Microbes detection in microfluidics-based microscopy imaging

Thomas Sadigh Rezvani

end of Masters' degree internship

Spring-Summer 2023

report submitted on November, 10th, 2023

internship supervisors : Matthias Függer, Thomas Novak
school supervisor : Raphaël Fournier-S'nhiehotta

Acknowledgements

I wish to thank my internship supervisors Matthias Fuëgger and Thomas Nowak for giving me the immense opportunity to join the team at LMF. I also thank the team at LMF for welcoming me.

I wish to thank my school supervisor and head of the Master's degree I am enrolled in, Raphaël Fournier-S'niehotta, whithout whom I couldn't have done the internship at all. I also thank my machine learning professor Michel Crucianu for his teachings and my deep learning assistant professor Nicolas Audebert for his availability and encouragements.

I wish to thank my mom, Elvi, Alejandra, Karl... For your encouragements and much more.

Table des matières

1 Introduction

The following is an internship report on the research internship I had the opportunity to do at “Ecole Normale Supérieure” of “Paris-Saclay” in 2023. I joined the Formal Methods laboratory (“LMF”) within which I joined a sub team that works on modeling microbiological phenomenons (example : the growth of a colony of bacteria depending on its environment). My supervisors are Matthias Füegger and Thomas Nowak. The lab has a microfluidic chip which they use with a microscope to take images of microbes as they pass trough the chip in a stream or not depending on wether the pump of the chip is pushing or not. The images are taken with a small camera that is attached to the microscope to record what is seen in the chip under the microscope. We give a few definitions which we will use in the further developments :

We define Computer Vision (“CV”) as such for the purpose of this report : CV is an interdisciplinary field which aims to retrieve information from digital images or videos (Wikipedia contributors, 2023a). It includes methods founded in Artificial Intelligence (“AI”) but not only. Particularly : signal processing has historically contributed to the field. AI based methods include ones which use deep *Neural Networks* (“NN”). Deep NNs are models which loosely mimic nerve connections in the brain. The Multilayer Perceptron (“MLP”) is a historical example. From a mathematical perspective, they are built on the composition of functions which present a non linearity operation. They have an architecture with layers of neurons. Each neuron represents a function with a non linearity property which enables projecting the data into a higher dimensional space in which classes can be linearly separated.

In CV in particular, deep *Convolutional Neural Networks* (“CNN”) are used, among other reasons, to take into account the *locality* property of the information that images have and share parameters of a neuron across the entire entry input (or tensor). To illustrate the motivation to use a CNN because of locality : imagine a picture of a bird. The entry image will have pixels (for any given color channel with pixels described as numbers in the array corresponding to the given color channel) which define a beak located close to pixels which define bird eyes. We want a model to take into account how a bird’s eyes and beak are located near each others in particular in the way it makes a decision regarding the image. A convolution achieves that in the following manner : if the convolution of this image with a filter produces some given activation. This activation depends on a "neighborhood" of pixels which the filter takes into account, at each convolution operation, in the spatial dimensions of the tensor it takes the shape of. We will mention the further motivations why to use a CNN later in the development.

This brings us to defining what is a *tensor* in a CNN. For the purpose of this report we will understand tensors simply as three dimensional arrays of numbers with the shape :

$$H \times W \times F$$

with H, W the spatial dimensions and F the feature dimension. We use them to describe the input images (the color channel being a special case of the feature dimension), CNN layers, filters and any intermediary or ultimate CNN operation output within the architecture. Within CV, a problem that has been studied is object detection which we define as a machine learning problem which has two components : one is to detect the presence of an object that's not background (background information is not directly analyzed albeit learned). The other one is *multi-class classification* of the detected object. Multi-class classification is the problem of assigning a label to an instance with the constraint that an instance can only have one out of the set of labels. In object detection, an image can have zero or several objects. Object detection as a *supervised learning* task can be solved with a CNN that is trained with pre-labeled images.

The actual subject of the internship is to detect microbes in microscopy imagery. An other problem which depends on the main one is to track microbes lineages based on what is captured on film. As the internship advances it turns out that the microbes tracking problem is not secondary and actually from the perspective of the lab, the greater problem. We attempt to solve both problems but are only able to propose a method for the first problem eventually. The problem having evolved throughout the internship, we wish to explicit the problem we aim to answer here : How to detect and classify microbes Chlamydomonas reinhardtii ("C.

reinhardtii") and Escherichia coli ("E. coli"), which pass through a micro-fluidics chip in microscopy imagery ? (How to use the approach to the previous problem to build a microbes tracker on top of it ?)

We choose a one-stage model called YOLO ("You Only Look Once") which detects objects and retrieves in particular the midpoint of a detected object, and the width and height of the bounding box that can be drawn around it to label it. We use transfer learning and fine-tuning to train the weights of this CNN based model with the aim of achieving near state-of-the-art results but fail at achieving satisfactory results. Still, our method may be of interest in CV applied to microbiology where labeling has a cost in terms of time, biology machines, with the requirement to have the data labeled or double checked by an expert.

2 Materials and Methods

2.1 Hypotheses and Constraints

We will state in details in this section our constraints and hypotheses in regards to our goal. We aim to solve the problem stated in introduction which is to classify the microbes we study. Our constraints that we identify are the following :

1. In the context of the micro-chip, all objects are to be detected on the ***same level***. The background would be the back glass wall of the chip but the chip is by design so thin so that the microbes that pass through it are on almost the same level.
2. How the microbes show on camera, they are relatively ***invariant to the angle*** they appear at. E. Coli is elongated in shape so if it was seen from above or below (not from the side) the long body could disappear thus changing the overall shape. This scenario, however, seems to be made less likely because of the microchip which ensures the microbes are flowing through a thin passage.
3. The microbes may reproduce or be ***clustered*** very close to each others and overlap.
4. Given the zoom capability of our microscope, the microbes will be quite ***small***, E.

Coli being really small.

5. The objects may appear moving or not
6. The background doesn't change
7. The images are withdrawn from a camera so they have a sequential property to them

What problems may arise from the constraints or hypotheses ? What hypotheses can we make use of, to define our method ?

The fact, that the objects are on the same level doesn't pose problem and in fact, makes the problem easier. One way to use this to our advantage is to have training data whose objects are also on the same level. Actually, in the transfer or fine tuning datasets, the objects are on the same level anyway within one image. They are not, however, necessarily on the same level from one to one other training image in the fine-tuning dataset. We still don't view this as a problem as we're keen to have a model that generalizes well even if the constraints changed within some extent. We'll elaborate on this in section 3.1.1. iii. The objects being invariant to the angle how they appear is also making the problem easier. We choose to monitor if this property holds during experiments but a CNN trained with enough training data would learn high level enough semantic information about the objects solving the problem of varying objects' angles anyway.

The scenario where the objects appear clustered together is of importance for our problem since both C. Reinhardtii and E. Coli often appear clustered together. To our knowledge, it appears this is a problem if the method to match anchor boxes to target bounding boxes uses Intersection over Union ("IoU"). This problem has been addressed with a method called Optimal Transport Assignment ("OTA"). Yolov7 uses a method called simOTA which is derived from OTA. We consider for now, that this could solve the problem.

Regarding, the fact that the microbes appear small, a CNN will output a feature map of low spatial resolution. Then, small object may "disappear" or become merged with other objects or background. An FPN used in the neck of the architecture of the chosen CNN based model could, to some extent, solve this problem but one-stage object detectors are shown to be less robust than R-CNN based models.

The fact that the microbes appear moving may decrease their resolution in images, which may make it more difficult for the model to successfully detect them. This may be solved by including in the training data, images with moving objects. An other approach would be to apply to some of the training images, data augmentation that blurs the objects.

If these problems seem like they can be handled by a CNN based object detector, what if we have scenarii that combine several of these problems ? For instance, a case of a small moving group of clustered objects ? We hypothesize that the objects would be very difficult to detect in such scenarii.

2.2 Motivation to choose Yolov7

Based on the power of CNNs to learn a multitude of high level features, we choose an object detector which is a model based on a CNN backbone. We choose a one-stage object detector, namely Yolov7.

2.2.1 Convolutional Neural Networks

i. The convolution operation

We phrase a simplified definition of the convolution operation which nonetheless suffices for the purpose of this report and will be the definition used in the developments. A convolution is a mathematical operation between two functions which, after one of the functions is reflected about the y-axis and shifted, calculates the integral of the product of the two functions (Wikipedia contributors, 2023b). It is formalized as so :

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z})g(\mathbf{x} - \mathbf{z})d\mathbf{z}$$

With $*$ the convolution operator, $f, g : \mathbf{R}^d \to \mathbf{R}$. In CNNs, we use a discrete convolution in three dimensions. The formula involves three sums : two in the two spatial dimensions and one in the feature dimension of the input. One may think of function f as the input image and g as the kernel (Note : the CNN is to learn the coefficients of the kernel in the backpropagation operation). With H, the representations, V, the kernel and X, the input, indices a and b, the spatial dimensions, c, d, the channel and kernel respectively, one formalization is (Zhang et al., 2023a) :

$$[\mathsf{H}]_{i,j,d} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} [\mathsf{V}]_{a,b,c,d} [\mathsf{X}]_{i+a,j+b,c}$$

With the convolution operation, CNN are able to retrieve feature maps of very long feature dimensions. This means that the model learns a multitude of complex combinations of different features which it is able to look for in an image. This is motivation to choose a model which uses convolutions in the backbone.

2.2.2 One-stage vs. two-stage object detectors

Object detectors are described as falling into one of two groups : one-stage or two-stage. We didn't go through all models to make a choice. We had a broad outlook on the different models that were known to us and chose to study in details : Yolov1 (version 1 of Yolo) and R-CNN. We had a curiosity to understand Yolo as it seemed to emerge a fast model which was interesting to us because our base signal was video. We contemplated the possibility to run a model on video and have live object detection and classification, hence the need for speed.

The one-stage group of models also includes EfficientDet which is based on an EfficientNet backbone. Yolov7 published results indicating it to be faster than EfficienDet by roughly +1200% on Ms Coco dataset ! We didn't look further into EfficientDet, because, on top of the speed argument, it seemed to us that Yolo in general had more useful resources publicly available such as rich Github repository, communities and discussions.

We studied R-CNN because it was studied as part of my curriculum at CNAM in the context of a graded project. Based on our analysis, Yolo has a straightforward architecture and seemed easier to implement. We even attempted to implement YoloV1 from scratch. But one-stage object detectors struggle to detect small objects because the tensor output by the backbone has taken the whole image as input. The models from the R-CNN family give a *region of interest* of the image as input to the backbone increasing the resolution of the output tensor.

R-CNN is composite and heavier in terms of computations though. Two-stage object detectors like R-CNN and its successors still may detect small object better than Yolo but we choose Yolov7 eventually, notably because we deem its use of an FPN in the head, a solution to detect small objects. We didn't further look into the improvements made to R-CNN with the following improved models of the family.

2.2.3 Key concepts of Yolov7 model

We only have a limited understanding of the Yolov7 model in all its details. However, we briefly state here fundamental concepts of Yolov7 (Wang et al., 2022) to help understand what the model does.

i. Object detectors

As an object detector, the model is going to do two things :

1. localize an object, if the image has one of the relevant objects, by outputting coordinates of bounding boxes that capture one of the possibly many objects,
2. estimate the probability that the object is of one of the classes. This is done for all the objects in the input image.

In this stage, the coordinates of the box around the objects are predicted and the class of the object too. Yolov7 in this context belongs to the family of anchor-based object detectors.

ii. An anchor-based model

The base Yolov7 model which we used, divides an image into cells. Based on this "grid", each cell is responsible to look for an object whose center lies within its boundary, with three different ***anchor boxes*** as bounding box candidates in a sense. Each of the three detection heads has its own grid based on the spatial dimensions of the feature tensor it uses for prediction (see the three blue tensors in figure 1 at the end of the forward pass). This set of grids is a Yolo application of the general concepts of anchor box *scale* (ZHANG et al., 2023b) and *stride*.

Each detection head uses its own set of three different anchor boxes (each of a specific *aspect ratio*) for every cell of the grid. These anchor boxes, we view in a way, as mirroring the possible shape of the object they look for in the input image. Just to enable the reader to check what we attempt to explain : if our detection head outputs three tensors of shapes : $20 \times 20 \times f$, $40 \times 40 \times f$, $80 \times 80 \times f$, this would mean that for a given image : $3 \cdot 20 \cdot 20 + 3 \cdot 40 \cdot 40 + 3 \cdot 80 \cdot 80 = 25200$ anchor boxes are produced.

These anchor boxes priors are pre-selected so that the algorithm is computationally optimal in the following way : only anchor boxes near the ground truth bounding box are considered in the learning (they are the anchor boxes associated with "*center priors*"). Within this subset of anchor boxes, only the ones which the transformed bounding box coordinates regression can apply to are selected.

iii. Matching anchor boxes to ground truth bounding boxes

At this stage, a matching algorithm is applied. A common algorithm for matching is based on the *Jaccard index*. Applied to pixels in an image, the *Jaccard index* is called *Intersection Over Union* ("IoU"). In Yolov7 however, the matching algorithm is based on an *Optimal Transport Assignment* ("OTA") called "simOTA" from previous works on Yolo (GE et al., 2021). It is used instead of IoU, in order to better deal with clustered or occluded objects (Note this is particularly relevant for our problem). This algorithm does an optimization approach to matching an anchor box to a ground truth bounding box for each image. We don't elaborate more on how simOTA works but note that the matching stage can also classify some candidate anchor boxes as "*background*", which, in our understanding enables the model to **learn** what is background too.

Once matching is done, we can label the class of the anchor box according to the matched GT bounding box. What can be calculated from there as well, is the *offset* of the

GT bounding box relative to its matched anchor box. The offset can be formalized and quantified. A transformation can be applied to the offset formalization so that the matched anchor box coordinates can be better applied regression.

This leads us to the Yolov7 loss (see Hughes, 2022a). It has three components :

1. The objectness loss score which is calculated using binary cross entropy. This component uses a specific IoU called "complete IoU" in the following way : if the predicted bounding box has a matched GT bounding box, complete IoU is calculated. If not, complete IoU is zero. On this basis, Binary Cross Entropy is calculated between predicted and GT objectness (code here[1]).
2. The bounding box coordinates loss which is calculated between all candidate anchor boxes and their matched GT. This component is a regression problem which minimizes the average difference between predicted bounding boxes and their matched GT on the basis of complete IoU (code here[2]).
3. The classification loss which is calculated using binary cross entropy between the predicted class and GT. This component being neither specific to Yolov7 nor new (code here[3]).

Having analyzed the loss function of Yolov7, we can examine what architecture is going to be used in the forward pass and backpropagation.

2.2.4 Yolov7 architecture

Something to have in mind is that the Yolov7 model has nearly 37 million parameters. This is to say the architecture is deep and complex. For the purpose of this internship report we will only attempt to describe some aspects of the architecture.

One of the new ideas brought with Yolov7 is a layer aggregation network block called Extended Efficient Layer Aggregation Network ("E-ELAN") block. We will explain what is an ELAN block but not what is E-ELAN because, in the main Yolov7 model which we

1. https://github.com/WongKinYiu/yolov7/blob/main/utils/loss.py#L485
2. https://github.com/WongKinYiu/yolov7/blob/main/utils/loss.py#L468-L469
3. https://github.com/WongKinYiu/yolov7/blob/main/utils/loss.py#L479

used (Yolov7 has several different models, other than the main one), only ELAN blocks are implemented. E-ELAN blocks are only implemented in the yolov7-e6e.yaml which is the architecture of a deeper Yolov7 that we didn't use.

The architecture is in the cfg/training/yolov7.yaml [4] file. The file shows that the backbone is made of 51 layers (counted from layer 0) and includes four ELAN blocks. The head is from layers 52 to 105 and has three detection heads based on a Feature Pyramid Network ("FPN") with three levels in the backbone.

The four ELAN blocks of the backbone are at layers 4-11, 17-24, 30-37, 43-50 [5] . We can visualize the structure of an ELAN block with the diagram from the paper (see figure 2).

The ELAN blocks follow after a down sampling block (called "MP" in the .yaml file) which for now, we oversimplify as either applying a $3 \times 3 \times \mathsf{f}$ kernel (with a stride of 2) or $2 \times 2 \times \mathsf{f}$ max pooling operation depending on what gradient descent deems optimal to minimize the loss function. This means that for the output of either a $2 \times 2 \times \mathsf{f}$ max pooling window or a convolution with a $3 \times 3 \times \mathsf{f}$ kernel, the next operation will be what the ELAN block applies.

If the output of the down sampling block is of size : $\mathsf{h} \times \mathsf{w} \times \mathsf{c}$: the first operations of the ELAN block are to apply point wise convolutions with two different parallel $1 \times 1 \times \mathsf{c}'$ convolution layers. The base convolution layer in Yolov7 is described in the class Conv() in the common.py file [6]. It notably uses Batch Normalization right after the convolution and then uses as non linearity, the Silu() by default. This enables applying on the result of the Max Pooling block an operation which, in our thinking : produces an added non linearity to the region captured by the last operation of the max pooling block. The two point-wise convolution layers, in and of themselves, produce two hidden representation "stacks" each of length c'.

4. https://github.com/WongKinYiu/yolov7/blob/main/cfg/training/yolov7.yaml
5. https://github.com/WongKinYiu/yolov7/blob/main/cfg/training/yolov7.yaml
6. https://github.com/WongKinYiu/yolov7/blob/main/models/common.py#L99

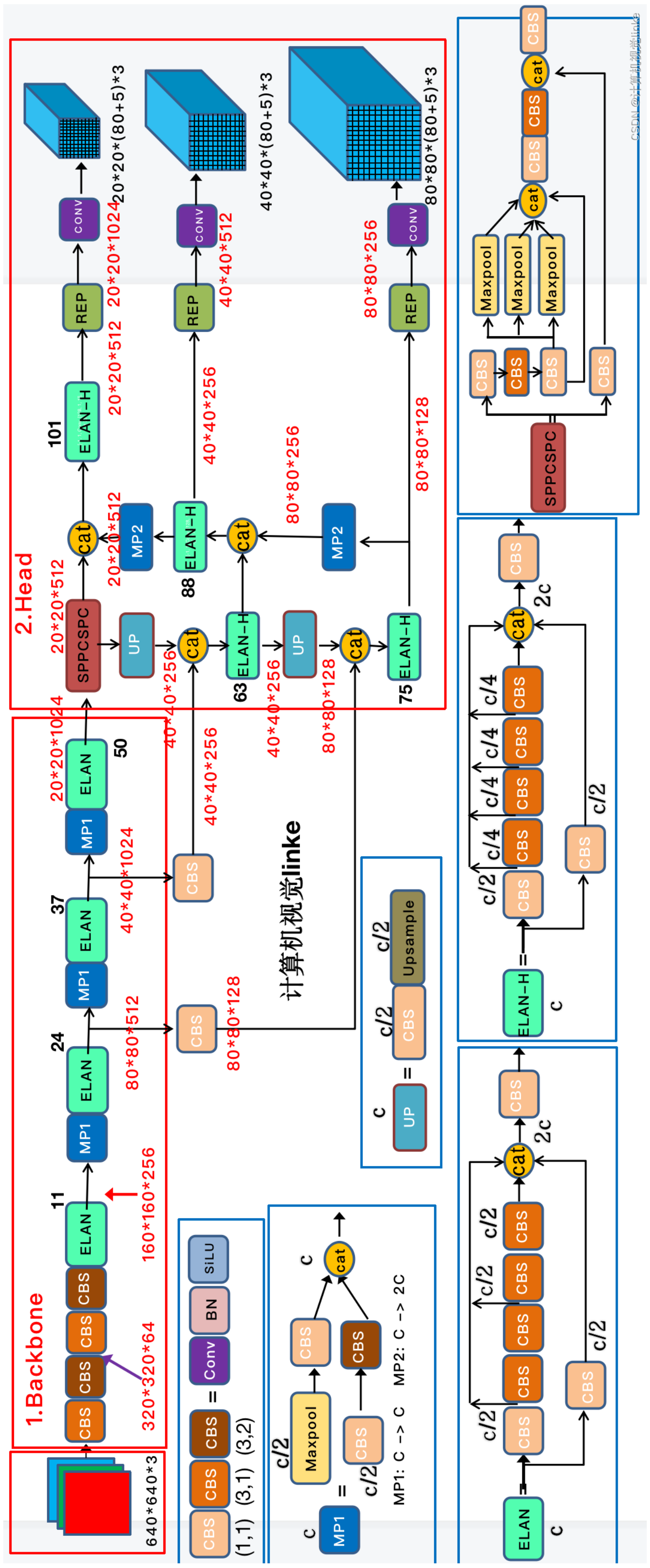
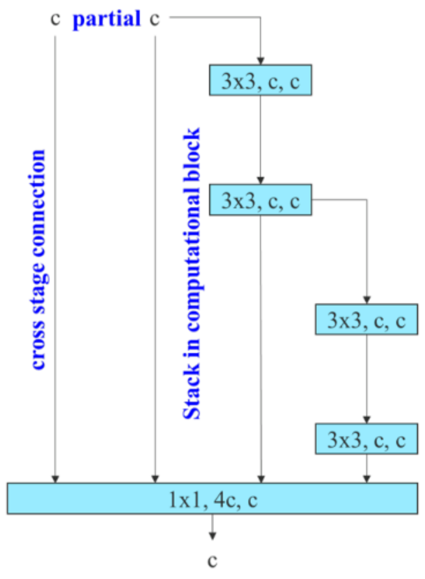


Figure 1 – Architecture of Yolov7 model (linke, 2023)



(c) ELAN [1]

Figure 2 – Architecture of an ELAN block

The second point wise convolutional layer from there is chained to two consecutive $3 \times 3 \times \mathsf{c}'$ convolution layers in a third parallel route. This enables adding a third “stack” of length c’ to the ELAN output tensor. These two $3 \times 3 \times \mathsf{c}'$ conv layers are again chained to an other pair of $3 \times 3 \times \mathsf{c}'$ conv layers which again enables producing even higher level features which are the last “stack” of the output tensor of the ELAN block. The outputs of the four parallel routes which we loosely called “stacks” are concatenated in a tensor of size $\mathsf{H} \times \mathsf{W} \times 4\mathsf{c}'$ to which a 1×1 convolution layer is applied so that the feature dimension of the output tensor is of the desired length instead of 4c’.

We view the ELAN block from the perspective of what GoogleNet aimed to achieve with the use of Networks In Networks ("NiN") blocks called Inception modules. Inception modules used parallel routes to enable the model to choose during learning which parallel route better enabled it to minimize the loss. GoogleNet’s inception modules used 3×3 convolutions chained to 1×1 convolutions to apply transformations **more complex than linear ones** to image regions screened by the previous 3×3 kernels.

In the light of this, we understand the ELAN block as allowing the model to choose during training whether it best minimizes the loss with the simple yet powerful 1×1 convolution route or with a 1×1 convolutional layer chained with more classic sequential 3×3 convolutions, with two intermediary routes in between. The routes are not exclusive of each others but the learning can make the values of intermediary tensors favor one or several routes over one or several others.

Based on our understanding of Yolov7, we address the problem of training the model.

2.3 Strategy for building training dataset

2.3.1 A layered method : transfer learning followed by fine-tuning

The following paragraph is heavily based on my deep learning course at CNAM (Audebert et al., 2023, see[7]).

We define for the scope of our report, transfer learning as the method which transfers the representations learned by some model (in the form of its CNN-based backbone weights) onto a given target object detection model whose objects which we want to identify are close enough semantically to the source dataset or some of its classes. In this method, only the classifier is trained on the target dataset.

We define Fine-tuning as the method where one starts from pretrained weights deemed semantically close to the target dataset, on top of which, the ***whole architecture*** is trained on the target dataset. It is required the target dataset is large enough to avoid overfitting and the ***learning rate is smaller*** because it is hypothesized that the fine-tuning picks up from where the pretrained weights got already.

Our target objects to detect are two specific microbes and we don't have manually labeled data to use for training at the time the internship starts. From this state of things, we could either look for an open source labeled dataset to transfer learn or manually label image data which specifically show C. Reinhardtii and E. Coli.

7. https://cedric.cnam.fr/vertigo/Cours/ml2/docs/coursDeep3.pdf (p27)

Each approach however has its limits : Regarding transfer learning : if we look for a dataset of images whose objects are close enough, semantically, to our microbes, how close would it be ? For example, what if we found one, close to some extent but which didn't include either one or neither one of our microbes ? Regarding fine-tuning : if we attempt to manually label our own dataset for training Yolov7 : can we label enough images ?

In general, if the target dataset is too small, as rough indication : for deep models, if it is less than thousands - up to tens of thousands sample images, fine-tuning would over-fit and as result, transfer learning is preferred.

These approaches are simple transfer learning/ fine-tuning approaches. Our strategy is do a three-layers approach which enables gradual specialization of the model :

1. Transfer by initialization. This is transfer-learning where instead of starting from randomly initialized weights, one initializes learning from weights pretrained on a large dataset (which has from hundreds of thousands up to millions of sample images).
2. Transfer learn from those pretrained weights into, hopefully, a large dataset of objects semantically close to our microbes.
3. Fine-tune, from there, with a second layer of training this time into a small manually labeled dataset.

In the first stage, the transferred weights are actually *not* semantically close to our target objects to detect. Rather, it is comparable to transfer learning only in the method which is used. This first stage takes into account a solved problem which is to favor ***fine-tuning from weights pre-trained*** on large datasets instead of training from randomly initialized weights in order to mitigate over-fitting. Yolov7 having nearly 37 million parameters, over-fitting is a likely problem that the model can face even with medium size datasets of hundreds of thousands of image samples. So we choose to initialize in our case with weights trained on a large dataset.

At the second stage, we make use of the following hypothesis : within the backbone, the early layers, in general, learn basic patterns and gradually as one goes deeper into the architecture, the learned weights are combinations of earlier weights which activate on higher level and complex patterns more and more specific to the target classes that the model detects. In this frame, it has been observed that the early layers of a deep CNN are very similar from one model to an other even if the classes they distinguish are not semantically close to one an other. So the **weights of the early layers** trained on a large dataset can be **not just starting points but *transferred* altogether** onto our model. We deem the 12th layer to be early enough that it has learned (on a large dataset) basic enough shareable patterns. Layers deeper than the 12th are trained on an intermediary dataset which has semantic proximity with our target objects to detect.

The third stage picks up from the resulting weights from the second stage and fine-tuning would be performed on a small manually labelled dataset.

In order to mitigate, over-fitting, we hypothesize that training with a ***very*** *early stopping* strategy may give results.

2.3.2 Datasets

Our strategy is applied but in practice the datasets we find don't satisfy all the properties we looked for. We detail the datasets below.

i. The Ms COCO dataset for transfer by initialization

We looked to initialize our training on weights pretrained on a large dataset but Yolov7 being very deep, we were initially keen to use a dataset of millions of images. In the end, we use the relatively large/ medium Ms COCO dataset. The 2017 version of the Ms COCO pretrained weights[8] has 80 classes and a training set of 118K images. The objects are not semantically close to the objects we aim to detect but the weights still learned to capture useful features especially in the early layers which activate on basic features such as lines or contrasting colors. So that's a better starting point for training than randomly initialized weights and the early layers can usefully be transferred as part of the set of weights.

8. https://cocodataset.org

ii. The EMDS-7 dataset as intermediary dataset

This EMDS-7 is the seventh version of the Environment Microorganisms Dataset (Yang et al., 2021). It has 42 classes if the "unknwown" class is included. It has 2365 images making it a small dataset. What is relevant for our problem are its objects : micro-organisms which are mostly single-cell organisms, the others being multi-cellular organisms still of microscopic size. Those are comparable to C. Reinhardtii and E. Coli which are single-cell organisms. Those environment micro-organisms have common features with our target objects like a ***membrane*** which defines the boundary between their body and the outside. So training on such a dataset would enable the model to learn useful features for our detection problem.

One thing to note is that training one this dataset may introduce *biaises.* We notice most of the micro-organisms are photographed on a light green or light blue background which might become an incentive for the model to detect objects if the background is light green/ blue. We also notice some of the images display the scale of the image which has previously been shown to biais a model to see something in an image if a scale is present.

The background color in this dataset is not like the background color that shows on the lab's microscope. However, the background in the EMDS-7 still has color variation and we need to keep in mind it doesn't take away from the ability of the Yolov7 to learn useful features from the 42 classes of microbes. Also, It will be only used on the transfer learning stage of the training. We hypothesize that, at this stage the possible biases are less problematic because the second layer of training comes on top of it and will be more decisive.

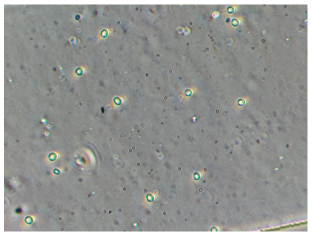
iii. The manually labelled dataset for fine-tuning

We start working on our method without a manually labelled dataset. We find a tool called "Roboflow" which is developed by a start-up and enables manually labelling a dataset. We take a risk because we don't know what problem might occur when manually labelling and yet, we make the assumption that we could gather a dataset of about a hundred or more manually labelled images. In the end, we are able to label only 24 images, which after data augmentation is applied, produces 60 images. We have three classes : C. Reinhardtii, E. Coli, and "unknown". As stated above, fine-tuning on a manually labelled dataset requires, a dataset large enough that justifies fine-tuning instead of transfer learning. Here, we didn't label enough images to fine-tune so our model would likely over-fit and not generalize well.

In our understanding however, a **trade-off** should be done here between over-fitting and loosing useful learning on the manually labeled dataset which is closest to future data. So, we choose to tolerate fine-tuning on a very small dataset.

Besides, the several ways how we tried to prevent over-fitting might counter-balance the problem that our fine-tuning dataset is too small. We might mitigate over-fitting with :

1. starting from weights pre-trained on Ms COCO,
2. the weights transfer at the first twelve layers,
3. very early stopping (at the cost of increasing biais),
4. and possibly, although we are not aware at the time of the internship if several layers transfer learning / fine-tuning can mitigate over-fitting, we suspect it might (we didn't look to see what results other labs might have gotten with layered transfer learning / fine-tuning).


(a) C. Reinhardthii image from the lab's camera (validation set)

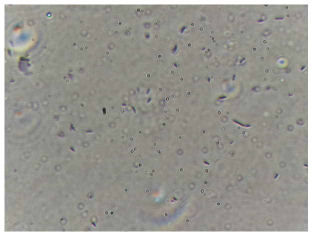

(b) E. Coli image from the lab's camera (train set)

Figure 3 – samples from the manually labelled dataset

3 Results

3.1 First stage : layered transfer learning on EMDS-7

3.1.1 An other level of layering : itself layered transfer learning

i. pre-trained weights download

Yolov7 was trained from scratch on the Ms COCO dataset. The official Yolov7 repository includes indications how to download the weights obtained from this training on the training set of Ms COCO, which we do (link for downloading[9]).

ii. Transfer learning on EMDS-7

We download the EMDS-7 dataset from here[10]. We then upload the EMDS-7 dataset on Roboflow so that it can be shared with the team more easily. We don't apply data augmentation because then, we deem this intermediary learning less critical than the fine-tuning which comes at the second stage.

In the transfer learning stage, we introduce our **own method** which, as far we are aware, hasn't been described in research. Based on the idea that from bottom to top of the architecture, the weights are gradually more specific to the classes of the dataset they were trained on (Ms COCO), we do an other "layering" within the transfer learning stage. This consists in getting the layers which are after the frozen first 12th, go through more training (more epochs) the deeper they are in the backbone (since they learned weights that are more and more specific to Ms COCO and thus more and more detrimental for detecting the EMDS-7 objects).

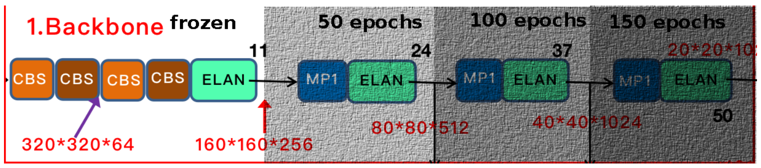


FIGURE 4 – Layered transfer learning method schema with more epochs run the deeper in the backbone which results in gradual specialization to target dataset.

Therefore, we train the non frozen layers in the following manner : first, we freeze the first 38 layers (recall they are numbered from 0). This enables training the last ELAN block

9. https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7_training.pt
10. https://figshare.com/articles/dataset/EMDS-7_DataSet/16869571

of the backbone which is particularly interesting in our view since it includes the point wise convolution route explained in 2.2.4 . It enables training the FPN too because the output of the last ELAN block of the backbone is fed to the FPN from the top.

We save the weights we obtain in this manner so we can pick up from there as we continue training. From the saved weights, in the same way, we do an other round of running 50 epochs after freezing at the 25th layer just before the second last ELAN block. We save the resulting weights here too and then go on the third round of training with freezing set for the first 12 layers. This produces a model which is gradually specialized to the intermediary target dataset as one goes from bottom to top of the architecture. Specifically, layers 12-24 had training of 50 epochs, layers 25-37 had 100 epochs and layers from 38 and beyond (including the FPN) had 150 epochs.

In our thinking, this method might mitigate over-fitting too in an additional way not mentioned in section 2.3.2 iii. We hypothesize here that this layered transfer learning with gradual specialization, enables **training only as little as necessary** (on a small dataset) by keeping the intermediary layers of the backbone not "fully" trained.

Note that, we were not able to reshuffle the training dataset in between layers of training within the layered transfer learning because Roboflow didn't seem to allow it and Yolov7 splits training, validation and test sets physically in folders on a disk making reshuffling (the training set) both heavy and risky.

3.1.2 Hyper-parameters

Our idea was to have a batch that can be representative of the full training set as much as possible while kept as small as possible so that learning is computationally faster. We wanted to use of a rule of thumb shared by Yann Le Cunn in his class of 2020 at New York University where he suggested choosing a batch size of roughly the same size as the number of different classes in the dataset[11]. When we tried with a batch size of 64 or 32, we got an error message so we had to set the **batch size** to the largest size which worked : **16**, in the

11. https://youtu.be/d9vdh3b787Y?t=1292

end. The batch size is set once and corresponds to its specific target dataset.

We use *momentum* and a slowing learning rate of 0.01 which reaches 0.001 at the end of a training round (we only use momentum as optimizer throughout the internship). One set of hyper-parameters which is in the Yolo.yaml file is the **size of the anchor boxes**[12]. As we mentioned in section 2.2.3., the predicted bounding boxes are matched anchor boxes whose offsets have been optimized during learning. The problem is the microbes and E. Coli in particular render the anchor boxes of level P5/32, useless (they are designed to look for large objects). This is an interesting set of hyper-parameters to consider tuning. Methods exist (see HUGHES, 2022b) but time doesn't allow we look into it so we hypothesize matching and box coordinates regression are sufficient.

3.1.3 Intermediary results

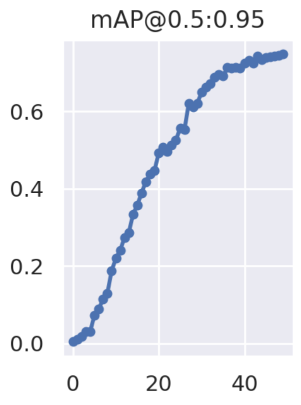
"mAP" for Mean Average Precision is a metric used in object detection to evaluate the performance of a model on a validation set. Without going into the details, we oversimplify it this way : for all the images of the test set, for a given class, the predicted bounding boxes are ranked based on their confidence (the ranking transcends images). Then, Precision vs. Recall for that given class, is plotted in an iterative way where, starting with the best ranked box. One plots Precision/Recall for the entire set of images based on the current boxes that have been taken into account. Precision and Recall here are based on a given IoU threshold, for exemple : 0.5 beyond which, the prediction is defined as a True Positive. After plotting the point at the coordinates (Precision, Recall), we remove the GT box and its predicted box from consideration. At the next iteration, re-calculate Precision and Recall for the entire set of images but including the predicted box that's being taken into account (we're at the box with the next best confidence score now). Plot the new calculated Precision/Recall point. Repeat until the GT boxes are exhausted and measure the area under the curve. The area is AP for one class. Repeat this algorithm for every class and take the average of the different AP. That's mAP for 0.5 threshold. Now in our case, with small objects that can be clustered, a threshold of 0.5 might be a little low. It would be best to use an average of the mAP found

12. https://github.com/WongKinYiu/yolov7/blob/main/cfg/training/yolov7.yaml#L8C2-L8C2

at different IoU thresholds from 0.5 to 0.95 which we choose as our evaluation metric.

The mAP on EMDS-7 is around 0.7 after the first 50 epochs which is good and quick (see figure 4 (a)). The mAP data of the next round of training were lost by the time we wrote this report. At the end of transfer learning, we have an mAP of around **0.76**.

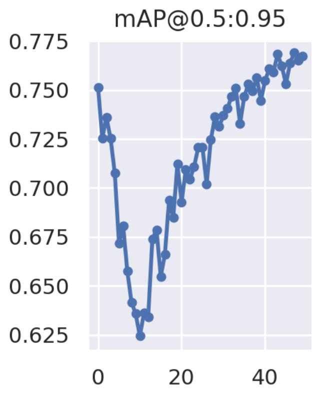
As we plot the mAP calculated over this layered training, we notice a drop early at each new round of 50 epochs. We hypothesize that this is due to the weights vector ($\mathbf{W}$) whose dimension becomes larger at each transfer layering (when we lower the level of the deepest frozen layer). The prediction during training (for any training sample) being a function of $\mathbf{W}$, when the freezing bar is lowered, the $\mathbf{W}$ that was learned becomes to some extent, initially, irrelevant in the new larger dimension of the new manifold. The cost function becomes less convex too with the added layers of the intermediary model. Those two factor possibly explain the observed drop.



(a) mAP after 50 epochs with layers 0-37 frozen

graphic lost

(b) mAP after 50 more epochs with layers 0-24 frozen



(c) mAP after an other 50 more epochs with layers 0-11 frozen

FIGURE 5 – We notice high mAP and a drop right after each new training is ran, each new round having a new larger set of weights to train.

3.2 Second stage : fine-tuning

3.2.1 Dataset details

We use our manually labelled dataset of 24 images augmented to 60 images.

One hypothesis in supervised learning is that the training samples are drawn from the same distribution as the future samples. This is never a reality in practice and future experiments that the team will perform are not understood in details so we give ourselves very simple constraints.

1. Class balance : We try to have roughly an equal number of E. Coli and C. Reinhardtii. Regarding the unknown objects, we hypothesize that they should rarely colonize the microchip so we only label them in four images and they amount to six objects in total.
2. Diversity : We collect images taken from the lab but also other images found on the internet, in order to enable the dataset to contain objects described with richer semantic information. This comes after we notice that the lab images resemble each others because of the sequential procedure how they are taken and are low resolution too. Redundancy in the features in the training set is normal and enables learning but we wish to enable the model to learn useful features about the objects that the lab images don't capture well. The extra images from the internet have better resolution and show microscopic features such as "microbe texture" and are easier to annotate too (especially in case of E. Coli). We think this would enable the model to learn more representations and mitigate over-fitting too. Note the images found on the internet show objects not necessarily of the same scale as what our camera captures but Yolov7's FPN solves this problem.

In the manually labelled dataset, the images never show both E. Coli and C. Reinhardtii on one same image. We may have images that include several classes but that's only if an object of class "unknown" is present. This may introduce *biais* in the model which encourages the model **not to detect E. Coli and C. Reinhardtii in one same image**. Out of the 24 base images, the training set is made up of 18 augmented to 54 images, the test and validation sets are made up of 3 images each. We upload it on Roboflow for ease of use accross the team [13]. Note data augmentation is only applied on the training set which is the most frequent method.

13. https://app.roboflow.com/fine-tuning-data/c.reinhardtii-e.coli/19

3.2.2 Hyper-parameters and other tuning

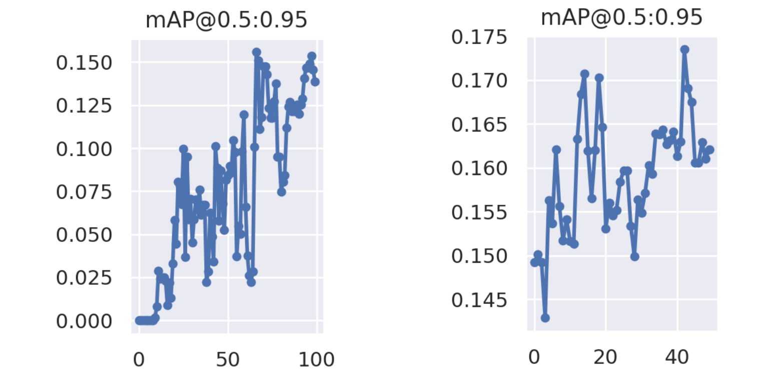
We don't freeze any layer to enable the whole CONV based backbone in particular to learn representations on this fine-tuning dataset.

In a bit of a step by step way, we run two training rounds. The first one with a decreasing learning rate of 0.005 which reaches 0.0005 at the end of training, 100 epochs and a batch size of 4. After this first training round, we plot the mAP over the epochs to see what results we get and if we are converging (recall we seek to apply very early stopping). The mAP seems pretty low especially the average mAP over different IoU thresholds but doesn't seem to have converged yet based on the shape of the curve.

We do from here, a second round of training over 50 epochs with an even smaller learning rate of 0.001 which decreases to 0.0001 at the end of training. This time, plotting the mAP, we can see we are starting to converge.

3.2.3 Final results

As just stated, the graph which plots mAP over epochs shows an increase of mAP but so small after a hundred epochs that we interpret it as showing training is nearing convergence. As such, we decide the second 50 epochs run is the last run of training. We have an mAP around 16% which is low and means the model *wouldn't* perform well.



(a) mAP after 100 epochs with lr=0.005-0.0005

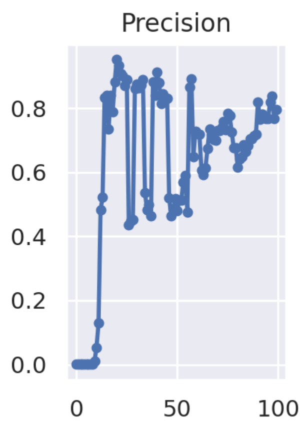
(b) mAP after 50 more epochs with lr=0.001-0.0001

Figure 6 – We notice low mAP which seems to start converging as we reach 150 epochs of fine-tuning.

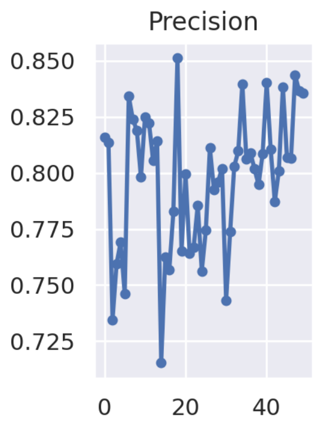
However, recall our validation set was made of only three images each including its set of objects. We can't draw conclusions on such a small validation set. The validation set doesn't enable any results to be statistically significant. Therefore we look deeper into our results to attempt to retrieve other information on our model performance. We can plot Precision and Recall which, first, we define based on Wikipedia (Wikipedia contributors, 2023c). In the following paragraphs we will define non-detected objects as classified as background. Therefore, They may be included in the False Negatives but are never Positive (We have limited understanding of Precision and Recall and, regardless, no conclusion is drawn due to the size of the validation set).

i. Precision

Precision measures the fraction of True Positives ("TP") over the total of positive predictions (TP + False Positives ("FP")). It answers the question : "how many relevant items are retrieved ?". If we plot Precison over the 150 epochs we can see the Precision is **starting to converge** and reaches a value a little over **0.8** which would be good precision. It means that, out of the GT objects, 80% were detected and correctly labelled. So that *would* be good performance. Again, we have a validation set of only 3 images so no conclusion is drawn.



(a) Precision after 100 epochs with lr=0.005-0.0005

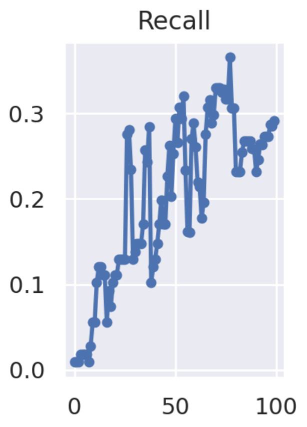


(b) Precision after 50 more epochs with lr=0.001-0.0001

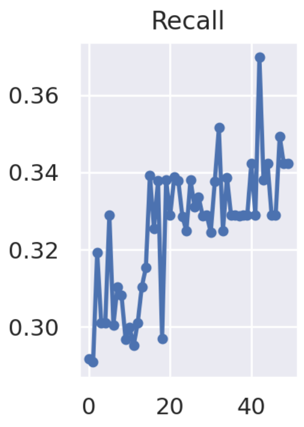
Figure 7 – We reach Precision around 85% which *would* be good performance.

ii. Recall

Recall is defined as the fraction of relevant instances that were retrieved. This metric enables measuring how much we miss both detecting and correctly classifying, GT objects that are present in the validation set images. If we plot Recall over the number of epochs ran, we see Recall reaches around **0.34** which *wouldn't* be good performance. It means we would miss around 66% of positive objects of a given class (from miss-classification or not detecting an object at all).



(a) Recall after 100 epochs with lr=0.005-0.0005



(b) Recall after 50 more epochs with lr=0.001-0.0001

Figure 8 – We reach Recall around 34% which *would* be bad performance.

So here, without drawing any risky conclusions, since Precision seemed good, we suspect the weakness of the model may lie in the detection of objects. That would stem from bad objectness prediction. Keep in mind here that E. Coli on our lab's images were small, blurry and often located close next to each others. The validation set included two images taken from the lab. So it could be that low performance on the E. Coli class is dragging the results down.

3.3 The over-fitting problem

Having finished training and retrieved final mAP, Precision and Recall we evaluate if we did over-fit or not. One approach can be to plot the loss on the validation set to see if, after decreasing initially, it starts to increase again which is a sign of over-fitting. When the model over-fits the training data, it looses ability to generalize and perform well on unseen data (the decrease doesn't continue and converge like in training). We are again brought to deal with the fact our validation set has only three images. So we cautiously analyze the plots printed at the end of training whith the loss over the number of epochs on the validation set.

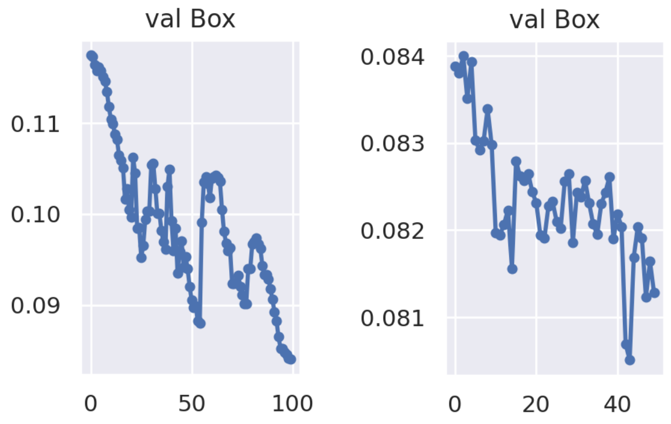


Figure 9 – We can see a noisy decrease of the box loss which slows down through fine-tuning.

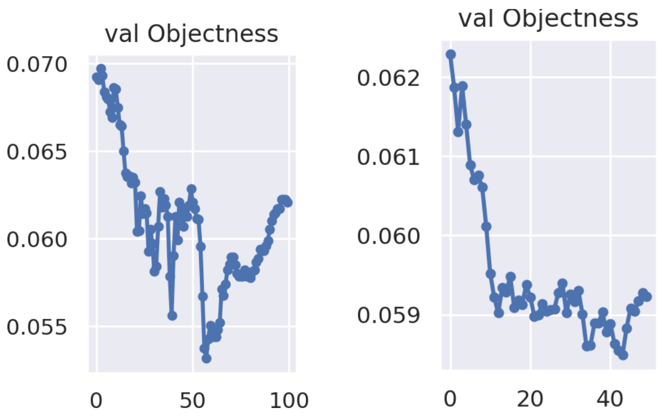


FIGURE 10 – We can see a noisy decrease of the objectness loss which seems to converge before 150 epochs.

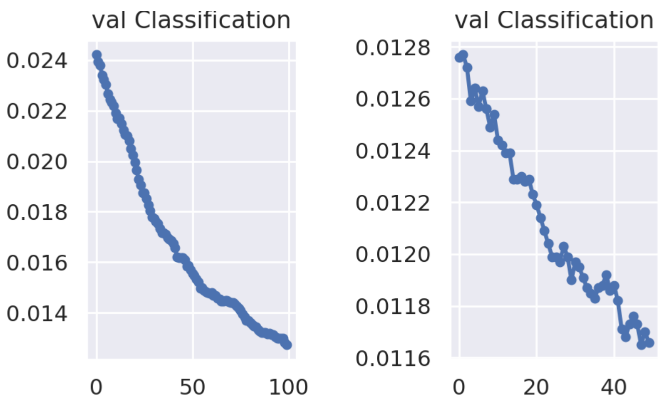


FIGURE 11 – We can see a smooth decrease of the classification loss which slows down towards the end of training.

When we plot the loss over fine-tuning on the validation set (each component of the loss plotted separately), we don't seem to observe any increase of the loss on the validation set. Yet, Objectness loss, however small, is very noisy and doesn't decrease much through fine-tuning. Altogether though, we see no sign of over-fitting.

In conclusion, results on the validation set show no sign of over-fitting but our validation set is so small we refrain from drawing conclusions. Yet, we hypothesize Recall is low because of the difficulty to even detect E. Coli. If we can't validate our model and hyper-parameters, we deem it unnecessary to calculate results on the test set which is as small as the validation set too.

As illustration only, we make a few non random tests which are consistent with our hypothesis (see one in figure 12 below).

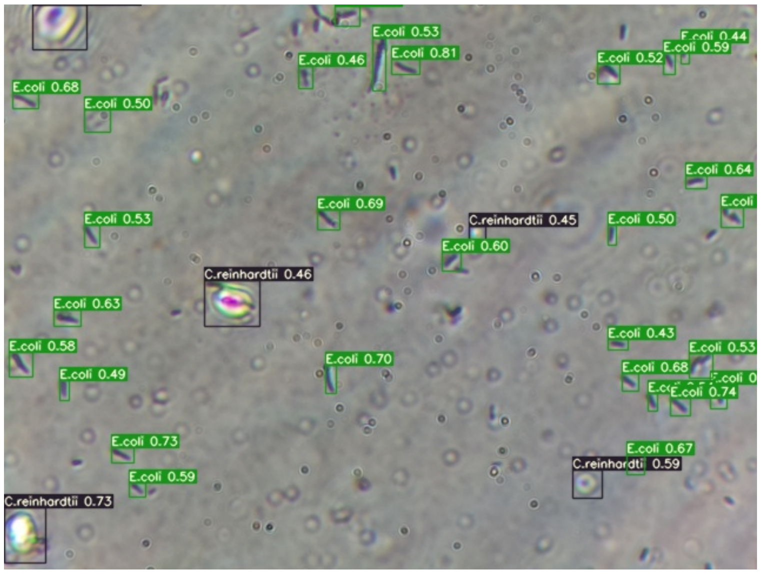


FIGURE 12 – A non random test on an unseen image. Note the model is robust enough that it "beats" the biais that E. Coli and C. reinhardtii are never both present in an image in training.

4 Discussion : achievements and possible improvements

Having not been able to evaluate our model, we discuss what our method achieved nonetheless.

4.1 No overfitting

One main concern throughout the internship was the risk of over-fitting to the training data since we had to rely on a manually labelled dataset in the last stage of training the model. Our results show no sign of over-fitting. It is unclear which of our tricks best helped mitigate over-fitting but as a whole, our method was successful in this regard. We think the layered transfer-learning described in 3.1.1 played a role.

4.2 Ability to learn from the target dataset

Our non random tests showed the model managed to learn representations to successfully detect some objects.

4.3 Avenues for improvements

Even though our final mAP is low, it may be, to a great extent, a result of the validation set being too small. We think that if our fine-tuning method had benefited a few improvements, our results might have improved a lot.

1. The fine-tuning set could have been increased in size so that mAP could have increased and been better interpreted. Tools exist such as automatic pre-labelling software.
2. Pre-processing the sample images captured by our camera in order to make the the object detection task easier for our model. One could apply some sharpening filters for examples.
3. As mentioned in 3.1.2 performance could have benefited from tuning the anchor box sizes and aspect ratios so that they better matched our target objects.
4. Regarding the fine-tuning method itself, compared with what other teams have done, we performed a simplified fine-tuning. This wouldn't be our major problem to address but fine-tuning is often performed on the target dataset with a small learning rate at every layers of the backbone except at the last layer which is trained with a normal learning rate.

This is the end of the report. The template used to write this report was created by Brendan Borne[14]. The report is published on Github[15].

14. https://www.overleaf.com/latex/templates/french-report-template/bbysksgswgtk
15. https://github.com/ZukoTom/internship_works

Références

Audebert, N., Ferecatu, M., & Thome, N. (2023). Réseaux convolutifs modernes. https://cedric.cnam.fr/vertigo/Cours/ml2/coursDeep4.html

Ge, Z., Liu, S., Wang, F., Li, Z., & Sun, J. (2021). YOLOX : Exceeding YOLO Series in 2021.

Hughes, C. (2022a). YOLOv7 : A Deep Dive into the Current State-of-the-Art for Object Detection. https://towardsdatascience.com/yolov7-a-deep-dive-into-the-current-state-of-the-art-for-object-detection-ce3ffedeeaeb#7c9e

Hughes, C. (2022b). YOLOv7 : A Deep Dive into the Current State-of-the-Art for Object Detection. https://towardsdatascience.com/yolov7-a-deep-dive-into-the-current-state-of-the-art-for-object-detection-ce3ffedeeaeb#1f98

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). YOLOv7 : Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv :2207.02696*.

Wikipedia contributors. (2023a). Computer vision — Wikipedia, The Free Encyclopedia [[Online ; accessed 15-October-2023]]. https://en.wikipedia.org/w/index.php?title=Computer_vision&oldid=1179523951

Wikipedia contributors. (2023b). Convolution — Wikipedia, The Free Encyclopedia [[Online ; accessed 15-October-2023]]. https://en.wikipedia.org/w/index.php?title=Convolution&oldid=1173797123

Wikipedia contributors. (2023c). Precision and recall — Wikipedia, The Free Encyclopedia [[Online ; accessed 6-November-2023]]. https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=1182034360

Yang, H., Li, C., Zhao, X., Cai, B., Zhang, J., Ma, P., Zhao, P., Chen, A., Jiang, T., Sun, H., Teng, Y., Qi, S., Jiang, T., & Grzegorzek, M. (2021). EMDS-7 : Environmental Microorganism Image Dataset Seventh Version for Multiple Object Detection Evaluation.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023a). Dive into Deep Learning. https://d2l.ai/chapter_convolutional-neural-networks/why-conv.html#convolutions

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023b). Dive into Deep Learning. https://d2l.ai/chapter_computer-vision/anchor.html

linke. (2023). A detailed explanation of the basic network structure of Yolov7. https://zhuanlan.zhihu.com/p/543743278

Microbes detection in microfluidics-based microscopy imaging

Abstract

We use Yolov7 to model an object detector to detect microbes under a microfluidics chip-based microscope. Our target manually-labelled dataset being very small, we first perform a layered transfer learning on top of which we perform fine-tuning. Our model doesn't perform as well as hoped but we think our method could be useful in object detection tasks whose target dataset is so small that over-fitting must be addressed.

Keywords— object detection, signal processing, deep learning, convolution, microscopy