80.6%.

Our results were astonishing since we surpassed the scores the researches reached in the paper. This fact is explainable by the intrinsic characteristic of our dataset: we have only 5 classes (plus the background) and, most important, the dataset is very homogeneous. Even if the toppings and their distribution over a pizza are different, a pizza is always a pizza and this permit to the network to easily separate the crust and the pizza's base from the toppings. This thing is very important to underline since our results are not dataset dependent because, for industrial applications, this characteristic of having very similar images is a constant factor and so we argue that our results can be translated to a very large set of industrial vision problems.

Given such good results we tried to fine tune the network with less samples using the dataset *Dataset_20*, also in this case without the background examples (the margherita pizzas). First of all, we trained the network *b1* (we switched directly from *b4* to *b1* because in our tests they resulted the two best models considering the trade-off between speed and performances) with 5 epoches and we obtained a 53% score on the test set, but the network this time seems to be able to learn more after the 4 epoches of the first test. Thus, we keep training until epoch 15 achieving a score of 78%. The fact that the network is very good also with such few training data is an enthusiastic result for our project: we can help the labeler just after 20 pizzas, meaning that from the 21th pizza on, the labeling time can decrease speeding up the whole process.

Given that, we started to optimize the training to obtain better and better results using the dataset *Dataset_20* and adding the margherita pizzas as background examples, applying the rotation data augmentation also on the margheritas and optimizing again the hyper parameters, we reached 79.9% mIoU with the model *b1* and 86.4% mIoU with the model *b4*.

Lastly, we used our discoveries (using margheritas, data augmentation on margheritas

and the new hyper parameter) to train two *b4* models respectively with the dataset *Dataset_80* and *Dataset_160* reaching our best results of 87.9% and 88.6% of mIoU on the test set. All the results are summed up in the chart in the figure 3.2.

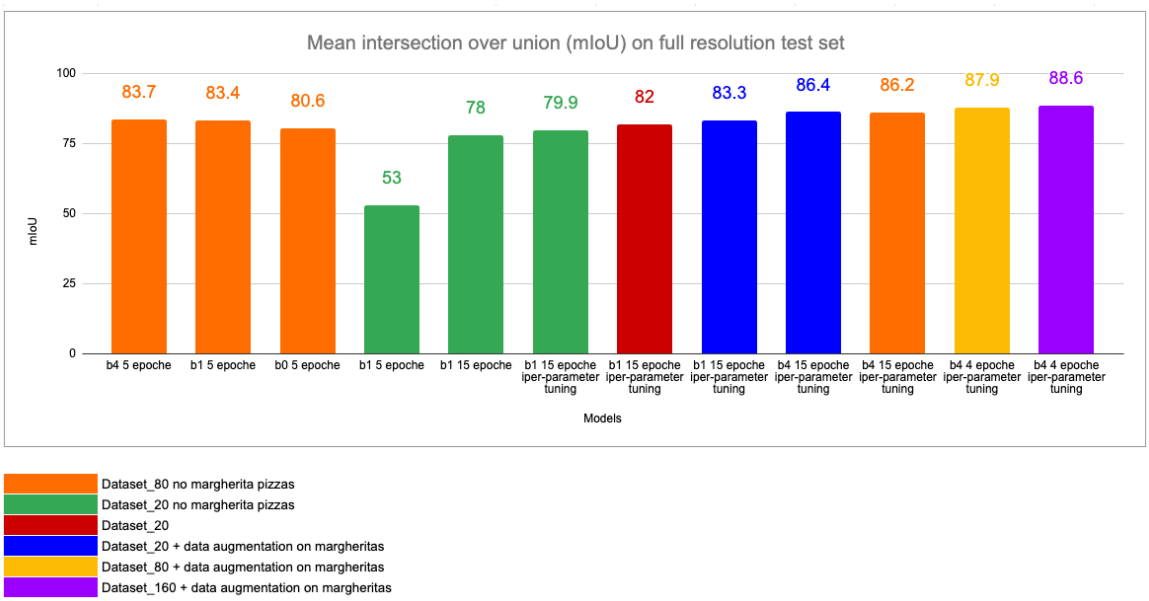
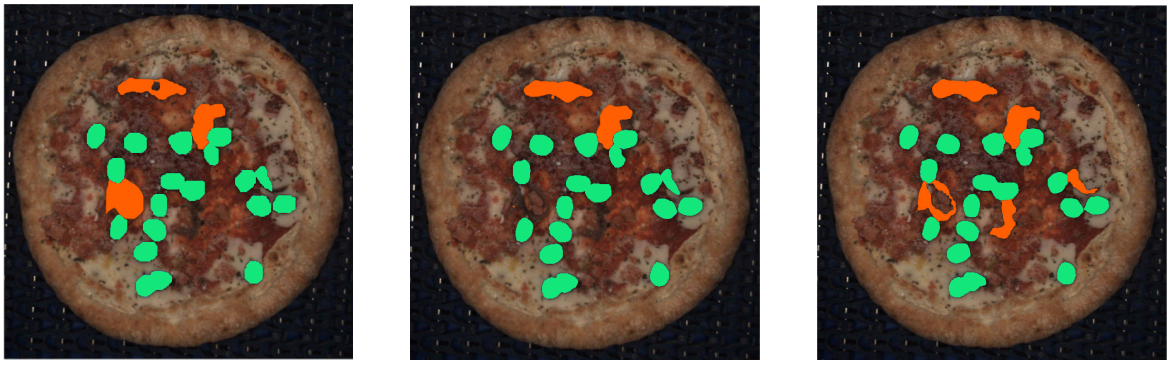


Figure 3.2: Comparison between the different training done with SegFormer.

After having evaluated the various models we trained, we chose to use for the comparisons the model *b4* trained on the dataset *Dataset_80* with data augmentation applied also on the margheritas (yellow column in the figure 3.2) since it is our second best SegFormer model and it was trained only with the *Dataset_80* and, for the reasons already explained, we want to put the focus on the models that perform well with less images possible. Making a visual comparison with the other two architectures discussed in the previous chapter, looking at figure 3.3 it is clear that SegFormer outclasses them. It is able to localize more anchovies, the segmentations are way more precise and there are not mislabelled objects. These results are very important for our pipeline and the time saving will be investigated in the last chapter (§5).

There is only one draw back in SegFormer and it is the fact that it performs semantic segmentation, while YOLO and YOLO + SAM, thanks to the object detection capabilities of YOLO perform instance segmentation. This is not a big problem, but it has an impact on the time that the labeler will use to adjust the segmentation proposed by the network. Again this thing will be better described in the last chapter (§5) with also a simple method to work



(a) YOLOv8 (b) YOLOv8 + SAM (c) SegFormer

Figure 3.3: The same pizza segmented with YOLO, YOLO + SAM and SegFormer trained on *Dataset_80* with data augmentation on margheritas.

around this problem using together YOLO + SAM and SegFormer.

Chapter 4

ViT-Adapter

As described in chapter 5, SegFormer really helped us in our aim to speed up the labeling process, but we were curious if a more recent architecture can do even better. For this reason, we gave a try to a vision adapter from the paper "Vision Transformer Adapter for Dense Predictions" [1].

4.1 Multi modal pretraining and task specific fine-tuning

In this paper, the researchers tried an approach which is the exact opposite of the one used in SegFormer. Inspired by the adapters in the NLP field, they tried to use a normal ViT –thus, without any image specific modifications like the use of the FPN technique— with multi modal pretraining to permit to the model to learn semantically reach representations. This can be done quite simply using different tokenizers based on the current input and this technique permits to a plain ViT to compete with vision-specific transformers. After this massive multi modal pretrain, a pretraining-free adapter is added to the architecture to adapt it to downstream vision dense tasks (e.g.: semantic segmentation), the figure 4.1 highlights the differences between the ViT-Adapter and a general transformer modified to work with object detection and semantic segmentation. To merge the ViT backbone with the adapter, the researches designed three modules: a **spatial prior module**, a **feature injector** and a **multi-scale extractor**.

Writing about the adapter module, the image is first of all feeded to the spatial prior

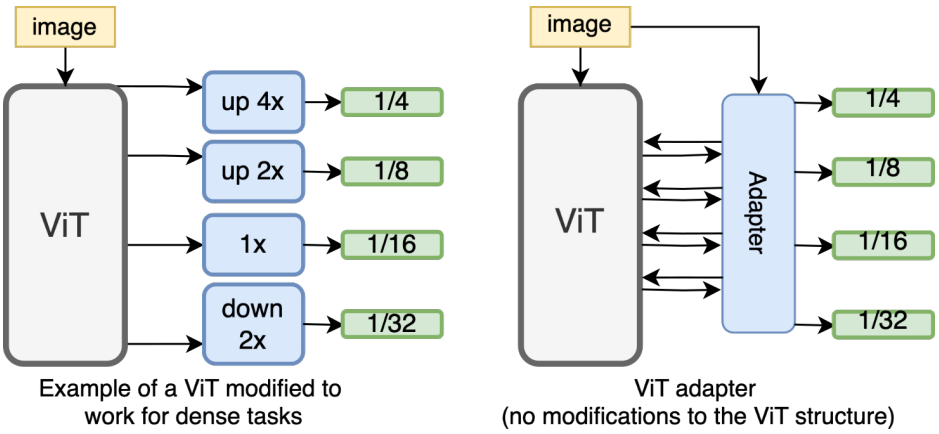


Figure 4.1: Simple schema to highlight to differences between the Adapter (right) and a ViT modified to work with dense tasks (left).

module and features with spatial resolution of $\frac{1}{8}$, $\frac{1}{16}$ and $\frac{1}{32}$ are generated, flattened and concatenated for the future steps. Usually, the number (N) of interactions between the adapter and the ViT is 4, thus the ViT is divided in N blocks with $\frac{NumberOfEncoderBlocks}{N}$ encoders each. At each block i, the spatial priors are injected in the encoders and then the multi-scale extractor computes the hierarchical features that will be used as input for the following injector, a schema is reported in the figure 4.2.

At the end, the output features are divided and reshaped to a resolution equal to $\frac{1}{8}$, $\frac{1}{16}$ and $\frac{1}{32}$ of the input. Lastly a transposed convolution is applied to the $\frac{1}{8}$ resolution feature map to get a feature with resolution of $\frac{1}{4}$ having in this way a set of feature maps suitable for FPN technique.

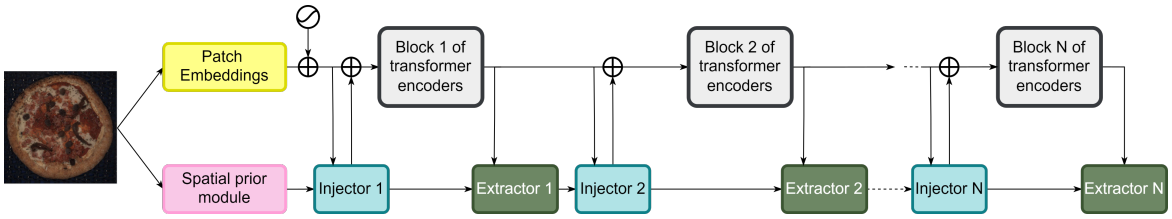


Figure 4.2: Structure of ViT-Adapter. The top path represents the standard ViT, while the bottom path represents the added atapter's modules. The $\oplus$ represents an element-wise addition, while the "undulated" $\oslash$ represents the position embedding.

4.1.1 Adapter structure

Following, the three main components of the ViT adapter are described more in details and a summary scheme is reported in the figure 4.3.

Spatial prior module This first module is composed by:

- A stem layer with 3 convolutions and a max pooling layer to rapidly decrease the spatial dimension and thus accelerate the following computations;
- A stack of stride-2 3x3 convolutions;
- Several 1x1 convolutions to obtain the wanted number of channels. As described above, this results in a feature pyramid output F_1, F_2, F_3 with D channels each, then these features are concatenated obtaining the feature token $F_{sp}^1 \in \mathbb{R}^{(\frac{H*W}{8^2}+\frac{H*W}{16^2}+\frac{H*W}{32^2})XD}$.

Injecting vision specific knowledge This module, as the name suggests, injects the obtained spatial priors into the ViT. For the i block of the ViT, the feature coming from it ($F_{vit}^i \in \mathbb{R}^{\frac{H*W}{16^2}XD}$) is used as query, while the once coming from the adapter module ($F_{sp}^i \in \mathbb{R}^{(\frac{H*W}{8^2}+\frac{H*W}{16^2}+\frac{H*W}{32^2})XD}$) is used as key and value and through the attention mechanism a new feature is computed. Furthermore, a learnable vector $\gamma \in \mathbb{R}^D$ is used to balance the computation (γ is initialized with 0s to ensure a not drastic change in the feature distribution of the ViT). Summing up, the output of the i-th injector is computed as follow:

$$\hat{F}_{vit}^i = F_{vit}^i + \gamma^i * Attention(LayerNorm(F_{vit}^i), LayerNorm(F_{sp}^i))$$

Multi-scale feature extractor The features computed by the injector are feeded to the next ViT block of encoders and the output is feeded to a feature extractor. The extractor, thanks to a feed-forward network and another attention module, generates multi scale features which are used as input for the next injector. The injector as well as the extractor uses sparse attention – in particulate deformable attention – to reduce the cost of the computations.

$$\hat{F}_{sp}^i = F_{sp}^i + Attention(norm(F_{sp}^i), norm(F_{vit}^{i+1}))$$
$$F_{sp}^{i+1} = \hat{F}_{sp}^i + FFN(norm(\hat{F}_{sp}^i))$$

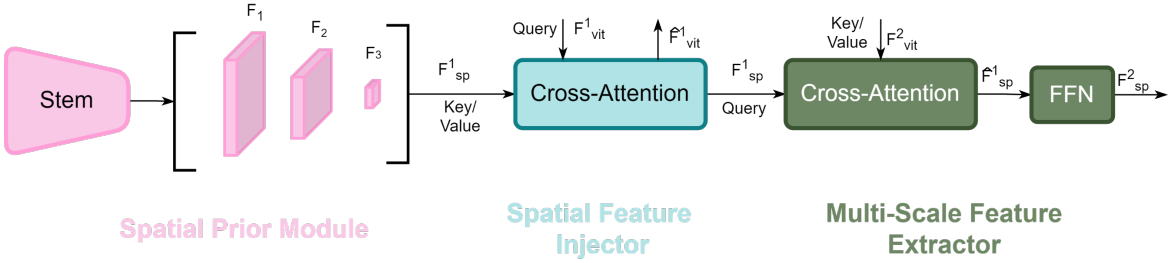


Figure 4.3: Structure of main modules of the ViT adapter.

Deformable attention

As mentioned previously, the ViT-Adapter uses deformable attention to speed up the training process. They took inspiration from the paper [29] where the researchers used the deformable attention to overcome some problems of the standard attention used in ViT. The standard attention looks at all possible location requiring a lot of time and memory and generating features that could be influenced by irrelevant areas of the image.

The deformable attention, which schema is reported in figure 4.4, uses a small set of keys (sampled around a reference point) for each query helping to converge faster. The attention weights as well as the sampling offsets are learned through linear projections and the features maps are extracted around the reference point using bilinear interpolation (since can happen that the sampled points are not aligned with pixel grid).

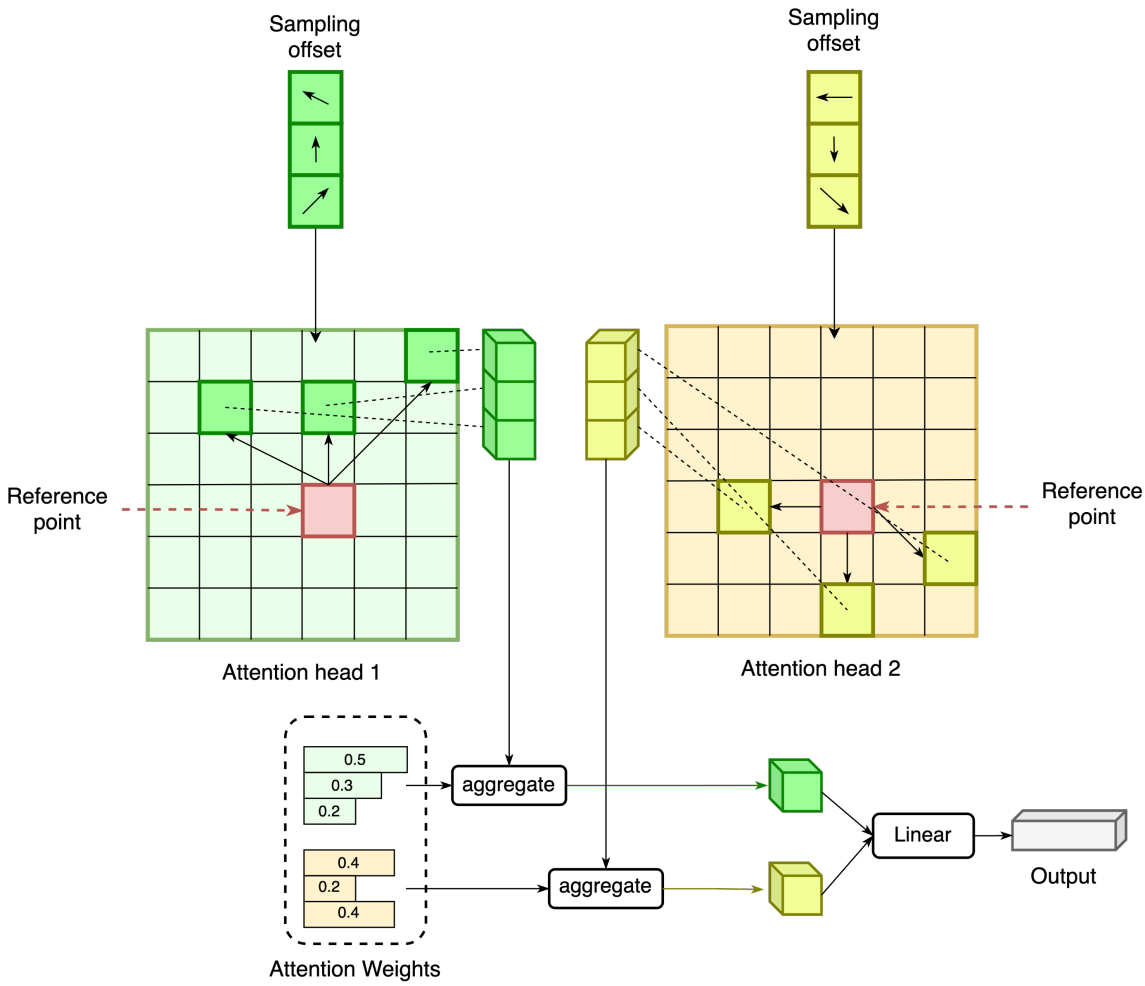


Figure 4.4: Example of deformable attention with two attention heads and three sampling points.

4.1.2 Mask2Former as segmentation head

In the ViT-Adapter paper [1], the researchers proposed some different heads for each proposed task. From the various proposals for semantic segmentation we chose to use *Mask2Former* [2] because they reached very good results with it.

MaskFormer

Mask2Former is a segmentation head by Meta AI research team which is able to address any segmentation task (panoptic, instance and semantic). Mask2former is built starting from *MaskFormer* [3] (always from Meta) with a new transformer decoder which uses masked attention.

MaskFormer is composed by three main modules which are also reported in Figure 4.5:

1. A backbone to extract features;
2. A pixel decoder which gradually upsamples the backbone's output with the aim to generate per-pixel embeddings;
3. A transformer decoder that computes learnable positional embeddings.

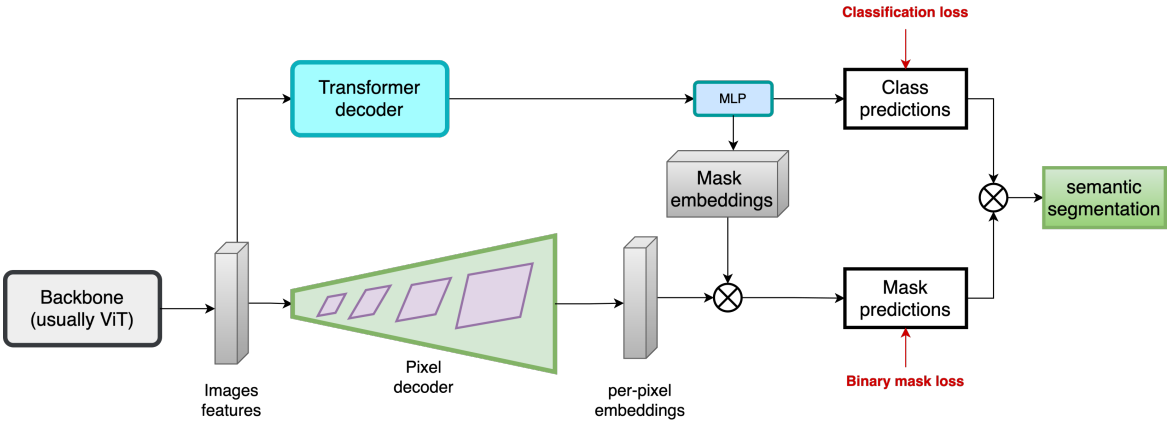


Figure 4.5: Structure of maskformer.

Mask2Former

As written before, Mask2Former has the same architecture of MaskFormer, with some improvements: the transformer decoder has a masked attention (instead of the standard one) and to deal with small objects the decoder has as input the multi scale features generated by the pixel decoder but only one feature for each decoder is given, a schema is reported in figure 4.6.

Masked attention The standard cross attention has some drawbacks, in particular it is computationally demanding and requires long training to converge. The researchers of Mask2Former hypothesised that, for each query, a masked attention could make the model focus on local features of foreground regions of the predicted mask, while the context information can be learned thanks to a self-attention module (a schema of the decoder with the masked attention is reported in figure 4.7). A standard cross-attention with skip connection can be written as follow:

$$X_l = softmax(Q_l * K_l^T) * V_l + X_{l-1}$$

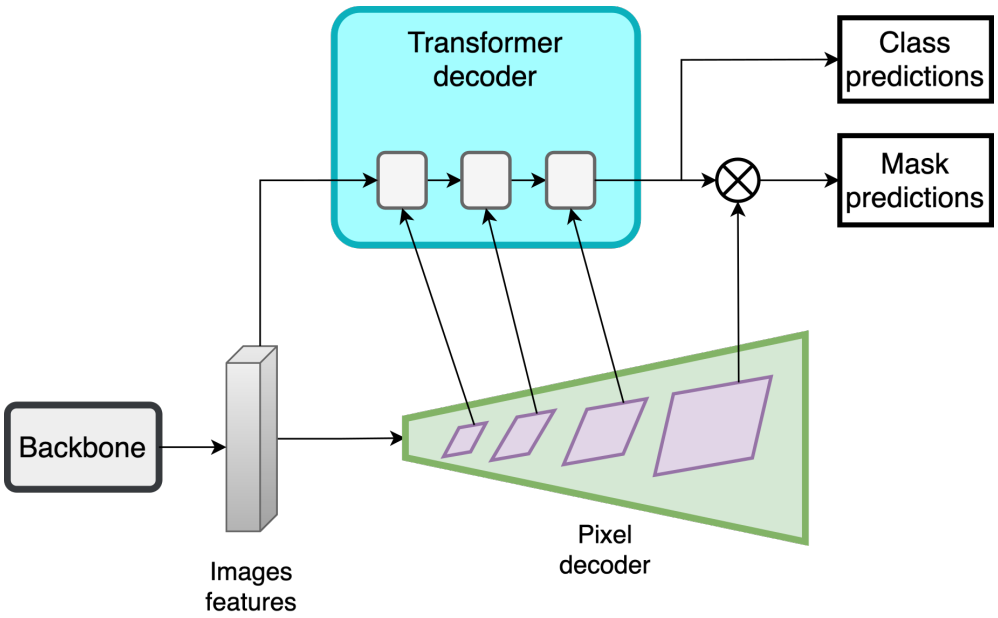


Figure 4.6: Structure of mask2former.

The proposed masked attention only add one attention mask ($\mathbb{M}$) to the formula:

$$\mathbb{M}_{l-1}(x,y) = \begin{cases} 0 \text{ if } M_{l-1}(x,y) = 1 \\ -\infty \text{ otherwise} \end{cases}$$

$$X_l = softmax(\mathbb{M}_{l-1} + Q_l * K_l^T) * V_l + X_{l-1}$$

M_{l-1} is the binarized output of the resized mask predicted by the previous decoder layer; M_0 is obtained from X_0 before feeding the queries to the decoder.

4.2 Training and results

Since ViT-Adapter is a recent network, to date, does not exist any high level library (e.g.: HuggingFace), that implements it. For this reason we used directly the code from the official GitHub page of the project [26]. To train this architecture we had three main problems:

Errors in the code In the GitHub page, the researchers provide a python file to train the network. It is easily executable through a .sh file in a Linux terminal, but we encountered a

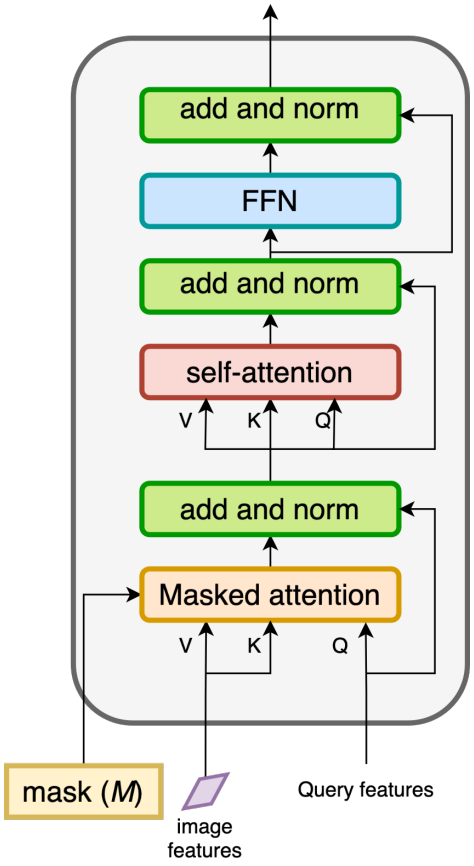


Figure 4.7: Decoder module with masked attention.

problem even tiring to fine tune the network with their example provided in Google Colab. After a quite long debug, we find out that they use a dictionary to create the network and the keys of such dictionary are strings like ”backbone.layer.l0.convolutional” but the actual keys in the dictionary do not contain the prefix ”backbone.”, thus we slightly modified the code to make it works. We also opened a pull request to inform the researchers about this.

Hardware requirements SpecialVideo, since is starting now to enter in the deep learning, has only one GPU RTX 4090 to train networks. This is more than enough to train YOLO and SegFormer but it has too few memory for training the ViT adapter with the large backbone. For this reason, we used Vast.ai [24] to rent four L40 GPUs which have 40GB of GPU's RAM each and together 289.7 TFLOPS which permit to train the adapter in a reasonable amount of time.

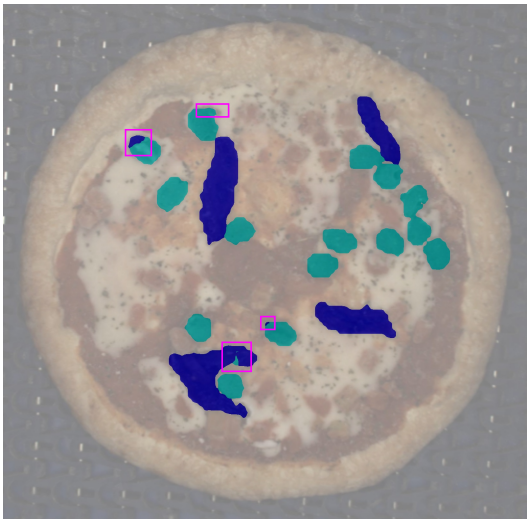
Missing of appropriate documentation This last problem was the one that wasted more time. To be able to train the network with our dataset, we had to modify several files, adding

a class to describe our dataset and modify some hyperparameters (the mean and the standard deviation used to normalize the images) which are dataset-dependent. These things are not explained anywhere and there are not comments in the code, for example, they did not indicate to modify the mean and standard deviation based on the dataset and also in different files regarding different datasets, they used the same values for these two values making the understanding of what to do quite hard. For this reason, a try and error approach was used to make all things work.

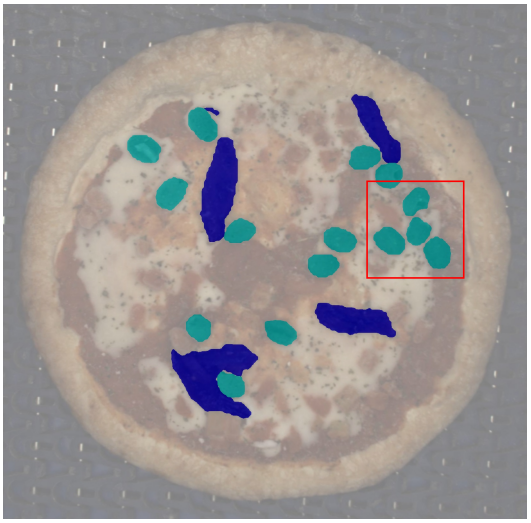
4.2.1 Training strategy

We started the training of the adapter using the proposed architecture with *beit* (a vision transformer trained to reconstruct a masked input) large as backbone and mask2former as segmentation head. First of all, we used the dataset *Dataset_80* without margheritas and without the rotation as data augmentation obtaining very low quality results probably due to lack of enough data. As second try, we went with the dataset *Dataset_160* without margheritas but with data augmentation; this second approach gave us good results achieving 87.2% mIoU (look at table 4.1 for details) and very good looking results, in particular, the most interesting thing was the fact that the adapter is able to divide in different masks objects that are of the same class and very near (without overlap) to each other, like in the example in figure 4.8b; this feature can really help the labeler as explained in the following chapter (§5). Unfortunately, we did not use this model for our tests with the labeler because this version of the adapter did not fit in the SpecialVideo's GPU.

Even if we did not have the possibility to call the labeler for other tests, we tried to fine tune also a smaller version of the adapter (beit base as backbone and mask2former as head) since this one can be fitted in the RTX 4090 and we wanted to see if it can replicate the results of the bigger one. We tried both with the *Dataset_80* and *Dataset_160* but even after a long training (4 and 2.5 hours with early stopping) the results were quite bad and are reported in the table 4.1. Moreover, the errors, visible in figure 4.9a that the adapter did are very bad for our purposes, since adjusting such errors requires a lot of time for the labeler.



(a) Test pizza segmented with beit base as backbone.



(b) Test pizza segmented with beit large as backbone.

Figure 4.8: Comparison between the two adapter models. In the left figure there are some rectangles to highlight some segmentation errors that are difficult to adjust for the labeler. In the right figure, there is highlighted a batch of olives that only the adapter large is able to segment as four different blobs.

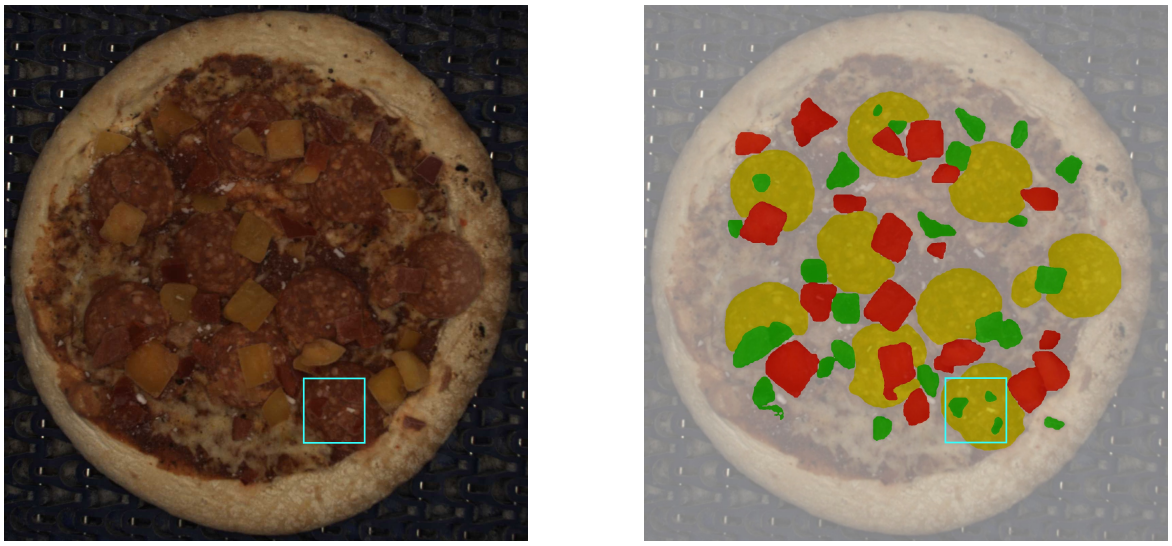
4.2.2 mIoU analysis

As aforementioned, the adapter, especially the large one, performed well on our task, the result are summed up in the table 4.1

Backbone	Dataset	mIoU on salami pizza	mIoU on olives pizza	mIoU
Large	*Dataset_160*	82.8%	90.5%	87.8%
Base	*Dataset_160*	75.8%	77.8%	80.0%

Table 4.1: Comparison between the results obtained with the base and large backbones. Note that the column *mIoU* consider also the results obtained on the background class that are 98.0% for the large model and 97.1% for the base model.

To our mind, the adapter with beit base backbone, even if it can be used on the RTX 4090 of SpecialVideo, is not usable, it has a low score (if compared with SegFormer or with the adapter with beit large) and this generates errors that are quite difficult to adjust. Writing about the bigger model, it is a pity that it cannot be used in the SpecialVideo's GPU since it achieve very good results in terms of mIoU and it performs particularly well on difficult parts like near blobs of the same class and on red peppers (Figure 4.9b) which are quite difficoult to see even for humans.



(a) Test image. (b) Segmentation of the image.

Figure 4.9: The adapter with beit large as backbone can segments parts that are very difficult to see for humans.

Chapter 5

Results

First of all, we will remind the purposes of this project to make the comprehension of this chapter easier. The main aim of the project was to speed up the labeling process in the context of semantic segmentation in industrial applications. The time saving comes principally from two aspects: the performances of the networks and the user-friendliness of the whole system. To test the former, we took 20 never seen images (10 with olives and anchovies and 10 with salami and peppers) to make inference with all the tested networks for a fair comparison; for the latter we tested all the main tasks, as described in §2.1, and then we asked to the labeler to adjust, for each network, the 20 proposed segmentation masks.

The tests were done in the following way: we call in SpecialVideo Federico for one day. He can be considered an expert pizza labeler since he worked for them for months and his task was to segment toppings on pizzas using SAM on CVAT online. He did for us several tests – always using the same 20 pizzas for a fair comparison –, first of all using SAM online to obtain a time base line, then using SAM with CVAT offline and lastly using several of our best models. We did not test the ViT-Adapter because in January, when the labeler had a free day to work for us, we were not ready to test it, but this was not a big problem because, as explained in the former chapter, the adapter with beit large as backbone cannot be fitted in the SpecialVideo's GPU while the adapter with beit base as backbone has not good results.

First of all, since the models run locally, we will analyze the improvements of using CVAT offline and then we will use the times used for the labeling on CVAT offline with SAM as baseline to show the improvements given by the use of our models.

5.1 CVAT offline to remove lag and delays

As discussed in the second chapter (§2.2), using CVAT offline can improve the performances of almost all activities that are useful for the labeling (e.g.: loading images, export the annotations, etc...). Using an offline version of CVAT comports two main advantages:

1. We can choose any SAM model (we used the huge one for better performances);
2. We do not have anymore a communication with a server which can speed up the labeling process removing the delay of the Internet communication.

Even if also CVAT online uses the huge SAM model, we suppose that they uses an old version of the weights since the labelers said to us that our offline version is able to segment better and so he was able to do the work in less time (i.e.: the labeler used less clicks to obtain the same mask on a topping). Just with this first improvement, we were able to pass from 57 minutes to 37 minutes, which is a time saving of 35%, see table 5.1 for more details.

5.2 Improving labeling time with fine-tuned models

After the first test, we proceeded with the tests using our models to help the labeler. The pipeline is like the previous one, but, after having uploaded the images on CVAT offline, we used the *automatic annotation* tool of CVAT to do inference with our models on the test images. Then, the labeler did the test using SAM and other CVAT tool to adjust the labels. Moreover, is worth to mention the fact that we chose to upload the annotation on CVAT as polygons, not as blobs on the image. The main difference is that on the polygons, is not possible to use the brush and rubber tools, but instead CVAT permits to simply modify the points that compose the polygons which is a faster approach to modify the annotations.

5.2.1 Loading fine-tuned models on CVAT offline

To start this second batch of tests, we needed to upload the models on CVAT. Due to time limitations (the labeler was available only for one day), we chose to do the tests with a subset of our models, in particular we used YOLO, YOLO + SAM and SegFormer *b4* (yellow bar in figure 3.2) all three trained on *Dataset_80* and SegFormer *b4* trained on *Dataset_160* (purple bar in figure 3.2).

To load a model two files are necessary:

yaml configuration file The first step to load a model on CVAT is to create a configuration file in yaml format. This file is composed by three main parts:

1. The initial part is useful to specify to CVAT how to use the model. It is mandatory to specify the labels that the model is able to identify (i.e.: Olives, anchovies, salami, etc...), the name of the file handler and give a name to identify the model on CVAT;
2. The second part is the creation of a Docker file. Here one can chooses the starting docker image (as example plain Ubuntu or some pytorch images to not install pytorch and cuda manually), gives a name to the Docker image, and installs all the system packages/python library required to run the handler file;
3. In the last part of the configuration file is possible to set if one wants or not to use the GPU (if available on the system).

handler file The second file is an handler for the http request generated by CVAT. Following the examples in the repository of CVAT we chose to use python. In this file two functions are required, the first one called *init_context* is used to instantiate the model loading the weights, while the second one is the *handler* which receives the images by CVAT and so can be used to made the inference and to generate the http response to send the labeled images to CVAT.

Our code is public available into the SpecialVideo GitHub page [17].

5.2.2 Labeling time analysis

As aforementioned, just switching from the offline to the online version of CVAT we were able to save 35% of time. But even better results were obtained using the models to infer the segmentations and then leaving to the labeler only the duty to check and adjust the segmentations, obtaining with the best model (SegFormer trained with *Dataset_160*) a time saving of 66.6% with regards to CVAT online (that was the method used by SpecialVideo before this project) and 48.6% over the use of CVAT offline with only SAM, all the results are summed up in the table 5.1.

	SAM online	SAM Offline		YOLO 80imgs	YOLO+SAM 80imgs	SegFormer 80imgs	SegFormer 160imgs	
SAM online	0	-35%	olives and anchovies: -33.3% salami and peppers: -35.4%	-49.1%	-56.1%	-56.1%	-66.6%	olives and anchovies: -55.6% salami and peppers: -68.8%
SAM Offline	-	0		-21.6%	-32.4%	-32.4%	-48.6%	olives and anchovies: -33.3% salami and peppers: -51.6%
YOLO 80imgs	-	-		0	-13.8%	-13.8%	-34.5%	
YOLO+SAM 80imgs	-	-		-	0	0	-24.0%	
SegFormer 80imgs	-	-		-	-	0	-24.0%	
SegFormer 160imgs	-	-		-	-	-	0	

Table 5.1: The table shows the improvements in percentage about the time used by the labeler to segment the 20 test images. Some cells are splitted to show the differences between the overall time saved and the time saved for a specific format of pizza. Note that the times are negative because they represent a time saving.

It is also very interesting to discuss about the following observations:

Differences between SegFormer trained on *Dataset_80* and on *Dataset_160* We noticed that there is a quite big improvement in the time saving between the two tested SegFormer models if compared to the mIoU scores. We registered a 24% time saving with only a 0.7% difference in the mIoU score as reported in figure 3.2. Trying to understand this difference we looked at the generated masks and also asked to the labeler. The answer was that the model trained on more images is more able to divide in different instances masks of near, not overlapping, objects with the same class, the same thing we discussed in the previous chapter about the bigger adapter model. This thing really helped the labeler since with SegFormer the only adjustment needed was to split masks in two or three different instances, while both the classification and the pixel-level accuracy of the masks were already perfect.

Performances of YOLO and YOLO+SAM Both YOLO and YOLO+SAM models performed worse than SegFormer, but they could be useful is some situations. The errors committed by these two models are the exact opposite of the error committed by SegFormer. Thanks to the object detection capabilities of YOLO the two models automatically perform instance segmentation dividing in different instances near objects of the same class (even if they are overlapped!). The problems in this case were the missing of some instances, the miss-labeling of some masks and the precision of some masks that were not perfect. Overall these two models helped less the labeler, but as we will discuss in §5.3 one can use together one of this two models and SegFormer to achieve better results.

Differences between the two formats Lastly, we investigated if there are differences between the time saved for the two typologies of pizza. As can be seen in the first two rows of the table 5.1, the pizzas with salami, red peppers and yellow peppers benefits more of our improvements. This can be explained by the fact that for both humans and neural network this formats is more difficult to segment and thus having cues by the network about the shape and location of the ingredients generates a bigger time saving, for example, using SAM online the labelers used 48 minutes to segment the 10 salami pizzas and only 9 to segment the 10 olive pizzas, while exploiting SegFormer trained on *Dataset_160* he used 15 minutes per the salami pizzas and 4 minutes for the olive pizzas.

SegFormer trained on *Dataset_20* We did not test this model due to too short availability of the labelers but looking at figure 3.2 it could be very useful for our pipeline. It reaches mIoU scores comparable with the tested model and we are pretty sure about that the time saving using it will be almost equal to the SegFormer model trained on *Dataset_80*. This result is such interesting because we can help the labeler – and so use less time in the labeling phase – even after only 20 images for each kind of pizzas.

5.3 Identification of the problems and possible solutions

As already mentioned, even if we were happy about the performance boost of our pipe line, we'd like to overcame to the issues of our models. Since SegFormer is very good in classifying the toppings and has good performances on all the ingredients, while YOLO's strength is to be able to perform instance segmentation, we started to think on how to merge these models.

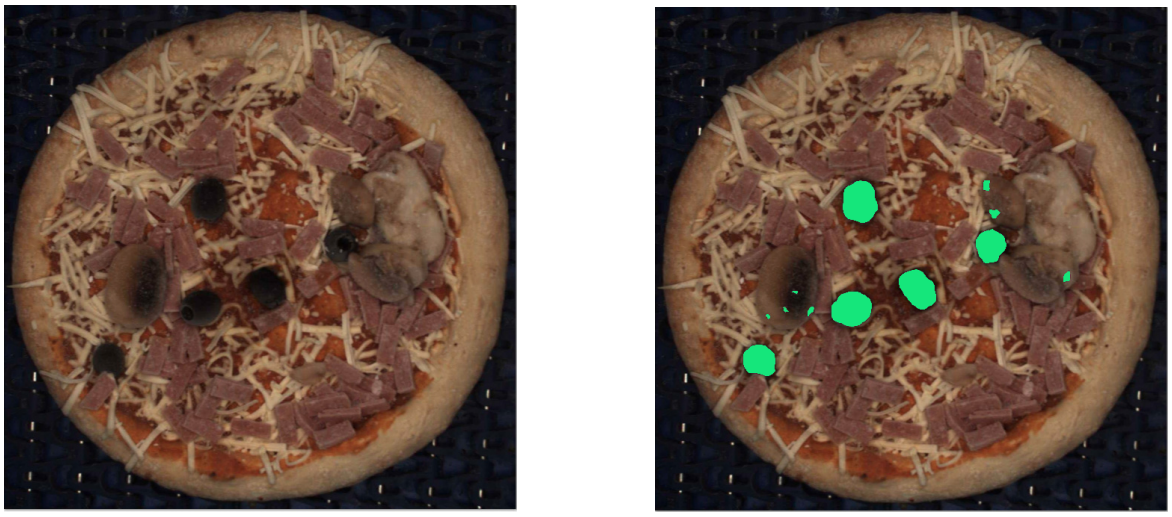
Looking at figure 2.5, it is clear that YOLO is very good in segment Salami and Olives which also are the only two ingredients that suffers from overlapping with objects of the same class, also the majority of near, not overlapping, blobs of the same classes are either salami or olives. Fortunately, when one wants to perform auto labeling, CVAT permits to choose on which labels perform the labeling. For this reason we tried to make the inference on salami and olives with YOLO+SAM followed by another inference on the remained classes with SegFormer. The results were very good, we were able to maintain the precision of SegFormer on difficult classes like the peppers, but at the same time we performed instance

segmentation on the two classes that benefit more of this capability of YOLO.

Unlikely, we did not test the time saving of this merge of models since we thought about this possibility the day after the tests thinking on how to work on the problems that the labeler identified. Moreover, this is not a general solution, since it depends on the data, for example, if YOLO had bad performances on salami and olives, this workaround would not have been possible, but this demonstrates that in same cases this technique can be helpful.

5.4 Transfer capabilities on new pizzas with a subset of already seen toppings

Lastly, we tried to make inference with SegFormer trained on *Dataset_160* on pizzas with one known ingredient and the others unknown. The aim of this test is to help the labeler with new flavours of pizza if one (or more) topping is in common with former pizzas. This is the case of capricciosa pizza, which has mushrooms, cubes of ham and **olives**. As figure 5.1 shows, the network is able to rightly identify all the olives with some little false positives. This is not a big issue, first of all, they can be deleted easily by the labeler in few seconds or, another solution, could be to check the area of the proposed blobs and deleting them algorithmically if the area is too big or too small with regards to the mean area of the class.



(a) Capricciosa pizza

(b) Capricciosa pizza with segmented olives

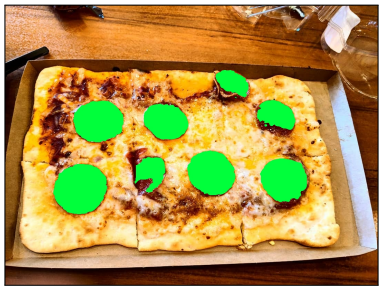
Figure 5.1: Example of difficult parts to segment. On the left some olives cover the anchovies, while on the right there are some red peppers on a salami slice which are difficult to see even for a human.

We also tried SegFormer trained on *Dataset_160* on pizzas with different image distributions (i.e.: not acquired by SpecialVideo), in particular we used chatGPT 4.0 to generate a pizza with black olives and we take a salami pizza from a public dataset available on Kaggle [18]. Also in this case SegFormer performed very well (see figure 5.2), and this is very important since demonstrates that the network learned the semantic of the toppings independently from the context, permitting to help the labeler to hypothetical future tasks with pizzas from different clients or event with tasks that have these ingredients on different kinds of food as showed in figure 5.3.

(a) Pizza generated by chatGPT 4.0

(b) Pizza generated by chatGPT 4.0 segmented

(c) Salami pizza from Kaggle

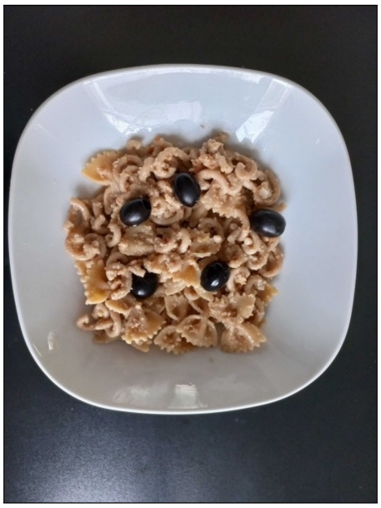


(d) Salami pizza from Kaggle segmented

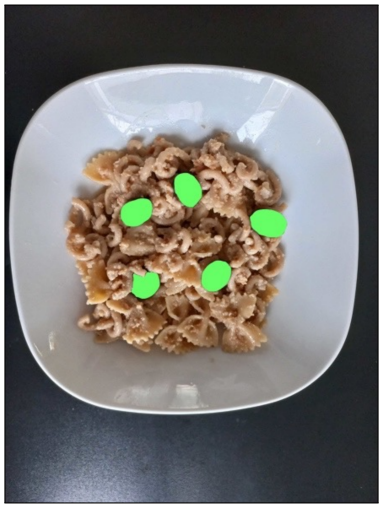
Figure 5.2: Examples of segmentation of different pizzas

(a) A salad with black olives

(b) A salad with black olives segmented



(c) Pasta with black olives



(d) Pasta with black olives segmented

Figure 5.3: Examples of segmentation of olives on different kinds of food. The used network was trained only with the olives and salami pizzas.

Chapter 6

Conclusions and future work

In this last chapter we will summarize our work and then we will propose some ideas for future developments of it.

With our project we obtained significant improvements to the segmentation pipeline. With our best model we reached a time saving of 66% over SAM used online and 48.6% over SAM used offline. Furthermore, we are sure that even after 20 images segmented only with SAM our framework can help the labeler saving time. Moreover, it is important to highlight that this pipeline is obviously extensible to every task and, especially for industrial applications, we think that we can obtain results comparable or even better than the ones reported in this thesis. The segmentation of pizza topping is quite a challenging task: there could be occluded ingredients, ingredients with the same color of others (like the red peppers with the salami) and other difficult situations. Instead, other industrial tasks could present less problems, for example we recently started to apply our work to a brand-new task. We had to segment some bottles (more or less 200 each image) placed straight or upside-down in a box, with a top view of the scene. This task is perfect for neural networks since they are robust to the light and prospective change of the bottles in the image and here we had way less problems if compared with pizzas. Just using YOLOv8 nano + SAM trained on few images we obtained results that can really speedup the process, here the only problem we encountered was the presence of some false negatives and false positives, but clearly we can fix this using better thresholds on YOLO or training a better model like SegFormer.

6.1 Future work

We also thought about how this system could be improved. Our main idea was to help the labeler since the beginning of the labelig process and this can be done with two approaches:

Unsupervised learning The unsupervised learning technique permits to train a neural network without having the labels. For this reason they can be used at the very beginning of our pipeline to help the labeler since the beginning of his work. SpecialVideo already explored a little this field with other thesis obtaining promising but not usable results. But, we think that using the results from the thesis of Cristian Davide Conte, the intern that worked for SpecialVideo before me (there is not the citation of the work since it is not available to date), with more recent unsupervised networks like DINOv2 from Meta [16] good and usable results can be achieved.

Using Large Language Models (LLMs) Another approach is to use LLMs to assign labels to unlabeled segmentation masks. For example, an idea of a pipeline could be the following one: SAM can be used to identify masks on a image in automatic way (see figure 1.2), then each blob can be passed to a multi-modal LLM to assign a label to it. We think that this approach could be very effective for our purposes.

Bibliography

[1] Z. Chen, Y. Duan, W. Wang, J. He, T. Lu, J. Dai, and Y. Qiao. Vision transformer adapter for dense predictions, 2023. arXiv: 2205.08534 [cs.CV].

[2] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar. Masked-attention mask transformer for universal image segmentation, 2022. arXiv: 2112.01527 [cs.CV].

[3] B. Cheng, A. G. Schwing, and A. Kirillov. Per-pixel classification is not all you need for semantic segmentation, 2021. arXiv: 2107.06278 [cs.CV].

[4] CVAT. URL: www.cvat.ai.

[5] CVAT download documentation. URL: https://opencv.github.io/cvat/docs/administration/basics/installation/.

[6] Docker's web page. URL: https://www.docker.com/.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[8] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners, 2021. arXiv: 2111.06377 [cs.CV].

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. arXiv: 1512.03385 [cs.CV].

[10] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. 25, 2012. F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors. URL: proceedings.neurips.cc/paper%5C_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[12] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection, 2018. arXiv: 1708.02002 [cs.CV].

[13] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: hierarchical vision transformer using shifted windows, 2021. arXiv: 2103.14030 [cs.CV].

[14] F. Milletari, N. Navab, and S.-A. Ahmadi. V-net: fully convolutional neural networks for volumetric medical image segmentation, 2016. arXiv: 1606.04797 [cs.CV].

[15] opencv's web site. URL: https://opencv.org/.

[16] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. Dinov2: learning robust visual features without supervision, 2024. arXiv: 2304.07193 [cs.CV].

[17] Our CVAT project. URL: https://github.com/Specialvideo/cvat-sv.

[18] Pizza toppings challenge on Kaggle. URL: https://www.kaggle.com/datasets/michaelbryantds/pizza-images-with-topping-labels.

[19] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021. arXiv: 2103.00020 [cs.CV].

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: unified, real-time object detection, 2016. arXiv: 1506.02640 [cs.CV].

[21] SAM web page (30/01/2024). URL: https://segment-anything.com/#:~:text=What%20type%20of%20prompts%20are%20supported%3F.

[22] SpecialVideo. URL: www.specialvideo.it/en/.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014. arXiv: 1409.4842 [cs.CV].

[24] Vast.ai website. URL: https://vast.ai/.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023. arXiv: 1706.03762 [cs.CL].

[26] ViT-Adapter GitHub repository. URL: https://github.com/czczup/ViT-Adapter.

[27] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo. Segformer: simple and efficient design for semantic segmentation with transformers, 2021. arXiv: 2105.15203 [cs.CV].

[28] YOLOv8 documentation. URL: https://docs.ultralytics.com/modes/#introduction.

[29] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai. Deformable detr: deformable transformers for end-to-end object detection, 2021. arXiv: 2010.04159 [cs.CV].

Acknowledgements

I'd like to start the acknowledgments saying that I am really grateful to SpecialVideo that accepted me as intern and trusted me during the whole period. I am glad for having found such a nice place without pressures, they helped me to develop my skills as both computer scientist and AI engineer. In particular, I'd like to thank Giuseppe, who followed me in the last months, he inspired me to be more curious, not only in my field but in general for giving me back the curiosity to explore also outside the IT field. I really appreciate also the trust by Rossana who was very enthusiastic about my project and asked me to write about it in the web site of SpecialVideo. In general, all the people inside the company were very nice to me and demonstrated that nice places of work do exist. I want to say thank you also to my supervisor prof. Luigi Di Stefano who, even if he's full of commitments, helped us to find suitable architectures for our work and also he founded time to explain to us how to continue our collaboration through a possible PhD. Following, the acknowledgements will be in Italian to be readable by every Italian person.

Come prima cosa vorrei ringraziare Elena, più che una migliore amica, una sorella, per essere sempre disponibile nei momenti di bisogno e per riuscire sempre a distrarmi da pensieri ed ansie quando siamo insieme. Ringrazio anche Andrea ed Elia per essere dei veri amici, senza di loro e senza Elena non so come avrei superato un brutto periodo passato qualche mese fa, probabilmente senza loro tre non mi sarei riuscito a laureare in questa sessione e non sarei stato meglio così velocemente.

Voglio ringraziare anche Elisabetta e Francesca, per aver creato un bellissimo clima in casa, cosa per nulla scontata quando ci si ritrova a vivere con persone mai viste prima, sicuramente questo ha reso gli ultimi mesi di studio più sereni.

In fine, ci tengo a ringraziare la mia psicologa Claudia per il supporto durante i momenti più complicati. Agli eventuali lettori vorrei dire di non aver paura di farsi aiutare, in Italia spesso andare da psicologi e psichiatri non è visto di buon occhio, si pensa che siano figure che stanno solo a rubare soldi o che siano riservate solo "ai pazzi", ma vi assicuro che ciò non è assolutamente vero, il periodo universitario, e in generale quello scolastico, possono essere molto provanti dal punto di vista mentale ed emotivo, non abbiate paura di affidarvi a professionisti se ne sentite il bisogno, molto probabilmente è la scelta migliore che potete fare per voi stess*.