Detection and Tracking of Soil Protists using Deep Learning

Jingmo Bai & Zuoyi Yu

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis TFRT-9999
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2024

Abstract

Plastic residues can fragment into nanoplastics and bring various pollutants to the soil, which results in a massive environmental risk that is endangering entire ecosystems. Soil protists, as a vital part of microbial food webs and carbon cycles, are also considered to be strongly affected by the presence of nanoplastics. Until now, many studies have found that plastic residues can have either positive or negative effects on different elements of ecosystems. However, no research has been conducted to quantify the impact of plastic on soil protists due to a lack of tools for visualizing and studying these microorganisms. Therefore, we try to use a deep learning-based object detection model, You Only Look Once (YOLO), to track and record the speed and trace of the protists in the soil chips.

In this work, YOLOv8 model is used to detect and classify 9 classes of protists in the videos acquired from the soil chips. To achieve better performance, several model improvement methods are tested. Generative Adversarial Networks (GANs) are also applied to generate synthetic images to solve the lack of data. Then we record the speed and trace and compare them among different treatment conditions to analyze the effects of nanoplastics on the protists. In conclusion, we demonstrate the feasibility of leveraging the power of AI and deep learning to help scientific research. We also conclude that high-concentration nanoplastics will cause the protists to move slower than usual, different protists have disparate moving patterns.

Acknowledgements

We would like to thank our supervisors, Bo Bernhardsson, Edith Hammer and Hangbang Zou for their constant support and constructive guidance throughout the thesis project. We thank Paola Micaela Mafla Endara for sharing the data collected in her PhD dissertation.

The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

We would like to express our deepest gratitude to our parents, for all the unconditional love and support not only during this thesis, but throughout the entire education journey. None of this would be possible without you.

Contents

1

Introduction

1.1 Background

Plastic Pollution in Soil

Plastic pollution has become a significant global environmental issue due to the extensive use of plastic materials in daily life combined with insufficient waste management around the world. The accumulation of plastic waste in the environment, including water bodies and soil poses significant threats to ecosystems, human health and the well-being of our planet.

Plastics are synthetic polymers mostly made from fossil fuel-based chemicals like natural gas or petroleum. They are versatile, durable, lightweight, and inexpensive to produce, leading to their widespread applications in various industries. However, these same properties also make them resistant to many natural processes of degradation, resulting in their persistence in the environment.

An estimated 9.2 billion tonnes of plastic were produced between 1950 and 2017, more than half of which has been produced since 2004. If global trends on plastic demand continue, it is estimated that by 2050 annual global plastic production will exceed 1.1 billion tonnes[Wikipedia contributors, 2024b].

The management of plastic waste is country-specific and normally insufficient around the world. Much of this waste ends up in landfills or improperly disposed of in the environment, where it breaks down into smaller fragments and may take centuries to decompose.

Soil Protists

Soil is the habitat of countless microorganisms. As plastic residues reside in the soil, a thorough understanding of their interactions with soil microorganisms may have a substantial ecological impact and has emerged as a critical study area, since these soil microorganisms play a vital role in global carbon cycling.

Protists include all eukaryotes except animals, land plants (Embryophyta) and arguably Fungi [Geisen et al., 2018a], and have been discovered in recent decades through environmental DNA research. They are consumers of bacteria, fungi, and

other tiny eukaryotes, and play important roles in microbial food webs and carbon cycles. They can even survive and disperse under extreme environments such as low or high pH, temperature, or salt stress [Wu et al., 2022]. Protists control populations and sculpt communities as parasites of larger protists as well as plants and animals. Plant development is enhanced by the release of nutrients from predatory soil protists. What's more, The study of soil protists is crucial to our comprehension of microbial biogeography and eukaryotic evolution[Geisen et al., 2018b].

However, despite their importance in soil ecosystems, protists remain the least explored microorganisms in soil ecosystems due to their massive diversity and technique challenges. Unlike bacteria or fungi, protists are difficult to culture for study in the laboratory, and many cannot be cultured at all. Their tiny size ($< 30\mu m$ long) and some intricate structures also require specialized techniques for study.

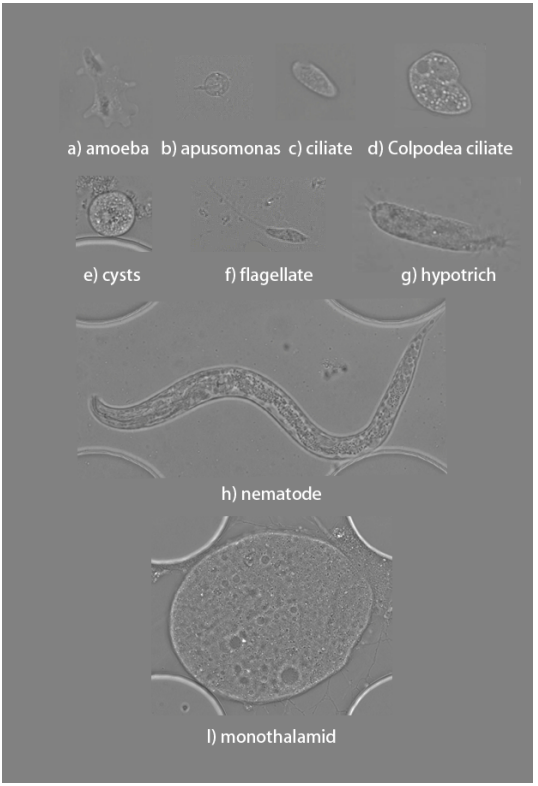


Figure 1.1 The examples of all classes of soil protists we need to classify in this work.

Microfluidic Chips

Traditional laboratory-based approaches to analyse the soil microbes community include growth-based approach, respiration-based approach, biomass-based approach, enzyme-based approach and stoichiometry-based approach. However, none of them provides the possibility to monitor the soil microbes and study their behaviour directly and visually in real-time. They are also incapable of simulating the natural soil environment at the micro-scale in the laboratory. The emerging application of Lab-on-a-Chip or microfluidic technologies provides precise spatiotemporal control over the microenvironments of soil organisms in combination with high-resolution imaging[Stanley et al., 2016].

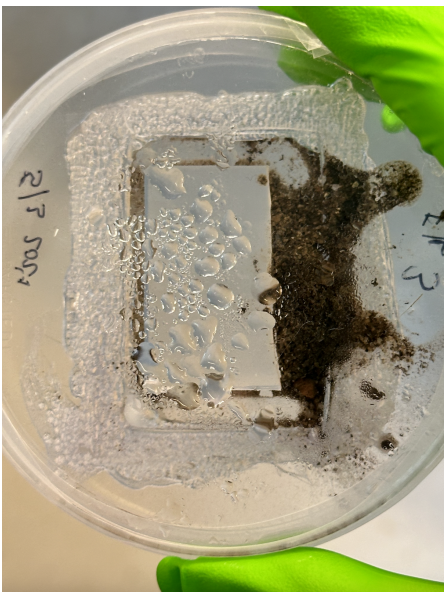


Figure 1.2 A microfluidic chip with soil samples.

Soil microfluidic chips are two-dimensional transparent microchips used to analyze soil samples at a microscale level in the laboratory, as shown in Figure 1.2. These chips allow researchers to study soil properties, microorganisms, nutrients, contaminants and their interactions in a controlled laboratory environment, providing unprecedented advantages over traditional approaches and novel biological and ecological insights.

With the help of microfluidic chips, we can build artificial microhabitats where microorganisms live, allowing us to monitor and track them in real-time with the help of a Nikon Ti2-E inverted light microscope and a Nikon Qi2 camera, as shown in Figure 1.3. This method provides a way to investigate their interactions with nanoplastics. We can collect hundreds or even thousands of images or videos of the

protists' community from the soil microfluidic chips with minimal effort, offering the possibility of systematic analysis.

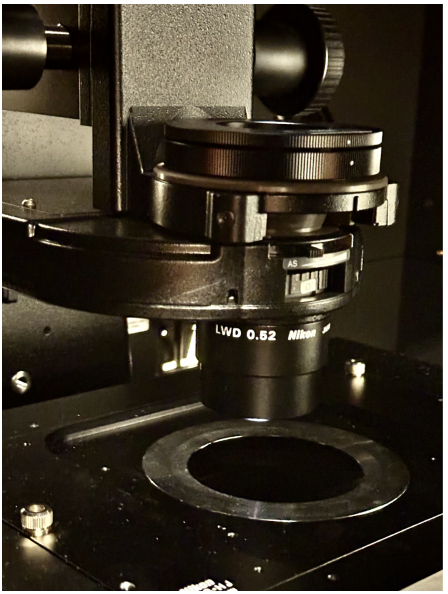


Figure 1.3 The Nikon Ti2-E inverted light microscope setup in the lab.

However, numerous data also become a heavy burden to researchers. Not to mention it is almost impossible for researchers to manually localize all the microbes and record their behavior like moving speed precisely. Therefore we need automatic tools to monitor and analyse microorganism community behaviour on the soil chips. That is why we need to build deep learning models in this thesis. With the development of deep learning techniques, we can train object detection models to localize, classify and track the interested protists, record their location and move trace at the same time. Those data can be very helpful for the analysis of the protists community.

Object Detection

Object detection is a computer vision task that involves identifying and locating visual objects within images or videos. Unlike image classification which only classifies the entire scene, object detection also determines their precise locations. These locations are typically presented by drawing rectangular boxes with labels, called bounding boxes, around the detected objects.

There exists a variety of object detection methods, which can be categorized into traditional methods and deep learning-based methods.

Traditional methods. Traditional object detection methods are built on region selection, feature extraction, and classification [Zhao et al., 2019].

A typical region selection method is sliding windows, which involves moving a window across the image and applying a classifier to each sub-region. This method is computationally expensive and inefficient.

A common handcrafted feature descriptor is histograms of oriented gradients (HOG) [Dalal and Triggs, 2005]. To detect objects of different sizes, the HOG detector rescales the input image multiple times while keeping the size of a detection window unchanged. Although it can be considered a great improvement as a feature descriptor and has been an important foundation of many object detectors and computer vision applications for many years, it is still difficult to manually design a robust feature descriptor that applies to all kinds of tasks and objects.

Deep learning-based methods. In the recent decade, the rapid development of deep learning techniques, especially Convolutional Neural Networks (CNN) [Krizhevsky et al., 2012], has shown remarkable possibilities across various domains. Their ability to learn complicated features from vast amounts of data has led to breakthroughs in the field of computer vision.

Deep learning-based methods can be further categorized into two-stage methods and one-stage methods. Two-stage methods generate region proposals first and then classify each proposal into different object categories. Some typical two-stage methods include R-CNN [Girshick et al., 2014], Fast R-CNN [Girshick, 2015], Faster R-CNN [Ren et al., 2015], and Mask R-CNN [He et al., 2017]. One-stage methods regard object detection as a regression or classification problem, mapping from image pixels to categories and locations directly in a single step. Some typical one-stage methods include You Only Look Once (YOLO) [Redmon et al., 2016], Single-Shot Multibox Detector (SSD) [Liu et al., 2016], and DETR [Carion et al., 2020a].

Generative Adversarial Networks (GAN)

The amount of data on some classes of protists is very insufficient. For some classes, we barely have dozens of samples. This will still be insufficient even after implementing traditional data augmentation methods. Therefore, we decided to use Generative Adversarial Networks (GANs)[Goodfellow et al., 2014] to generate more synthetic data.

GANs are a class of generative models that typically contain two neural networks: the generator and the discriminator. These networks are trained simultaneously and compete against each other by playing a mini-max two-player game based on game theory.

The generator network is trained to capture the real data distribution and generate fake samples by passing random Gaussian noise through a neural network. The discriminator network is trained to estimate the probability that a sample is real or fake to distinguish real samples and fake samples. They are trained simultaneously

using a value function: one tries to maximize it while the other tries to minimize it. During training, the generator distribution will get closer and closer to the real data distribution, becoming capable of generating more realistic samples. In theory, with enough network capacity and training, both networks can reach a local Nash equilibrium where neither can improve because the distributions are equal. At this point, the generator can produce very realistic samples useful for data augmentation.

1.2 Aim and Scope

We aim to leverage the power of AI and Deep Learning techniques to help investigate the interactions between plastic residues and protists. We build a model to detect and track different species of protists, then record the data and analyze their behaviour and properties under various treatment conditions. Hopefully, we can uncover interactions between nanoplastics and protists.

This thesis project is in a proof of concept phase, which means that we build and test experimental models to demonstrate the feasibility of utilizing the power of AI and deep learning to help scientific research. Data are available for only a few species of protists and the amount is quite limited for a deep learning approach. Therefore, we will not be able to build a perfect model in this thesis project, especially given the limited data. In the future, the goal is to deploy a well-performing and generalized model to the microscopes in the department laboratory to enhance the efficiency of scientific research.

1.3 Outline of the Thesis

The rest of this thesis is organized as follows. In Section 2, we introduce the relevant theory concerning generative adversarial networks and object detection methods. Section 3 introduces our dataset, data selection, and data augmentation methods. In Section 4, we detail our workflow, including data pre-processing, synthetic data generation and object detection model training. In Section 5, we present our results and discussion on model comparison and analysis of interactions between nanoplastics and protists. In Section 6, we conclude this thesis and outline future research directions.

2

Theory

2.1 Data Pre-processing

Data pre-processing includes every process concerning manipulating data before using them to train the model. It should include clipping images from videos, grayscaling, resizing the images to the same size, and data augmentation. The main visual features that distinguish different protists from static microscopic images are their shape and contrast instead of colour. In this case, colour is redundant information, so we perform grayscaling on all the data to decrease computation complexity.

Due to the lack of data, we perform many traditional data augmentation methods, including rotation, flipping, cropping, translation, blurring and adding noise.

However, even after performing traditional data augmentation methods, the amount of data on some classes of protists is still very insufficient. For some of the classes like flagellate and ciliate, we have only less than one hundred samples. We therefore decided to use generative models to generate more synthetic data to help train the model.

2.2 Generative Adversarial Networks

GANs are a class of generative models which typically contain two neural networks, the generator and the discriminator. They are trained simultaneously and compete against each other by playing a mini-max two-player game based on game theory.

Vanilla GAN

The vanilla GAN was first introduced by Goodfellow et al. in 2014 [Goodfellow et al., 2014]. It is an adversarial process that simultaneously trains two models based on game theory. This framework corresponds to a minimax two-player game. A generator G captures the real data distribution and a discriminator D estimates the probability that a sample is real or fake. The generator generates fake samples by passing Gaussian random noise through a multilayer perceptron. The discriminator is also a

multilayer perceptron. They can be trained simultaneously using back-propagation and dropout algorithm [Hinton et al., 2012] with a value function $V(G,D)$:

$$\min_G \max_D \mathbb{E}_{x\sim\mathbb{P}_r}[\log D(x)] + \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[\log(1-D(\tilde{x})] \tag{2.1}$$

where $\mathbb{P}_r$ is the real data distribution and $\mathbb{P}_g$ is the generator distribution, $\tilde{x} = G(z)$. Minimizing this value function amounts to minimizing the Jensen-Shannon divergence between $\mathbb{P}_r$ and $\mathbb{P}_g$ [Goodfellow et al., 2014].

We train D to maximize the probability of correctly classifying both the real samples and fake samples from G. We simultaneously train G to minimize the probability of the generated samples being correctly classified by D.

Theoretically, if G and D have enough capacity, they will reach a point at which both cannot improve because the distributions are equal, which means finding a local Nash equilibrium.

However, in practice, it is difficult to get both models to converge, because this value function may not provide sufficient gradient for G to learn. It is usual that early in training, when G is poor, D can easily reject fake samples with high confidence because they are just random noises at the beginning. In this case, $\log(1-D(G(z)))$ saturates. The discriminator loss converges quickly to zero, D's gradient vanishes, and G stops updating, it's called the vanishing gradient. The GAN-based models are known for their difficulties in training [Goodfellow et al., 2020]. In practice, training usually ends up with non-convergence, and the generated images suffer from being noisy and incomprehensible.

Deep Convolutional GAN (DCGAN)

Deep Convolutional GAN (DCGAN) [Radford et al., 2016] is a class of architectures that scale GANs using CNN architectures which is commonly used in supervised learning. The original paper claimed that this class of architectures resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models, by adopting and modifying some changes to CNN architectures.

The first is the all convolutional net [Springenberg et al., 2014] which replaces any pooling layers with convolutions (discriminator) and transposed convolutions (generator). The second is to use Batch Normalization [Ioffe and Szegedy, 2015] in both generator and discriminator which stabilizes learning by normalizing the input to each unit to have zero mean and unit variance. The third is to remove fully connected hidden layers for deeper architectures.

Compared to vanilla GAN, DCGAN uses convolutional networks instead of multilayer perceptron which allows it to be capable of generating high-resolution images. The network structure is mostly from the all-convolutional net. This architecture contains no pooling layers. When the generator needs to increase the spatial dimension, it uses transposed convolution with a stride greater than 1. DCGAN also

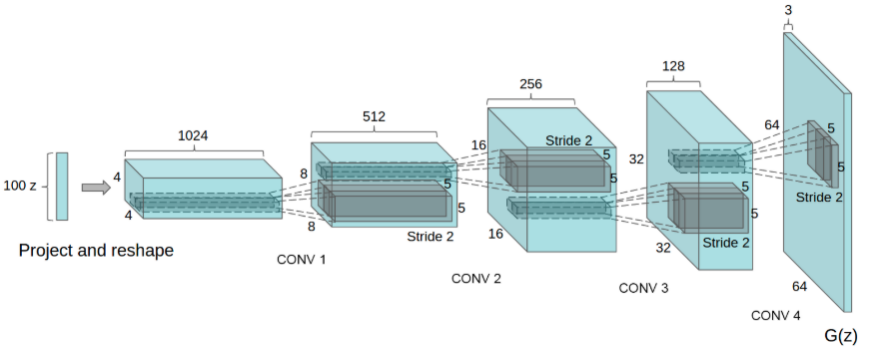


Figure 2.1 An illustration of the DCGAN generator architecture. A series of four transposed convolutions convert high-level representation into a 64 × 64 image. No fully connected layers are used. Figure from [Radford et al., 2016].

use the Adam optimizer [Kingma and Ba, 2014] rather than Stochastic Gradient Descent (SGD) with momentum, which makes training more stable. However, it uses the same value function so it does not address the training difficulties. In practice, it is still common to get non-convergence during training.

Wasserstein GAN (WGAN)

Wasserstein GAN was introduced as an alternative to traditional GAN training, it can improve the stability when training GANs, and get rid of problems like mode collapse[Arjovsky et al., 2017]. The original paper showed that the divergences which GANs typically minimize are potentially not continuous with respect to the generator's parameters, leading to training difficulty. It proposed an alternative value function derived from an approximation of the Wasserstein-1 distance $W(q,p)$, which is informally defined as the minimum cost of transporting mass in order to transform the distribution q into the distribution p (where the cost is mass times transport distance). Under mild assumptions, $W(q,p)$ is continuous everywhere and differentiable almost everywhere. The WGAN value function is constructed using the Kantorovich-Rubinstein duality to obtain

$$\min_G \max_D \; \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] \tag{2.2}$$

In that case, under an optimal discriminator, minimizing the value function with respect to the generator parameters minimizes $W(\mathbb{P}_r,\mathbb{P}_g)$ [Gulrajani et al., 2017]. Compared to the original GAN value function, this value function is more likely to provide stable gradients that are useful for updating the generator.

To enforce a Lipschitz constraint on the discriminator, WGAN proposed to clip the weights of the discriminator. However, as the original paper stated, weight clipping is clearly a terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, it will take a long time for any weights to reach optimality. This can

reduce the capacity of the discriminator model, making it learn simpler functions. If the clipping parameter is small, this can easily lead to vanishing gradients when the number of layers is big [Arjovsky et al., 2017]. However, they stuck with weight clipping due to its simplicity, leaving this topic of enforcing Lipschitz constraints in neural networks for further investigation and potential improvement.

In practice, WGAN still generates poor samples or fails to converge. Without careful tuning of the clipping threshold c, either vanishing or exploding gradients could happen because of interactions between the weight constraint and the cost function.

WGAN Gradient Penalty (WGAN-GP)

The WGAN made progress toward stable GAN training, but can sometimes still generate poor samples or fail to converge. These problems are often due to weight clipping to enforce a Lipschitz constraint on the discriminator. The WGAN gradient penalty proposed an improved method for training the discriminator, by directly constraining the gradient norm of the discriminator's output with respect to its input. They enforce a soft version of the constraint with a penalty on the gradient norm for random samples $\hat{x}$ [Gulrajani et al., 2017]. The modified value function adds a term to penalize the gradient:

$$\min_G \max_D \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] + \lambda \mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \tag{2.3}$$

The sampling distribution $\mathbb{P}_{\hat{x}}$ is defined as sampling uniformly along straight lines between pairs of points sampled from the data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$.

The WGAN-GP is designed to be easier to train, using a different formulation of the training objective which does not suffer from the training difficulties. The WGAN-GP may also be trained successfully even without batch normalisation; it is also less sensitive to the choice of non-linearities used between convolutional layers.

According to the experiments in [Gulrajani et al., 2017], WGAN-GP shows strong modelling performance and stability across a variety of architectures even without careful hyperparameter tuning. It significantly outperforms weight clipping with improved performance, training speed and sample quality.

2.3 Object Detection

Object detection is a computer vision approach for locating and recognizing items in images or videos. Different from image classification, which categorizes complete images, object detection methods identify objects using exact bounding boxes around them to indicate their positions within the image. Deep learning models,

such as convolutional neural networks (CNNs), are commonly used in object detection algorithms and are commonly divided into two categories: one-step and two-step models. The biggest difference between these two models is that the one-step model converts the input image into a grid and predicts bounding boxes and class probabilities directly, while the two-step model needs an extra region proposal step to find out all the potential objects, and then is followed by a region classifier. The most well-known of these two are Region-Based Convolutional Neural Network (R-CNN) [Goodfellow et al., 2014] and You Only Look Once(YOLO) separately [Jiang et al., 2022].

Region Based Convolutional Neural Network (R-CNN)

R-CNN is a deep learning architecture used for object detection, introducing a two-step approach. It consists of three modules: the first module applies a selective search algorithm to find relative region proposals, the second uses a CNN to extract features from each region, and the third module predicts labels using linear Support Vector Machines (SVMs).

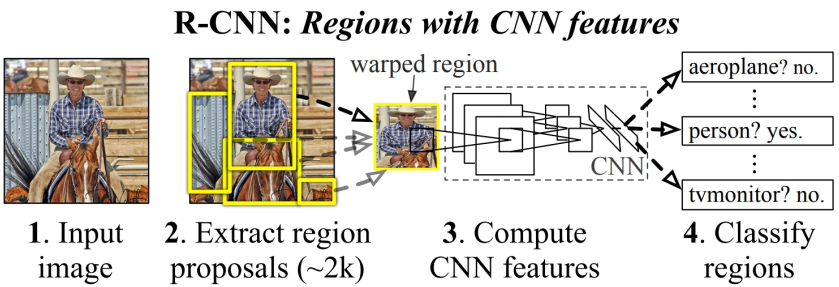


Figure 2.2 Overview of R-CNN architecture. It takes an image as input, then extracts about 2000 proposal bounding boxes sent to a CNN to extract features. Lastly, it uses class-specific linear SVMs to predict labels. Figure from [Girshick et al., 2014].

Region Proposal. R-CNN begins with dividing the input image into multiple regions, which are thought to be all potential regions that include all the objects in the image. These regions are called "region proposals". A famous and common method to locate objects, Exhaustive Search, is often used to find out the locations of these region proposals. This approach uses sliding windows through the whole image to locate the objects. However, this method requires thousands of sliding windows for objects, even for small image sizes, making it really computationally expensive. To make it more efficient, a method called Selective Search combines Exhaustive Search and segmentation is applied. This is the algorithm used in R-CNN [Uijlings et al., 2013].

Selective Search is based on a hierarchical grouping algorithm developed in the paper "Efficient Graph-Based Image Segmentation" by Felzenszwalb and Huttenlocher [Felzenszwalb and Huttenlocher, 2004]. This method can generate a set of small initial regions without spanning multiple objects fast. After getting these starting regions, a greedy algorithm is applied to combine small regions into larger regions: First, all neighboring regions' similarities are computed. Then the two most similar regions are chosen and combined into a larger, single region. This procedure of combining is repeated until the entire image is composed of a single region. Finally, the algorithm uses these segmented region proposals to generate potential object bounding boxes. The illustration of the Selective Search algorithm is shown in Figure 2.3. We can see as the iteration grows, that the sizes of each region proposal become larger and that the method begins to find the object locations.

Figure 2.3 An illustration of Selective Search algorithm at different iteration. There are four types of similarities considered in the algorithm: color, texture, size, and fill. The bounding boxes (below) are drawn based on the segmented region proposals (above). Figure from [Felzenszwalb and Huttenlocher, 2004]

Feature Extraction. After using Selective Search generating around 2000 region proposals, these regions are wrapped to a consistent input size that the CNN expects (227×227 pixel size in R-CNN) and the CNN extracts features for further classification. The CNN is AlexNet which is generally fine-tuned on a large dataset such as ImageNet. The output of this CNN is a high-dimensional vector which represents the features of the input image.

R-CNN chooses the most straightforward of the numerous possible transformations for the arbitrary-shaped areas. It warps all of the pixels in a tight bounding box surrounding the candidate region to the necessary size, regardless of its dimensions or aspect ratio. Before warping, the size of the tight bounding box is increased so that there are exactly 16 pixels in the warped frame.

Bounding Box Regression and Non-Maximum Suppression (NMS). Besides classifying objects, R-CNN also applies bounding box regression and NMS: Bounding Box Regression aims to improve the accuracy of object location through a separate regression model, which is trained for each class in order to fine-tune the bounding box's size of the detected object [Felzenszwalb et al., 2009]. After classifying and regressing bounding boxes, NMS is applied to eliminate highly overlapping or duplicate proposal bounding boxes on the image, which implies a high probability of the same object. NMS helps to ensure that there is only one bounding box for each target object.

You Only Look Once v8(YOLOv8)

In our work, as we want to eventually deploy our model to microscopes and run in real-time, we have high requirements for model performance, especially for running time. However, from above we can see traditional CNN models are too slow to run in real-time, even though there are some improved versions released such as faster R-CNN, as they always have very complicated pipelines. Therefore, to solve this problem, YOLO was developed. Instead of extracting features from proposal regions and then using a separate classifier, YOLO divides input images into grids and predicts bounding boxes and classifications from grids directly, making YOLO really faster in inference time. YOLOv8 is used in our work, which is a series of improvements and extensions made by Ultralytics to the YOLOv5 architecture [Jocher et al., 2023].

The main idea of YOLO is that it converts the object detection task into the regression task by finding and predicting the objects directly, which are called anchor-free detectors. Instead of using sliding windows of different sizes across the whole image like CNN, YOLO divides the input image into $S \times S$ grids. Each grid predicts the bounding box location and the class of object individually.

The architecture of YOLOv8 is illustrated in Figure 2.4. There are mainly 5 modules in YOLOv8 structure:

1. **Conv module** with 2d batch normalization and SiLU activation function.
2. **C2F (cross-stage partial bottleneck with two convolutions) module** reduces the channel size of the feature map, which helps to detect objects better with various sizes.
3. **SPPF(spatial pyramid pooling fast) module** in backbone accelerates computation by pooling features into a fixed-size map and generates a fixed feature representation of objects of various sizes in an image.

4. **Upsample module** in the neck, upsamples the spatial dimensions of the previous layer.

5. **Detection module** has two traces to calculate the loss of bounding box location and class separately. Three Detect modules top-to-bottom in the head are specialized in small, medium, and large objects respectively.

The detailed structures of these modules are shown in Figure 2.5

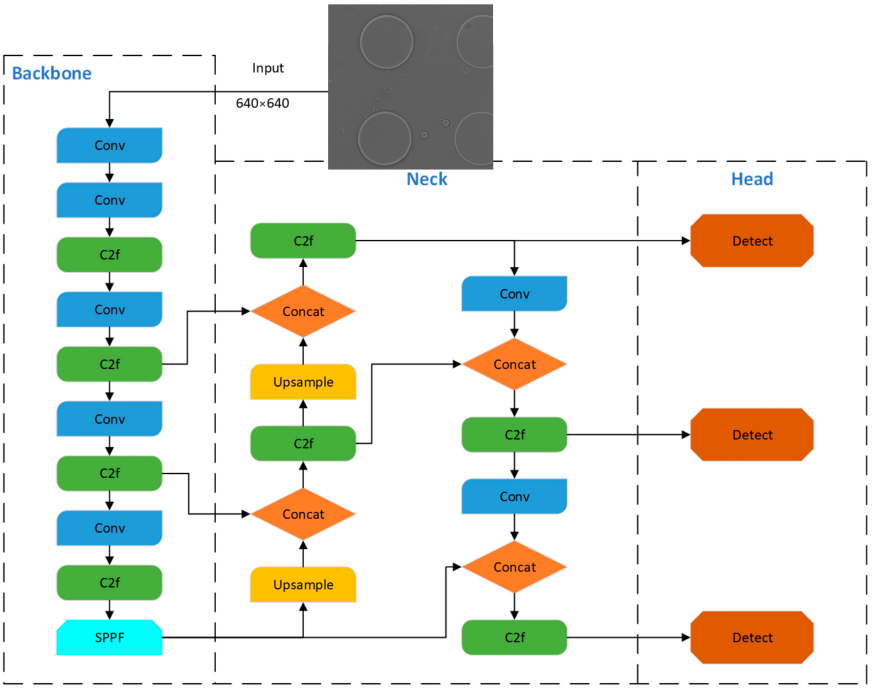


Figure 2.4 The architecture of YOLOv8 cited from [Zhai et al., 2023].

2.4 Model Improvements

Deep learning models are generally tested on the COCO dataset, which contains 80 categories of common objects in our daily lives. Therefore, even though these state-of-the-art models have very high accuracy in the paper, it is still more likely to perform poorly in microbiological applications, as they are quite different from the common test dataset. Therefore, to have a better performance on our custom dataset, some modifications of YOLO architecture are tried in our work.

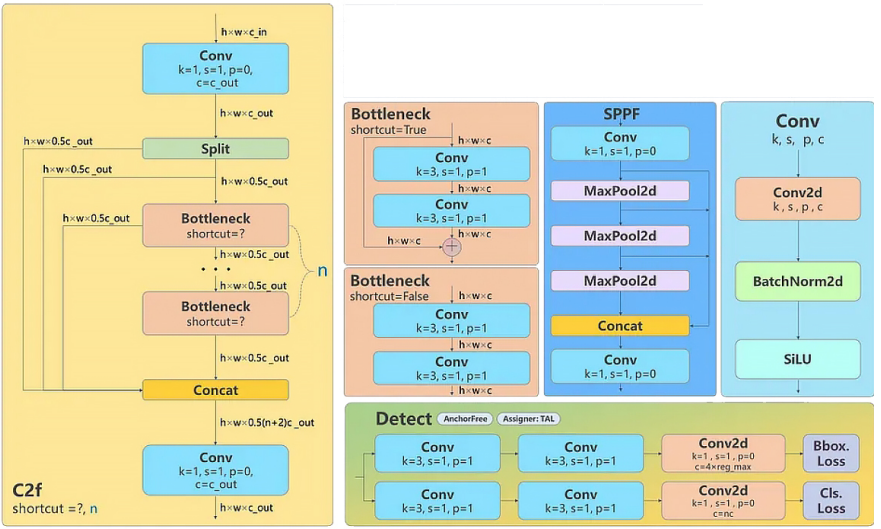


Figure 2.5 The details of the modules used in YOLOv8. [Jocher et al., 2023]

Transfer Learning

Transfer learning is a typical method in the field of machine learning, referring to training a model based on a previous, related task [Zhuang et al., 2021]. Transfer learning helps the model to quickly adapt to a new dataset with few resources required by reusing old models. It is used to reduce training time, improve model performance, solve unbalanced dataset problems, and improve model generalization ability [Sharma, 2023].

Transfer learning can be categorized into four ways based on different kinds of technical approaches: sample-based transfer, feature-based transfer, model-based transfer, and relation-based transfer [Devan et al., 2019]. In our work, we tried model-based transfer learning, which shares parameters between new and pre-trained models. In general, in a machine learning model, neural networks learn edge features in the first layer, shape features in the middle, and task-specific features in the latter layers. For most objects, similar edge features are shared. Objects from similar fields also have quite similar shape features (such as cells in biology). Transfer learning uses these properties by freezing some layers to keep parameters from the pre-trained models, training only on the remaining layers. Depending on the number of frozen layers, there are three standard transfer learning approaches: The first method freezes all of the convolutional layers from the pre-trained model and only trains the fully connected layers. The second method freezes partial convolutional layers from the pre-trained model and trains the remaining convolutional and fully connected layers. The third method trains the whole network based on the previously trained model parameters [Devan et al., 2019].

Deformable Convolutional Network (DCN)

A major problem in computer vision is that CNNs are unable to adapt to geometric alterations and transformations. CNNs try to overcome this challenge by augmenting datasets or applying transformation-invariant features and algorithms. However, this acquires prior knowledge about transformations, so it only works for known and fixed variations. It also adds the complexity of the model, which needs more computational resources. Therefore, a more practical approach to addressing this issue is to use DCN, which attempts and succeeds in adapting to extraordinary alterations.

For the regular 2D convolution, it samples over the input feature map using a grid R, which defines the receptive field size and dilation. Then, for each location p_0 on the feature map y, we have

$$y(p_0) = \sum_{p_n \in R} w(p_n) \cdot x(p_0 + p_n), \tag{2.4}$$

where p_n enumerates the locations in R. For deformable convolution, we introduce the grid R augmented by the offsets $\triangle p_n$, which represents that the grid R is irregular. Then Equation (2.4) in the deformable form can be calculated by

$$y(p_0) = \sum_{p_n \in R} w(p_n) \cdot x(p_0 + p_n + \triangle p_n). \tag{2.5}$$

In practice, the offset $\triangle p_n$ is fractional, to estimate the value of $p_0 + p_n + \triangle p_n$, Equation (2.5) is calculated with bilinear interpolation:

$$x(p) = \sum_q G(q, p) \cdot x(q), \tag{2.6}$$

where G is the interpolation kernel, q enumerates all original spatial locations in the feature map x, and p denotes an arbitrary location ($p = p_0 + p_n + \triangle p_n$) [Dai et al., 2017].

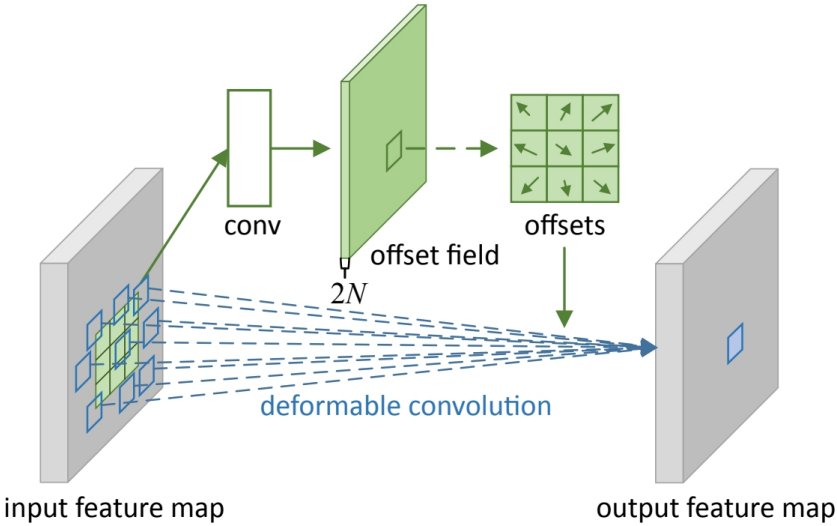


Figure 2.6 An illustration of 3×3 deformable convolution. Figure from [Dai et al., 2017].

The illustration of a 2D 3×3 deformable convolution is shown in Figure 2.6. The offsets are calculated by applying a convolutional layer to the same feature map, and the output offset fields are of the same spatial resolution as the feature map. The offsets and the convolutional kernels that produce the output features are learned at the same time during training.

Realtime Detection Transformer (RT-DETR)

For a traditional object detector, it is normal to use anchor boxes to locate objects and NMS to eliminate overlapping bounding boxes. However, these two steps are always time-consuming due to their complex architecture. Therefore, end-to-end object detectors, which drop the use of anchors and NMS, are developed. The first end-to-end object detector, called DETR (DEtection TRansformer), was initially proposed by Carion et al. [Carion et al., 2020b]. It predicts one-to-one item sets directly by use of bipartite matching. By simplifying the detection pipeline, DETR addresses the NMS-related performance barrier. However, there are some other problems for DETR, such as slow training convergence and difficulty with query optimization.

RT-DETR, the first real-time end-to-end object detector developed by Baidu, provides real-time object detection ability while also having really high accuracy. It uses Vision Transformers (ViT) to separate intra-scale and cross-scale fusion, allowing for the effective processing of multiscale data. As is shown in Figure 2.7, the RT-DETR model consists of a backbone, a hybrid encoder, and a transformer decoder in the head. The encoder takes the features from the last three layers $(S3, S4, S5)$ in

the backbone as input, then combines intra-scale interaction and cross-scale fusion to transform multi-scale features into an image feature sequence. Then, a defined number of image features from the encoder output are extracted by IoU-aware query selection and serve as the initial queries for the decoder. These queries are iteratively improved by the decoder and additional prediction heads, to generate object boxes and confidence ratings.

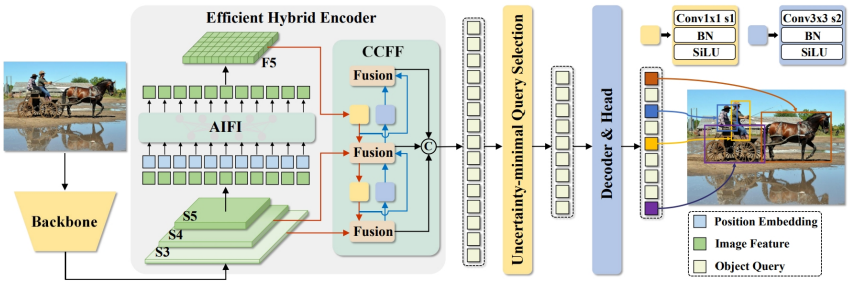


Figure 2.7 The architecture of RT-DETR. Figure from [Zhao et al., 2023].

Wise-IoU

The loss function for bounding box regression (BBR) is essential to object detection tasks. Most BBR loss functions are designed under the assumption that the training datasets have high quality. However, it is possible that the training datasets are of low quality, which will lead to low loss function performance. Therefore, Wise-IoU is designed to solve this problem, by evaluating the quality of anchor boxes and providing a wise gradient gain allocation strategy. This enables Wise-IoU to concentrate on anchor boxes of average quality and enhance the overall performance of the object detector [Tong et al., 2023].

When a training dataset has low-quality samples, geometric parameters like aspect ratio and distance will affect the generalization ability as the penalty for low-quality examples will degrade. Therefore, Wise-IoU develops a loss function to reduce the penalty of geometric factors when the anchor box and target box coincide well:

$$
\begin{aligned}
L_{WIoUv1} &= R_{WIoU} L_{IoU} \\
R_{WIoU} &= exp(\frac{(x-x_{gt})^2 + (y-y_{gt})^2}{(W_g^2 + H_g^2)^*})
\end{aligned}
\tag{2.7}
$$

where W_g, H_g are the size of the smallest enclosing box and the superscript $*$ means that W_g, H_g are detached from the computational graph to avoid bad gradient during training [Tong et al., 2023].

Wise-IoU can also be applied with Focal Loss:

$$L_{WIoUv2} = (\frac{L^*_{IoU}}{\overline{L_{IoU}}})^{\gamma} L_{WIoUv1} \tag{2.8}$$

2.5 Evaluation Metrics

The two most significant metrics for object detection are accuracy (including classification accuracy and localization accuracy) and speed.

Precision P and Recall R

Precision P evaluates the model's ability to give correct predictions, by calculating the accuracy of the detections, which is the proportion of true positives among all positive predictions. On the other hand, recall R measures the model's ability to retrieve all instances instead of missing some of them, by calculating the proportion of true positives among all actual positives.

Mean Average Precision (mAP)

A common evaluation metric used to evaluate classification accuracy is average precision (AP). AP summarizes the precision-recall curve for a single class by computing the area under the curve (AUC), providing a single value that measures the model's precision and recall performance. mAP is the mean of the APs calculated for all classes. It provides a single number that indicates the overall precision-recall performance of the model across all classes.

Intersection over Union (IoU)

A common evaluation metric used to evaluate localization accuracy is Intersection over Union (IoU). IoU calculates the overlap area between the ground truth bounding box and the predicted bounding box. It is defined as the area of the intersection divided by the area of the total area covered by the union of the two boxes. IoU values range from 0 to 1, where 0 means no overlap between the predicted and ground truth boxes and 1 means perfect overlap between two bounding boxes. A higher IoU indicates better localization accuracy. Typically, a threshold (e.g., 0.5) is set to determine whether a prediction is a true positive.

mAP at IoU thresholds

In practice, mAP is often calculated with multiple IoU thresholds to assess both classification and localization accuracy. Typically, we use mAP@0.5 which evaluates AP at an IoU threshold of 0.5, and mAP@0.5:0.95 which evaluates the mean of mean AP across multiple IoU thresholds varying from 0.5 to 0.95 in steps of 0.05. It provides a comprehensive evaluation of the model's performance across different

detection difficulties. Such metrics provide a robust criterion for assessing object detection models' ability.

3

Data

3.1 Original Data

In this work, we aim to track and detect 9 classes of protists in the videos: amoeba, apusomonas, ciliate, flagellate, monothalamid, nematode, colpodea ciliate, hypotrich and cysts. All the data we used in this work were collected in a soil chip project conducted by P. Micaela Mafla-Endara [Mafla-Endara, 2023], who recorded data in the form of videos captured with a microscope in three different chips with the same design, different by treatment (see below). All the videos were taken using a Nikon Ti2-E inverted light microscope with a Nikon Qi2 camera in the lab. The length of captured videos spans from tens of seconds to several minutes, each containing only one or a few protists, and some with camera movements. Depending on the treatments, some videos turn off the light for several seconds to find the position of fluorescent nanoplastics. As the protists are totally black, except with some nanoplastics inside their body, we decided to manually trim the clips after the lights were turned off.

There are 2-3 control groups in each chip, and the other groups received different treatments concerning different concentrations (0.5, 2, and 10 mg/L) of engineered nanoplastics. The treatments of all the data we use is shown in Table 3.1. There are several different videos under one treatment for the same or different protists. For example, *A_ENTRY_C001* and *A_ENTRY_C002* captured a nematode and a cyst separately, *B_ENTRY_F001* and *B_ENTRY_F002* are both about a monothalamid. The length of the videos depends on the type of protists. In general, amoebas have much longer time over 60 seconds, while ciliate and colpodea ciliate have only several seconds as they usually move very fast.

3.2 Labelled Data

A machine learning model learns features according to labelled images, so we need to label the protists in the videos under image format. The tool we used for labelling is called Roboflow [Dwyer et al., 2024], which is a platform designed to simplify

Table 3.1 Overview of original data.

Chip	Video Name	Treatment
A	A_ENTRY_A	Control
	A_ENTRY_B	Control
	A_ENTRY_C	0.5mg/L 60 nm beads
	A_ENTRY_D	0.5mg/L 60 nm beads
	A_ENTRY_E	2.0mg/L 60 nm beads
	A_ENTRY_F	2.0mg/L 60 nm beads
	A_ENTRY_G	10.0mg/L 60 nm beads
	A_ENTRY_H	10.0mg/L 60 nm beads
B	B_ENTRY_A	Control
	B_ENTRY_B	Control
	B_ENTRY_C	0.5mg/L 60 nm beads
	B_ENTRY_D	0.5mg/L 60 nm beads
	B_ENTRY_E	2.0mg/L 60 nm beads
	B_ENTRY_F	2.0mg/L 60 nm beads
	B_ENTRY_G	10.0mg/L 60 nm beads
	B_ENTRY_H	10.0mg/L 60 nm beads
C	C_ENTRY_A	10.0mg/L 60 nm beads
	C_ENTRY_B	Control
	C_ENTRY_C	Control
	C_ENTRY_D	10.0mg/L 50-60 nm beads
	C_ENTRY_E	Control
	C_ENTRY_F	10.0mg/L 50-60 nm beads
	C_ENTRY_G	10.0mg/L 60 nm beads
	C_ENTRY_H	10.0mg/L 200 nm beads

and streamline the process of managing and preparing image datasets for machine learning tasks. Roboflow can upload and sample videos allowing manually chosen *frames/second*. We sample the videos *2 frames/second* except for amoeba, as they usually move really slowly so we often choose *1 frame every 2 seconds* to reduce similar images in the dataset. The number of videos and corresponding images we firstly used is shown in Table 3.2. From the table we can find some of the classes like amoeba and (colpodea) ciliate, appear more frequently than the others like flagellate, leading to unbalanced data distribution. Therefore, to avoid the imbalanced training dataset affecting the performance of our model, we augment the classes nematode, cysts, apousomonas, flagellate and hypotrich by 5 times and add to our dataset, and use GANs to generate some "fake" flagellates, cysts and colpodea ciliates. The dataset is then split into the training dataset, validation dataset, and test dataset by the ratio $70 : 20 : 10$.

During training, we found the quality of used videos and labelled data affecting

Table 3.2 The initial number of videos and sampled images.

Class	Number of Videos	Number of Images
amoeba	19	1614
apusomonas	3	77
ciliate	3	267
flagellate	4	52
monothalamid	6	238
nematode	4	86
colpodea ciliate	6	114
hypotrich	1	46
cysts	5	32

the model performance a lot. We will talk about in detail in the following Dataset Analysis part. So we modify the dataset step by step as we will clarify in Chapter 4.1, and the final version of the dataset we used is shown in Table 3.3.

Table 3.3 The final dataset we used in our work.

Class	Number of Images
amoeba	947
apusomonas	267
ciliate	340
flagellate	341
monothalamid	178
nematode	350
colpodea ciliate	244
hypotrich	188
cysts	280

3.3 Dataset Analysis

From Table 3.3 we can find that a problem of our dataset is the insufficient and unbalanced numbers among classes due to the limited video sources. However, even though we are that lack of data, during labelling and training, we still have to drop some videos for better performance due to the following reasons.

The first problem we find is to define the appropriate species of protists for the model. For example, the species we named "ciliate" and "colpodea ciliate" both belong to the phylum of Ciliophora. As there are potentially 27,000 - 40,000 existing species of ciliate [Hawksworth, 2009], it is impossible to assign different names to all different species. Some needs to be assigned the same general label, such as

ciliate. For us human beings, we can identify their class by the hair-like organelles called cilium easily. However, this is really difficult for a deep learning model to learn as the shapes of different species of ciliates vary a lot, as shown in Figure 3.1. We therefore have to split "ciliate" into two classes to avoid confusing the model. The same problem also occurs for "amoeba" and "flagellate". This is a specie called "amoeboflagellate", which seems almost the same as *amoeba* with a flagellum but belongs to $flagellate$. If we categorize it into $flagellate$, the model will be confused by the difference between *amoeba* and $flagellate$ so the model will have really bad accuracy on these two classes. Therefore, we decided to drop the videos of $amoeboflagellate$.

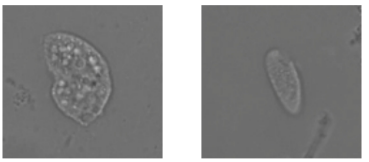


Figure 3.1 The example of two kinds of ciliates. In our dataset, the left one is labelled as colpodea ciliate and the right one is labelled as ciliate.

Another problem also happens to these classes with "hair". An easy and reliable way to identify the protists in our dataset is to find if they have (and the number and length of) flagellum or cilium. For a video, this task is not difficult. But for an image, it may fail to see their flagellum or cilium due to their movement or the resolution and focus of the microscope camera. This situation becomes worse when labelling. Almost half of sampled images of $flagellate$ and $ciliate$ cannot see their flagellum and cilium in the images, which lead to make the model fail to learn this important feature during training.

Figure 3.2 Example of failing to see flagellum or cilium in the sampled images.

The last problem we find in our dataset is caused by the background. Besides protists we need to detect, there are also few other species in our videos that are not included in our detection work. There are also some dead protists, bacteria, fungi, nanoplastics and other undetermined things in the background. For example, Figure-3.3 is a sampled image with a cyst. However, we can see it really looks like the

nanoplastics outside the segmentation, but it has a shell. At the top left corner, there are also a "shell" and an "amoeba" that look like our target protist. This problem happens quite often in almost all the videos, and it is impossible for us to label all of them. What's more, we find amoebas sometimes prefer to move with nanoplastics (Figure 3.4). This situation is really worth studying. But this kind of amoeba looks quite different from a common amoeba, which is generally with a "clear" body as shown in Figure 1.1. Without this video, the model will treat them as background. But with labelling, the model will be confused by *amoeba* and background during training. So during the training process, the model finds it hard to tell the difference between the background and our target objects with these problems, which causes our model to sometimes fail to find the protists, or label the objects belonging to the background.

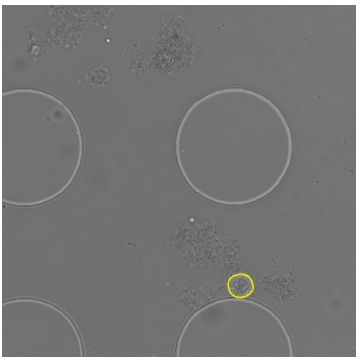


Figure 3.3 An example of complicated background. Besides our target object within yellow circle, there are lots of other objects such as particles, nanoplastics, dead body etc.

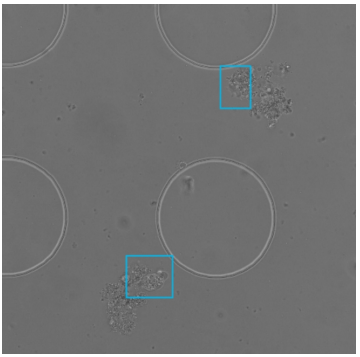


Figure 3.4 A video captured two amoebas moving with nanoplastics.

4

Methods

4.1 Data Pre-processing

At the beginning of this project, we were given a dataset with 45 videos, which needed to be organised and labelled by ourselves. At first, we labelled all data into 9 classes: *amoeba, amoeboflagellate, apusomonas, ciliate, flagellate, cysts, monothalamid, nematode, hypotrich*. The dataset was also augmented by rotation, crop, blur, noise, and brightness in Roboflow before training. However, we found that the accuracy of $amoeba, amoeboflagellate, flagellate$ and $ciliate$ was quite low. So we had to go back to pre-process our dataset. We thought the precision of bounding boxes affected the model performance a lot. Therefore, we totally modified our dataset. Firstly, we relabelled all images of $amoeboflagellate$ to *amoeba* or $flagellate$, unless the cases where they looked really different from these two classes. Then we deleted a large amount of low-quality *amoeba*, as they have been already over-represented. Then we split the original $ciliate$ class into two classes: $ciliate$ and $colpodeaciliate$, as these two species look really different from each other.

After solving these problems that occurred in the original dataset, we needed to solve another important problem: the lack of data.

4.2 Generate Synthetic Data

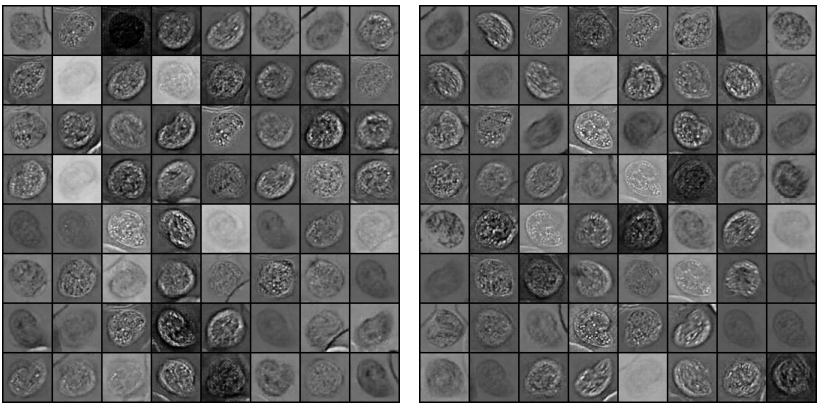
To remedy the problem of the lack of data, we tried to use generative models to generate synthetic data, especially for those classes that have much less amount of samples compared to the others, such as $colpodeaciliate$ and $cysts$.

There exist a lot of generative models, including GAN-based models, VAE-based models, and Diffusion-based models. During this thesis project, we tried all of them, however, only GAN-based models satisfied our requirements: to generate images of protists with various shapes while maintaining genuine patterns and looking reasonable.

We first tried to use vanilla GAN and DCGAN, which are known for their ability to generate realistic images. However, GAN-based models are also famous for

their difficulties in training, including non-convergence and mode collapse. When we tried to train the vanilla GAN and DCGAN, they both failed to converge. The loss of the generator kept going up during training because the poor data distribution learnt by the generator in the early stage during training could be easily distinguished by the discriminator. The discriminator became almost perfect and rarely made mistakes. The loss of discriminator therefore approached 0. The training of G and D never became balanced again. After 10000 iterations, the generated images were still all noise without any information.

To solve the training difficulties for GANs, we tried to use other more advanced models, especially WGAN and its modified version WGAN-GP as described in Ch.2. WGAN and WGAN-GP use an alternative value function derived from an approximation of the Wasserstein-1 distance $W(q, p)$, which is informally defined as the minimum cost of transporting mass. This alternative value function can provide more stable gradients for the generator to update which makes training less challenging. When we tried to train a WGAN model, it was able to generate reasonable images which were much better than vanilla GAN and DCGAN. However, due to the weight clipping on the discriminator during training, the WGAN models improved very slowly. After 10000 iterations, the generated images look reasonable to some extent, we can recognize them as some kind of micro-organisms, but still lack details in shape and all the generated samples look similar, which means the generator has only learnt some simple distribution.



(a) (b)

Figure 4.1 Ciliate samples generated by WGAN-GP. They have convincing quality, fine details and natural textures.

When we added a gradient penalty and trained a WGAN-GP model, the generated samples improved much faster. After only around 1500 iterations, the generator was able to generate reasonable images that outperformed the WGAN model.

After 5000 iterations, the generator had been able to generate visually convincing samples, which had fine details in shape and colour, natural textures, various orientations, locations, and lighting conditions. The generator even generated some samples with the structure pillar which appeared a few times in the training dataset.

Finally, we trained three WGAN-GP models for ciliate, cysts and flagellate, to solve the unbalanced data problem. We used them to generate 100 samples for each class. We did not generate more images for hypotrich and monthalamid because they are relatively large in size compared to other protists, and our detection model was already able to detect them very well. The synthetic images are convincingly realistic and we can not distinguish them from the real data, some examples of generated images of ciliate are shown in Figure 4.1.

4.3 Model Training

We use computational resources Alvis cluster provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) for training in this work.

We mainly use NVIDIA Tesla A40 GPU with 48GB RAM or NVIDIA Tesla A100 HGX GPU with 40GB RAM to train YOLO models, depending on the model complexity.

The training hyperparameters are chosen as follow: epochs=[100, 150, 200], image size=640, batch size=[64, 128, 256, 400], dropout=0.3, weight decay=0.0005, warmup epochs=3, warmup momentum=0.8. We use AdamW as optimizer with learning rate=0.000714, momentum=0.9.

YOLO

Transfer Learning. In the training phase, we began with training from new models. This worked but took many more epochs, at least 400 epochs, to have a good performance, so we tried to use transfer learning to reduce training time. Unfortunately, we failed to find any open-source dataset on protists or microorganisms, so we had to use the pre-trained models from YOLO which have been pre-trained on the COCO dataset [Lin et al., 2014]. After running a few experiments on the same datasets with the same hyper-parameters, only different in model size, we got a comparison between the n, s, m, l pre-trained models. Considering the accuracy, speed and computational cost, we choose $YOLOv8s.pt$ as our baseline model. After applying transfer learning, training time reduced significantly to around 150 epochs, and the mAP also increased by 4%.

Improvements. To improve the performance of our model, we also tried to modify the structure of YOLO. The first idea we came up with was to use DCN, see Ch.2.4. The common CNN only has fixed sampling points, which makes it hard to learn the features of objects with irregular shapes. But for the class *amoeba* in our datasets, they do not have any regular shapes and change their shapes all the time. So we implemented the DCN to try to improve the ability to learn irregular features.

Then we also tried to modify the loss function of YOLO. There are some samples that are not that good or typical, we hope our model does not pay much attention to these kinds of data. Therefore, we tried to apply Wise-IoU to make the model have dynamic bounding box regression loss to avoid this problem.

At last, as the Transformer is the most popular architecture these years, we also wanted to add it to our model. However, it is difficult for traditional Transformers to run in real-time, especially compared with YOLO. Therefore, we found an architecture named RT-DETR, which can fuse feature maps in different sizes and use convolution to make the model more efficient for working in real-time object detection tasks.

Two-step YOLO

After training our model and evaluating it on the test dataset, we also applied our model to totally new datasets: such as videos with different microscope camera settings, or protists in really complicated backgrounds. Then we found that our model had bad generalization ability on these kinds of datasets. One of the reasons why this happened we assumed is that there were too many objects in the background that seemed similar to our target protists, and then the model was confused by the missing of these objects' bounding boxes. We thought this problem could be solved by labelling our data in more detail, like labelling all confusing objects in the backgrounds with "Unknown" or something else. However, it is impossible to label all such things for 3,000 images. Therefore, instead of labelling all images, we hoped to let our model learn the features of backgrounds, and then isolate our target protists from backgrounds.

Based on the above idea, we constructed our model as a two-step YOLO model: first, a detection model to find out all protists (or moving objects), then a classification model to classify all the isolated objects from the detection model. In detail, first, we modify our dataset: convert all the labels to one label "Moving" in the detection project, and then create a new classification project with all the targets without backgrounds in it. Then we train a detection model and a classification model using these two datasets separately. After having satisfied models, the detection model is used as the first step to detect all the protists, then the current frame is cropped according to the bounding box coordinates. The cropped images are sent to the second classification model to predict a class, and finally, the class and bounding box are drawn on the original current frame. What's more, to keep the predicted class stable, we record the tracking IDs of all detected amoebas in one video and choose the most often predicted class so far as the current label instead of using the current predicted class directly.

The two-step YOLO model has two significant advantages: better generalization ability and more stable predicted results. As this model learns the features of backgrounds to some extent during the first training, it can be easier to find out the positions of protists rather than being distracted by complicated backgrounds in

new videos. What's more, as we know a classification task is usually more accurate than a detection task. Therefore, as long as the model can locate the right positions of the protists, the second classification model has a higher probability of returning the right class compared with the one-step model. Moreover, as the shapes of many protists are not fixed, the predicted class may change as the protists move. Therefore, compared with the one-step model which uses the predicted class directly, the two-step model can use algorithms to control the stability of the predicted class as we mentioned before.

4.4 Camera Motion Compensation

To analyze the movement and speed of protists, we need to know the absolute position of each protist in the soil chips, and then sum up the distances between each frame to represent the moving distance or speed. To capture the details of the protists clearly, our microscope camera usually uses a large magnification. Therefore, the camera has to move to keep capturing the protists when they are likely to move out of our screen. However, YOLO only returns relative positions of the bounding boxes, which leads to wrong calculations while the camera moving. Therefore, compensating for the camera movement is necessary. The method we use for compensating is Fast Fourier Transform (FFT).

Fast Fourier Transform (FFT)

The Fast Fourier Transform is an important algorithm in signal processing that computes the Discrete Fourier Transform (DFT) of a sequence, or its inverse (IDFT). FFT translates signals from their original domains, such as time or space, to frequency domain representations with exceptional efficiency. The DFT is created by decomposing a sequence of values into components with distinct frequencies [Heideman et al., 1984]. However, it is sometimes too slow to compute the DFT directly from the specification in practice. Therefore, an FFT computes such modifications quickly by converting the DFT matrix into a product of sparse (mainly zero) elements [Van Loan, 1992]. So the FFT reduces the computational complexity of the DFT from $O(n^2)$ to $O(nlogn)$, making it possible to be applied for real-time applications and large-scale data processing.

Given a sequence $x[N]$ of length N, FFT computes its DFT, $X[k]$ using the formula:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi nk}{N}} \tag{4.1}$$

Algorithms

As the camera moves, the pixel at (i, j) in one image will occur in the other image at $(i + dx, j + dy)$. To compensate for the camera motion, the algorithm

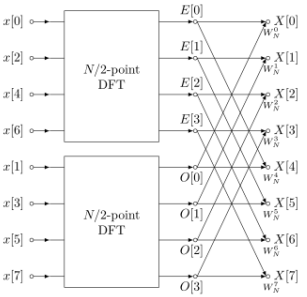


Figure 4.2 An example of FFT algorithm structure with a division into half-size FFTs [Wikipedia contributors, 2024a].

mainly has two functions: a $findTranslation()$ function to find (dx, dy), and a $compensateMotion()$ function to generate a "whole map" of the soil chip according to (dx, dy).

findTranslation() Function.

1. **Compute the 2D FFT of both images** to convert two images from spatial domain to frequency domain for further calculation.

2. **Compute the Cross-Power Spectrum according to**

$$R(u,v) = \frac{F_1(u,v) \cdot F_2^*(u,v)}{|F_1(u,v) \cdot F_2^*(u,v)|} \tag{4.2}$$

   Cross-Power Spectrum is designed to highlight the phase differences caused by the displacement between the images, and the normalization part ensures that the R contains only phase information while minimizing the impact of the images' amplitude information.

3. **Apply the Inverse 2D FFT** to go back to the spatial domain to get the correlation surface $r(x,y)$.

4. **Find the Peak.** The location (x_{peak}, y_{peak}) in $r(x,y)$ corresponds to the estimated displacement (dx, dy) between the two images. This works since the correlation is maximal when the two images are best aligned, and the position of the peak in the correlation surface corresponds to the relative shift (dx, dy) between the two images.

compensateMotion() Function.

- **Apply translation to the frame and update the old image buffer.**
- **The map size is 4200*4200, and the initial frame is set to the center of the map** so that it can work for movement in any direction.

- **Update the frame to the "map".** First add (dx, dy) to the center (r, c) to get the translation location(dr, dc), then update the region of interest in the map with the current frame.

5

Results and Discussion

5.1 Model Performance

GANs

In this section, we demonstrate and evaluate the performance of different GAN models as presented in Chapter 2.2. The vanilla GAN is known for its training difficulties, especially non-convergence and mode collapse. As shown in Figure 5.1, the training ended up with the vanishing gradient problem, and the generated images suffer from being pure noise.

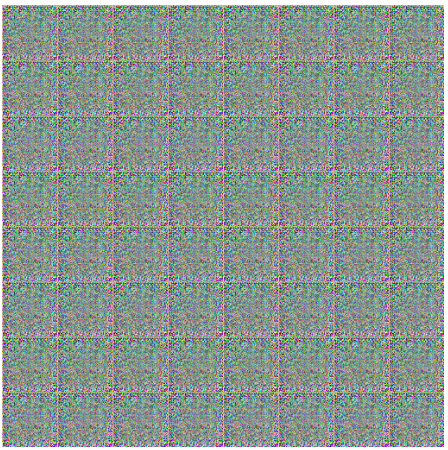


Figure 5.1 Ciliate samples generated by vanilla GAN. Training failed and only generated some pure noise.

The DCGAN model does not address the problem of training difficulties. When we train our model using DCGAN, it is still common to fail to converge and only produce noise images. With careful hyperparameter tuning, the DCGAN model can generate some poor-quality images occasionally, as shown in Figure 5.2.

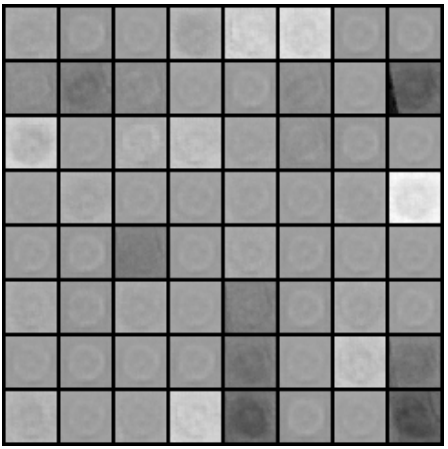


Figure 5.2 Ciliate samples generated by DCGAN. Their quality is very poor.

The WGAN model uses an alternative cost function to improve the stability when training GANs. However, the weight clipping method in this model can reduce the capacity of the discriminator, making it learn simpler functions. As shown in Figure 5.3, the generated images look reasonable to some extent, however still a bit noisy. This is because the learnt distribution lacks complexity.

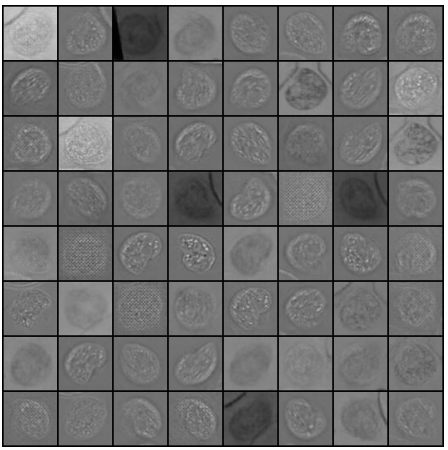
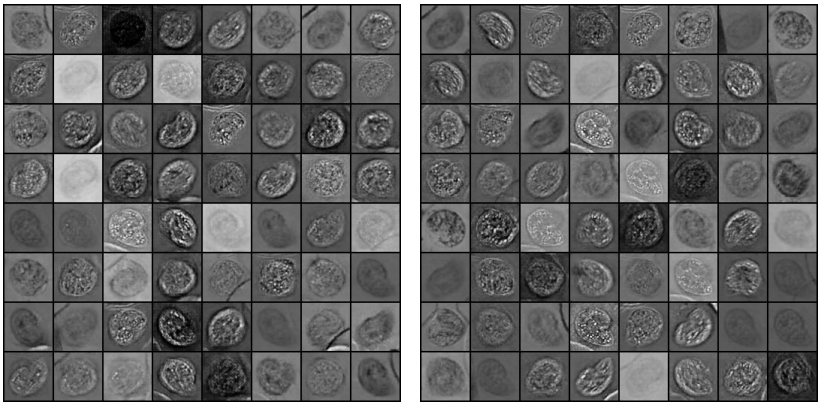


Figure 5.3 Ciliate samples generated by WGAN. They look reasonable to some extent, while still lacking quality.

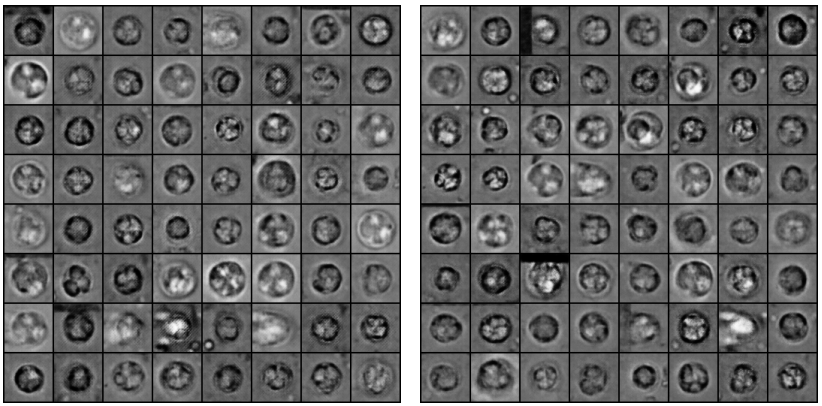
The WGAN-GP model uses an improved training method by introducing a gradient penalty term, greatly improving training stability even without careful hyperparameter tuning. It significantly outperforms weight clipping with improved per-

formance, faster training speed and higher sample quality. As shown in Figure 5.4 and Figure 5.5, the generated images of ciliate and cysts are convincingly realistic. They have fine details in shape and colour, natural textures, various orientations, locations, and lighting conditions.



(a) (b)

Figure 5.4 Ciliate samples generated by WGAN-GP. They look very realistic, with convincing quality, fine details and natural textures.



(a) (b)

Figure 5.5 Cysts samples generated by WGAN-GP. Their quality is convincing as well.

Model Comparison

To find the best model for our dataset, we start with training with the original YOLO structure. Table 5.1 shows the performance of the models we tested in our work. We can see that compared with training from a new model, using pre-trained models greatly improves the performance. For pre-trained models $YOLOv8s, YOLOv8m, YOLOv8l$, they have similar good performance, but the inference time increases a lot. Therefore, as a trade-off, we choose $YOLOv8s.pt$ as our best model.

Table 5.1 The performance of the one-step models without any modification in the architecture of YOLO, where FLOPs stand for Floating Point Operations, Params stand for Parameter Numbers. The file extension name represents different types of models: *.yaml* means training from scratch, *.pt* means using a pre-trained model. n, s, m, l represent different numbers of layers, and $n < s < m < l$ (the detailed parameters of these four models are shown in Appendix A).

Model Structure	mAP @0.5	mAP @0.5:0.95	Inference Time (ms)	FLOPs (B)	Params
YOLOv8n.yaml	0.959	0.803	1.1	**8.1**	**3007598**
YOLOv8s.yaml	0.967	0.837	**0.8**	28.5	11129454
YOLOv8m.yaml	0.973	0.843	1.5	78.7	25845550
YOLOv8l.yaml	0.973	0.839	2.4	164.9	43614318
YOLOv8n.pt	0.971	0.845	0.9	**8.1**	3007598
YOLOv8s.pt	**0.978**	0.863	1.5	28.5	11129454
YOLOv8m.pt	**0.978**	0.863	3.0	78.7	25845550
YOLOv8l.pt	0.968	**0.865**	5.0	164.9	43614318

Then we train the models with modifications in the architecture of YOLO. As DCN changes the backbone of YOLO, we cannot continue training on a pre-trained model, so we choose to train from $YOLOv8m.yaml$ since it has both good accuracy and acceptable inference time. W-IoU is trained on our best model $YOLOv8.pt$, and RT-DETR also uses the model $RT-DETR-L$ pre-trained on COCO dataset [Lin et al., 2014]. The performance of our modified models is shown in Table 5.2.

Table 5.2 The performance of the one-step models with modifications in the architecture of YOLO.

Model Structure	mAP @0.5	mAP @0.5:0.95	Inference Time (ms)	FLOPs (B)	Params
YOLOv8m.yaml + DCN	0.965	0.813	1.1	8.1	3007598
YOLOv8s.pt + W-IoU	0.967	0.837	0.8	28.5	11129454
RT-DETR	0.982	0.869	2.45	103.5	32002235

Table 5.3 The detailed information for each class of our best model *YOLOv8s.pt*.

Class	Box(P	R	mAP@0.5	mAP@0.5:0.95)
amoeba	0.938	0.955	0.982	0.841
apusomonas	0.991	0.937	0.975	0.779
ciliate	0.914	0.96	0.961	0.721
cysts	0.932	1	0.995	0.812
flagellate	0.89	0.881	0.906	0.779
hypotrich	1	1	0.995	0.976
monothalamid	1	1	0.995	0.949
nematode	0.959	0.983	0.986	0.977
colpodea ciliate	1	1	0.995	0.942

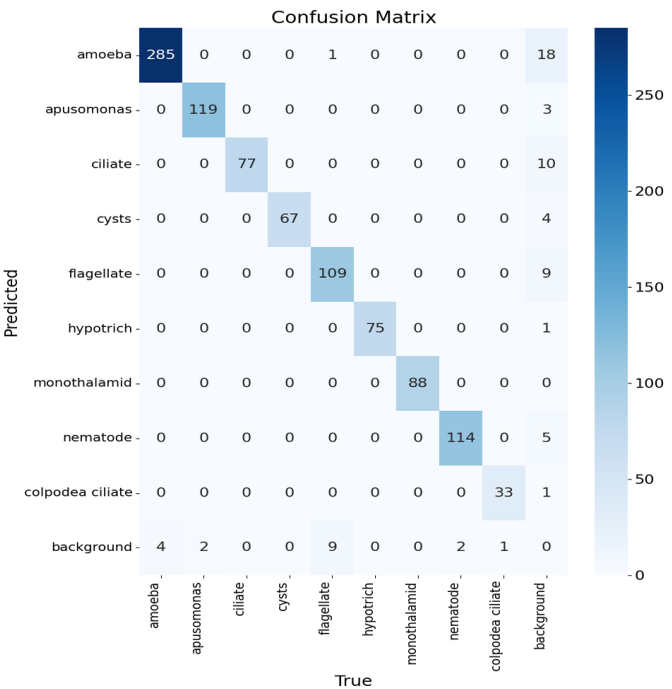


Figure 5.6 The confusion matrix of the model *YOLOv8s.pt*.

For the two-step YOLO model, the best detection model has *mAP@0.5 = 0.981* and *mAP@0.5:0.95 = 0.876*, and the best classification model is almost 100% correct for the test dataset. The confusion matrices of these two models are shown in Figure 5.8.

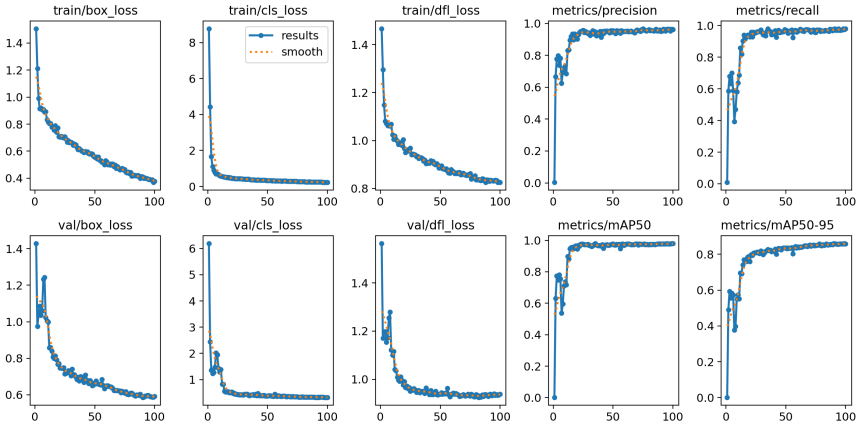
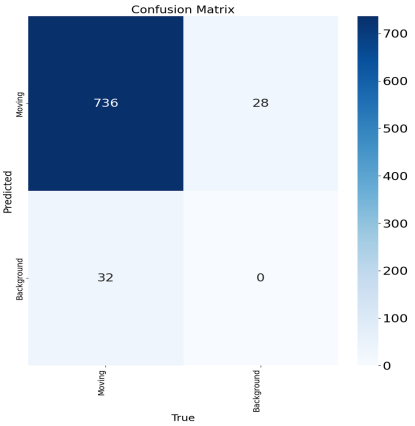
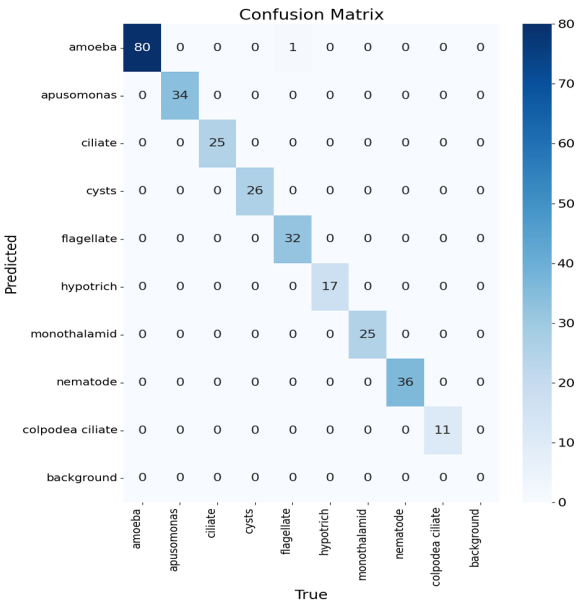


Figure 5.7 The loss curves of the model *YOLOv8s.pt*. "Box_loss" measures how much do the predicted bounding boxes fit the ground truth object, and it is usually a regression loss such as smooth L1 [Huber, 1992]. "Cls_loss" measures the correctness of the classification of each predicted bounding box, and it is often called cross entropy loss [Rumelhart et al., 1986]. "Dfl_loss" represents "Dual Focal Loss" and is used to solve class imbalance problem by putting more importance on less frequent classes during training [Lin et al., 2017], [Hossain et al., 2021].



(a) Moving object detection model



(b) Classification model

Figure 5.8 The confusion matrices of two-step model.

5.2 Data Analysis

In this part, we want to analyze the effect of nanoplastics on the protists, such as their speed and movement trajectory (trace). Among the nine target classes, *cysts* and *monothalamid* almost do not move in the videos, *nematode* is not a protist so we are not curious about it, and *hytorich* moves too fast to be well tracked so we cannot get the absolute positions of bounding boxes. Therefore, these 4 classes are dropped in the analysis part and only 5 classes remain: $amoeba, apusomonas, ciliate, flagellate, colpodeaciliate$, and 37 videos in total. However, another problem raises as shown in Table 5.4: except *amoeba*, we do not have data for all treatments. And even though it has videos under one treatment, it is not enough to get any reliable conclusion or hypothesis. Therefore, we decide to put our emphasis on analyzing the speed and traces among classes, and to show some figures with or without nanoplastics.

Table 5.4 The number of videos under different treatments we used for data analysis.

Class	Total Video Numbers	control	0.5mg/L	2mg/L	10mg/L
amoeba	14	2	3	2	7
apusomonas	3	0	0	2	1
ciliate	6	2	4	0	0
flagellate	5	1	0	0	4
colpodea ciliate	9	0	4	5	0

Camera Motion Compensation

The amplitude of the camera movement is species-dependent: the camera is almost fixed for *amoeba* and *apusonomas* as they move very slowly, and the camera moves very much for the rest of the classes, especially for *ciliate*. If the camera moves slowly and smoothly, the compensation algorithm works well, but for a few videos where the camera moves rapidly, our algorithm may fail to find corresponding points. But for most of the videos, even though the algorithm generated some distorted frames in the background, the movements of the protists are still smooth so we can still get the right speed. Some compensated examples are shown below. The algorithm described in Chapter 4.4 has been used to patch together consecutive frames in the video, compensating for the camera movement. The black areas represent parts not covered in any video frame by the camera.

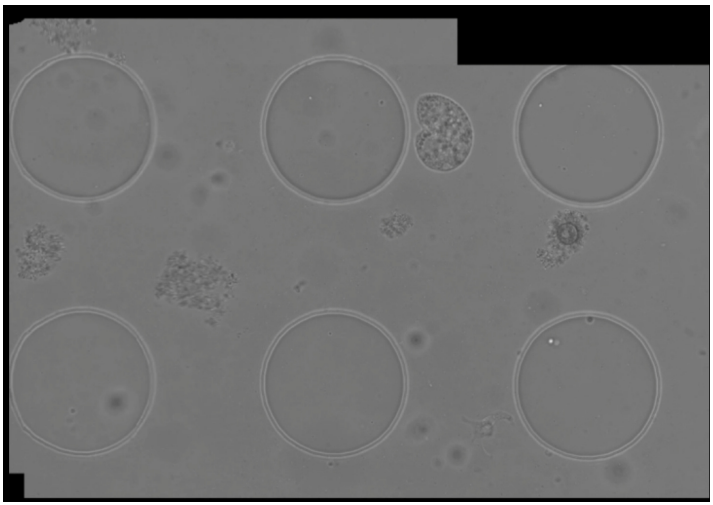


Figure 5.9 An example of slow and smooth camera movement. The generated "map" is really good almost without any distortion as can be seen on the non-distorted background.

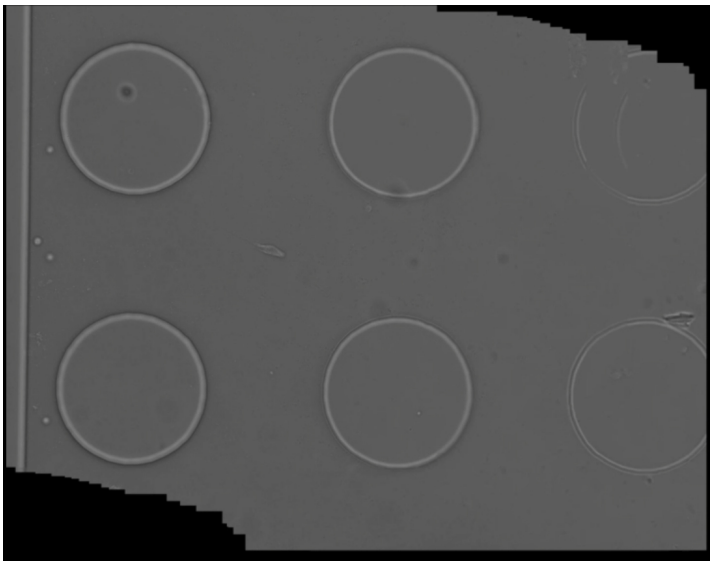


Figure 5.10 An example of a suddenly quick camera movement in the video. We can see that there are some distorted backgrounds at the top right corner, but the absolute positions of these frames are still quick right.

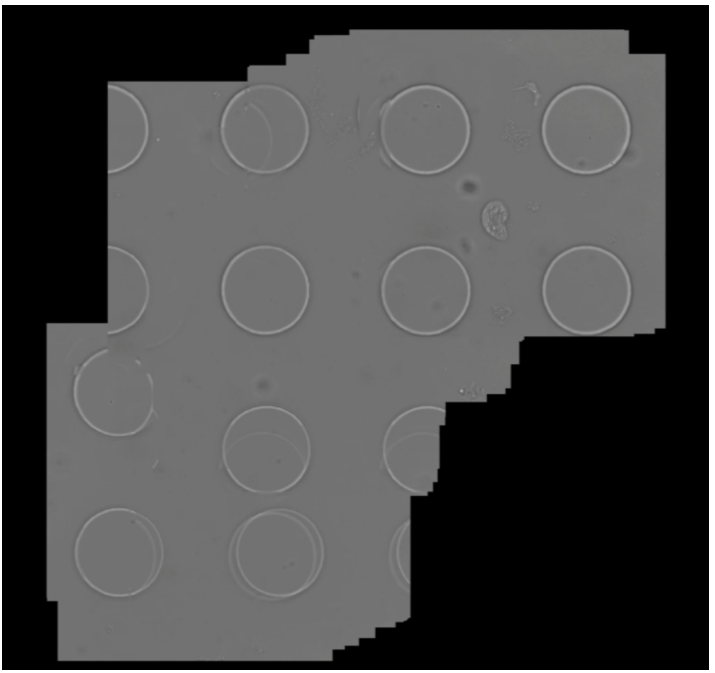
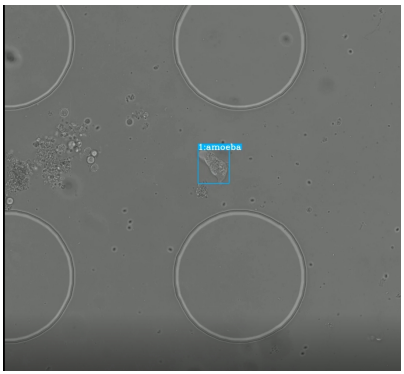


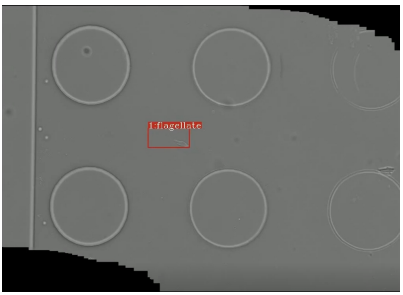
Figure 5.11 An example of a very quick movement of a ciliate, which leads to bad compensation when the camera fast moves as the ciliate. This can be seen by the irregular pattern of circular object (pillars) that in reality are positional in a regular grid.

Model Tracking Results

Figure 5.12 shows some examples of the results of our tracking model from the videos.

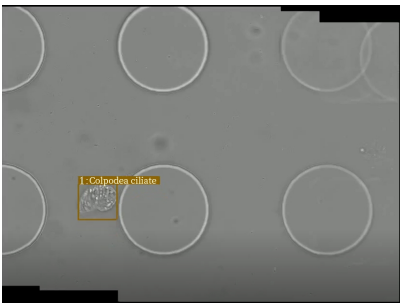


(a) amoeba



(b) flagellate

(c) ciliate



(d) colpodea ciliate

Figure 5.12 Some tracking examples detected by our model.

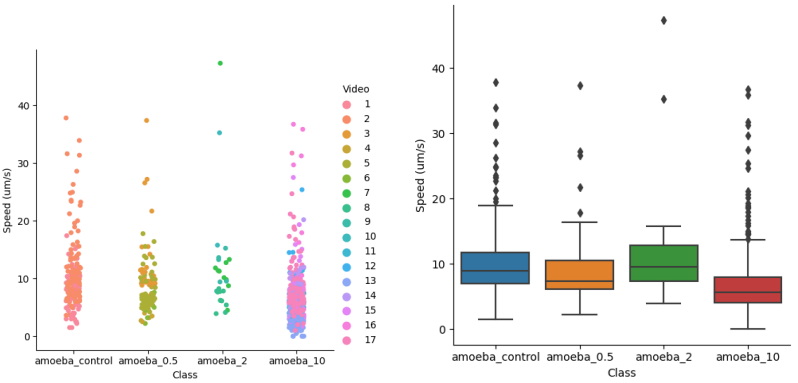
Speed Analysis

In this part, we compare speed over time among classes and speed distribution among classes to find the difference in speed between different species. What's more, to analyze the effect of nanoplastics, we plot figures of each class under different treatments.

To estimate the speeds, we record the center coordinates given by our model of all the bounding boxes in the videos and calculate the Euclidean distances divided by the magnification ratio between each adjacent center. The sum of every 30

distances is assumed to be a one-second distance (as the fps of each video is 30). The magnification ratios are not the same for different videos, so we use a software called *ImageJ* to measure the ratios manually (we are given that the diameter of each pillar is $100\mu m$).

Amoeba. From Figure 5.13, it seems that the speed of amoeba varies a lot under different environments and treatments as there are many outliers. As the only class which has all data under different treatments, we can see the speed distributions of $control$, $0.5mg/L$ and $2mg/L$ do not differ much, while $10mg/L$ has a significant negative effect on the speed of amoeba movements.



(a) (b)

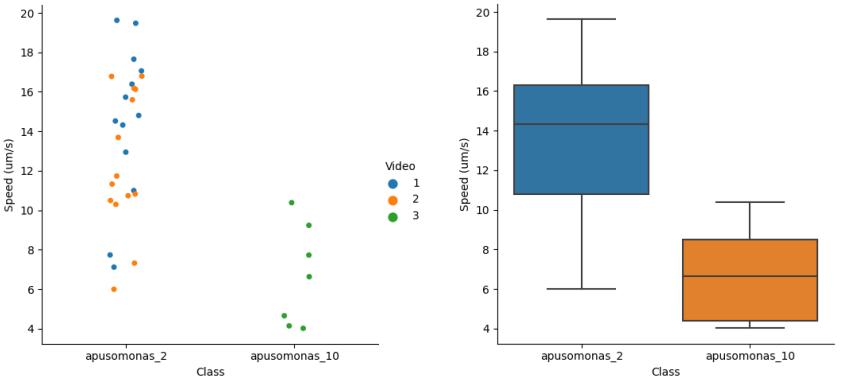
Figure 5.13 The plot of amoeba's speed distribution ($\mu m/s$) in 14 different videos. One colour represents one amoeba in the upper figure.

Apusomonas. From Figure 5.14, we can also find that the speed of $10mg/L$ is much slower than the $2mg/L$ treatment.

Ciliate. From the box figure of ciliate shown in 5.15, we can see there are almost no effects on ciliates under $0.5mg/L$ treatment, and ciliate moves much faster than amoeba and apusomonas. The speed around $400nm/s$ should be an error in our data.

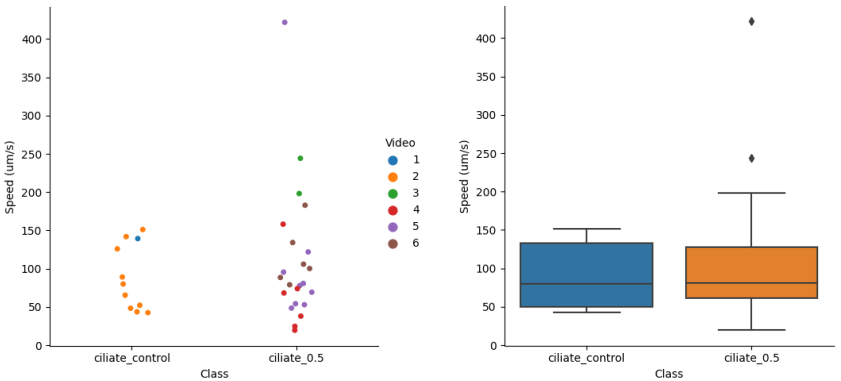
Flagellate. Figure 5.16(b) shows that the median of speed with $10mg/L$ is larger than the control environment. However, as we only have 3 data points in the control environment, we think we cannot say that the presentation of nanopasltics accelerates the speed of flagellate. But we can find that flagellates move faster than amoeba and apusomonas, but slower than ciliate.

Colpodea Ciliate. Quite similar as ciliate, colpodea ciliate also moves fast among these 5 classes. Even though we do not have any control data, it is clear that the speed $2mg/L$ is slower than with $0.5mg/L$ treatment. The speed seems to vary based



(a) (b)

Figure 5.14 The plot of apusomonas' speed distribution ($\mu m/s$) in 3 different videos. One colour represents one apusomonas in the left figure.

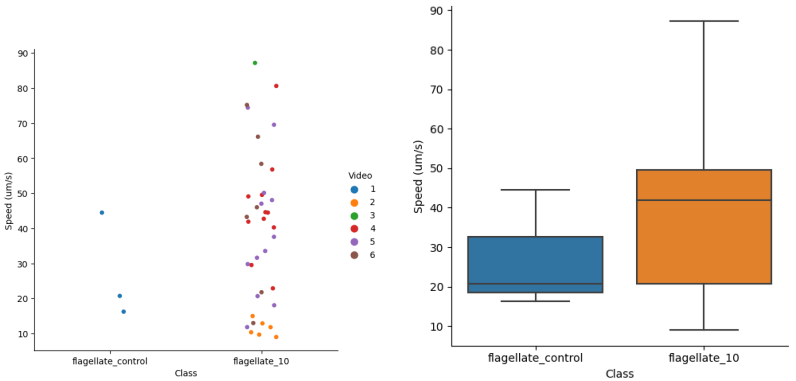


(a) (b)

Figure 5.15 The plot of ciliate's speed distribution ($\mu m/s$) in 6 different videos. One colour represents one ciliate in the left figure.

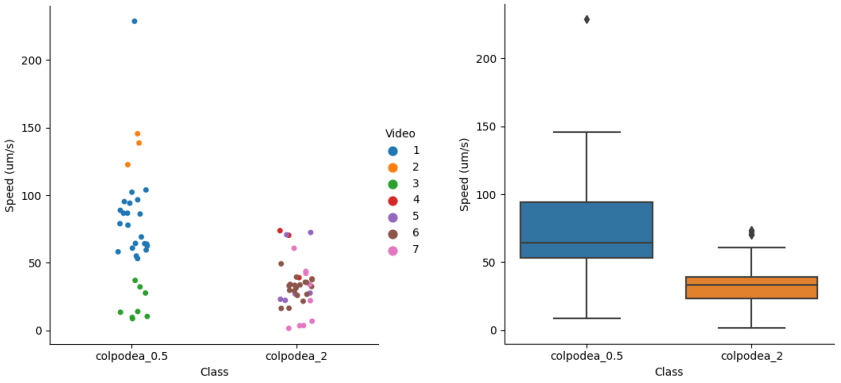
on different colpodea ciliates, as the speed dots are separate for different colpodea ciliates in Figure 5.17(a).

Speed Distribution among Classes. Because of the lack of data, we prefer to ignore the treatments and analyze all data among classes. Figure 5.18 shows the speed distribution among 5 classes. It is quite apparent that amoeba and apusomonas move the slowest, while both ciliate and colpodea ciliate have faster speeds. There are also quite a lot of outliers for amoeba.



(a) (b)

Figure 5.16 The plot of flagellate's speed distribution ($\mu m/s$) in 5 different videos. One colour represents one flagellate in the left figure.



(a) (b)

Figure 5.17 The plot of Colpodea ciliate's speed distribution ($\mu m/s$) in 9 different videos. One colour represents one Colpodea ciliate in the left figure.

Speed Comparison over Time. We also try to visualize the differences among classes over time, shown in Figure 5.19. As we expected, amoeba moves the slowest while ciliate moves the fastest.

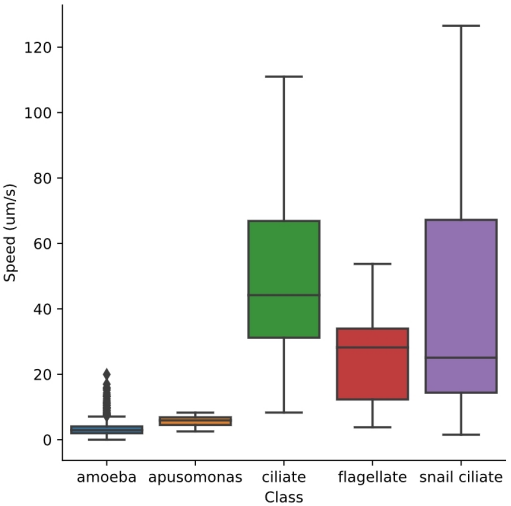


Figure 5.18 The speed distribution among 5 classes, all videos are included but we have removed some outliers for better visualization.

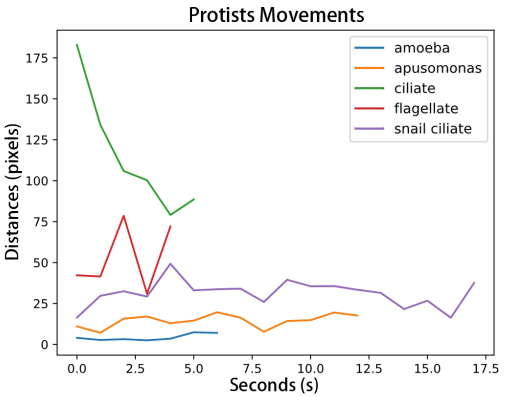
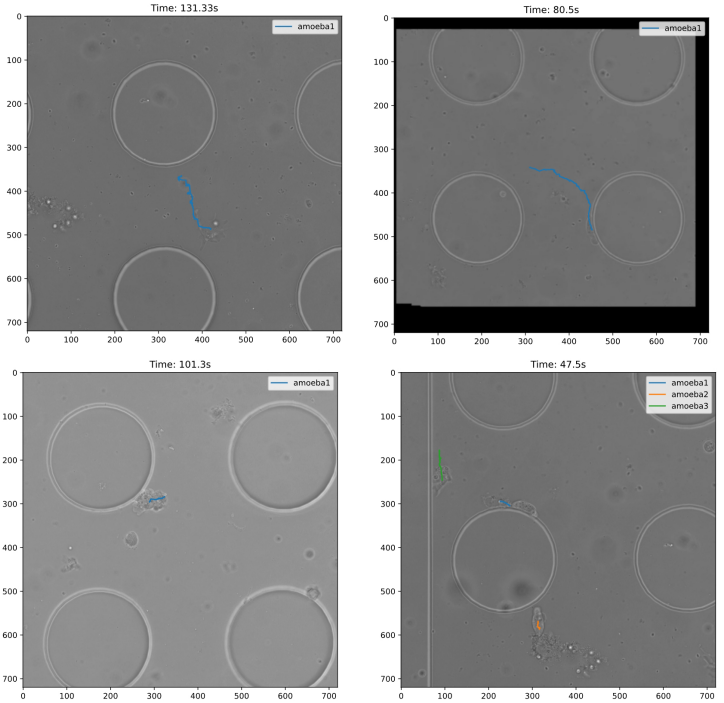


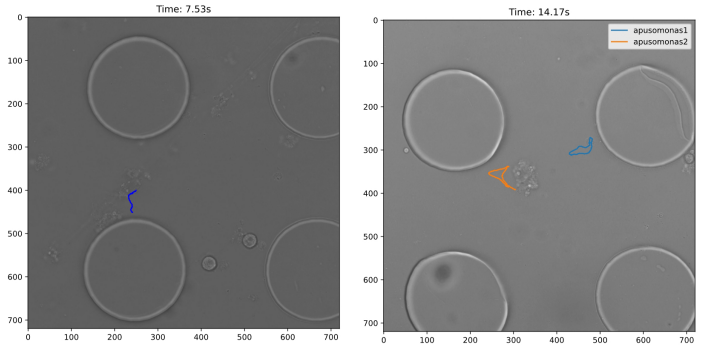
Figure 5.19 The speed comparison of 5 classes over time. The data are chosen from one video of each class randomly.

Trace Analysis

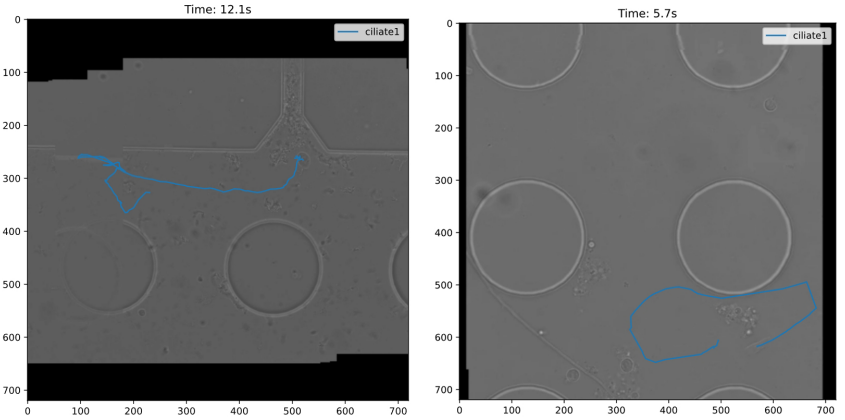
To analyse the trace features of each class, such as shapes or directions, we plot traces on the last frame of each video and select some typical figures in Figure 5.20.



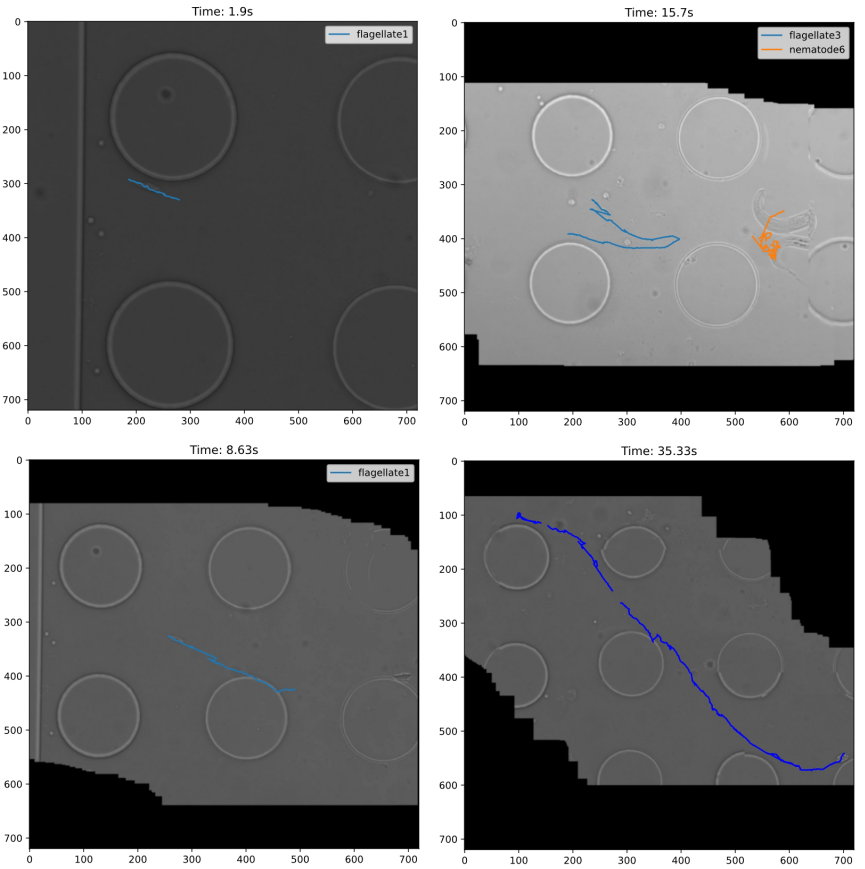
(a) Traces of amoebas



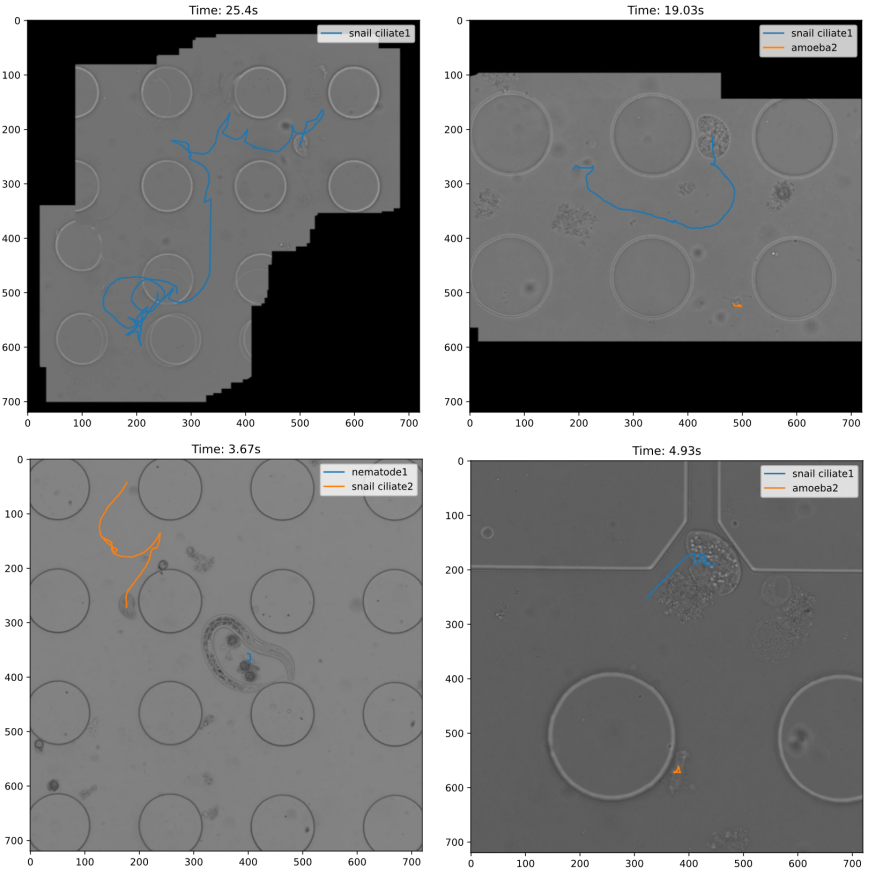
(b) Traces of apusomonases



(c) Traces of ciliates



(d) Traces of flagellates



(e) Traces of colpodea ciliates

Figure 5.20 The examples of traces of 5 classes shown on the original videos. The scale of the axes is pixels. The figure in the upper left corner of 5.20(e) shows that the colpodea ciliate moves into a pillar. This is caused by the failure of our compensated algorithm: the position of this pillar is wrongly patched due to the fast camera movement.

5.3 Discussion

Model

From Table 5.1 we can see that even though the pre-trained model is trained on COCO datasets, which is totally different from our custom dataset, models with the filename *.pt* still improve the performance of our model compared with training from scratch. What's more, instead of performing better with large-scale mod-

els as we imagined before, our dataset works well on a smaller pre-trained model: *YOLOv8s.pt*, which we assume is caused by overfitting. RT-DETR also has very good performance, but it is more computationally costly than YOLO.

The modification of the backbone also seems to be not very efficient as shown in Table 5.2. This modification forces the model to be trained from scratch, but as we do not have a large enough dataset, it is apparent that it cannot have a better performance compared with a pre-trained model. However, from Table 5.2 we can see the modification of the loss function does not really work as well, this also reduces the accuracy of the model. We think this may caused by the structure of YOLO. YOLO has a loss function designed to handle positive and negative samples, making it difficult for a general loss function algorithm to outperform.

For the two-step YOLO model, the classification model is almost 100% correct on the test dataset, and the object detection model has a slightly better performance than the one-step YOLO model. This implies that the two-step model is more accurate in recognizing the protists, and has a slightly better ability to find the position of the protists in the videos. In practice, this indeed works. The two-step model is more easier to find out the positions of protists in new videos with different settings, and more stable class labels due to both perfect classification accuracy and manual algorithm.

Data Analysis

Even though we do not have enough data, Figure 5.13 to Figure **??** still imply that if the concentration of the nanoplastics is high enough (for example $10mg/L$ in our work), the speed of the protists will be slower than control and low concentration environments. The fastest class among these 5 classes is ciliate, while the slowest class is amoeba. From Figure 5.19, we can see that the speed does not change much for amoeba and apusomonas, while the other 3 classes change their speed irregularly, especially for ciliate.

We can find some interesting features from the traces shown in Figure 5.20. Amoebas prefer to move along an edge, such as the edge of a pillar or soil chip. They are also the only class that we find "like" to move with nanoplastics in their body. Many videos are recording the amoebas moving with nanoplastics or trying to pull nanoplastics somewhere else, even though this makes them move slower. Apusomonas are quite small and move slowly, and use their flagellum to sense the objects in the way. In the two videos we have in our dataset, they both occur near the nanoplastics. The traces of ciliates show that they are quite "energetic" compared with the other classes. They have irregular and long movement traces in a short time. The same for the colpodea ciliates, they also move fast without any regularity. Both of these two classes also seem to be more likely to get stuck in the corner of the soil chips. Flagellates have a clear pattern of movement: they usually move smoothly in a straight line until they find some obstacles before them.

Problems. However, both speeds and traces are based on a premise: stable bounding boxes. We use the Euclidean distance between every adjacent bounding box's center to represent the speed. Therefore, to have a precise Euclidean distance, the location and size of the bounding boxes should be accurate. However, even though our model has high accuracy in classification, it is hard to ensure the size of the bounding box is appropriate, especially for flagellate due to its flagellum. Depending on the quality of the captured videos, sometimes it is really difficult for our model to detect the flagellum and only their "head" is detected. Therefore, the center coordinates will vary a lot even though the flagellate does not move much. There is also another problem for amoebas: as amoebas move by extending and retracting pseudopods [Singleton, Sainsbury, et al., 2001], the location of the center coordinates sometimes cannot represent its moving speed or direction. For example, if an amoeba extends part of itself but is still within the previous bounding box, the center coordinate will not change but it is indeed moving. Therefore, other methods should be applied for amoebas for more precise calculation, such as calculating the center of the segmentation instead.

The model performance causes another problem: the tracking ID for one object changes sometimes, especially for the compensated videos, which leads to discontinuous trace figures and speed distribution that disturbs analysis. In general, the YOLO model assigns one tracking ID for the same object, and we use this tracking ID to differentiate objects and analyze their features. However, our model sometimes assigns different tracking IDs for one object, so we have to modify the coordinates manually for better analysis.

6

Conclusion and Future Directions

6.1 Research Aims

In this thesis, we successfully build and test models to detect and track different species of protists with decent performance and accuracy. We then record the data and analyze their behaviour and properties under different treatment conditions. We find indications on that nanoplastics have some effect on protists, specifically, high-concentration nanoplastics seems to cause the protists to move slower than usual. We also find that different protists have disparate moving patterns, specifically, compared to other classes of protists, amoeba tend to move relatively slowly and ciliate tend to move relatively fast.

However, those results are based on too little data than we wanted, so they might not be highly reliable. The speed estimation, traces, and tracking of camera movement are still imperfect and can be refined.

We successfully fulfil the aims of this project, which demonstrate the feasibility of leveraging the power of AI and deep learning to help scientific research.

6.2 Future Directions

Our model still lacks decent generalization ability for unseen data without fine-tuning, thus not suitable for deployment as a finished tool. We further suggest some research directions to enhance practical implications.

Acquiring More Data

Much more high-quality data of different classes of protists need to be collected. Deep learning techniques highly rely on huge amounts of data to achieve high performance. To train a robust model to be deployed for laboratory research, sufficient data is necessary.

As the collection of dedicated large-scale micro-organism datasets with good quality requires a lot of time and effort, there is also the possibility of performing data augmentation by generating synthetic data with generative models. With more powerful generative models emerging in recent years, we can generate more synthetic images with higher quality to overcome the lack of datasets.

However, synthetic data can only help the classification and detection task. It can not be used to improve the speed estimation or analysis of impact of nanoplastics on microorganisms.

Learning Strategy

One of the reasons why large-scale datasets are hard to acquire is that manual annotation is expensive. A huge amount of unlabelled data can be relatively easy and cheap to acquire. A large research effort has been dedicated to methods to learn from unlabeled data. The fields of semi-supervised learning and self-supervised learning have demonstrated promising results. With the help of advancements in those fields, one could leverage a large amount of unlabelled data to learn visual representations and get a higher-level understanding of the data to improve performance.

Models advancements

To address the generalization problem, one other possible approach is to use more advanced models than conventional CNNs, which have a stronger ability to extract features. ViT-based models have shown promising results in various computer vision tasks, often achieving competitive or superior performance compared to CNNs. Those ViT-based models are believed to exceed the previous state-of-the-art model by a large margin. With more powerful and advanced models introduced, we can expect them to have stronger performance and generalization capacity.

Bibliography

Arjovsky, M., S. Chintala, and L. Bottou (2017). "Wasserstein generative adversarial networks". In: *International conference on machine learning*. PMLR, pp. 214–223.

Carion, N., F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko (2020a). "End-to-end object detection with transformers". In: *European conference on computer vision*. Springer, pp. 213–229.

Carion, N., F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko (2020b). "End-to-end object detection with transformers". In: *European conference on computer vision*. Springer, pp. 213–229.

Dai, J., H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei (2017). *Deformable convolutional networks*. arXiv: 1703.06211 [cs.CV].

Dalal, N. and B. Triggs (2005). "Histograms of oriented gradients for human detection". In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. Ieee, pp. 886–893.

Devan, K. S., P. Walther, J. von Einem, T. Ropinski, H. A. Kestler, and C. Read (2019). "Detection of herpesvirus capsids in transmission electron microscopy images using transfer learning". *Histochemistry and cell biology* **151**, pp. 101–114.

Dwyer, B., J. Nelson, T. Hansen, et al. (2024). *Roboflow (version 1.0)*. Version 1.0. Computer vision. URL: https://roboflow.com.

Felzenszwalb, P. F., R. B. Girshick, D. McAllester, and D. Ramanan (2009). "Object detection with discriminatively trained part-based models". *IEEE transactions on pattern analysis and machine intelligence* **32**:9, pp. 1627–1645.

Felzenszwalb, P. F. and D. P. Huttenlocher (2004). "Efficient graph-based image segmentation". *International journal of computer vision* **59**, pp. 167–181.

Geisen, S., E. A. Mitchell, S. Adl, M. Bonkowski, M. Dunthorn, F. Ekelund, L. D. Fernández, A. Jousset, V. Krashevska, D. Singer, et al. (2018a). "Soil protists: a fertile frontier in soil biology research". *FEMS Microbiology Reviews* **42**:3, pp. 293–323.

Geisen, S., E. A. Mitchell, S. Adl, M. Bonkowski, M. Dunthorn, F. Ekelund, L. D. Fernández, A. Jousset, V. Krashevska, D. Singer, et al. (2018b). "Soil protists: a fertile frontier in soil biology research". *FEMS Microbiology Reviews* **42**:3, pp. 293–323.

Girshick, R. (2015). "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

Girshick, R., J. Donahue, T. Darrell, and J. Malik (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative adversarial nets". *Advances in neural information processing systems* **27**.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2020). "Generative adversarial networks". *Communications of the ACM* **63**:11, pp. 139–144.

Gulrajani, I., F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville (2017). "Improved training of wasserstein gans". *Advances in neural information processing systems* **30**.

Hawksworth, D. L. (2009). *Protist diversity and geographical distribution*. Springer Netherlands.

He, K., G. Gkioxari, P. Dollár, and R. Girshick (2017). "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969.

Heideman, M., D. Johnson, and C. Burrus (1984). "Gauss and the history of the fast fourier transform". *IEEE Assp Magazine* **1**:4, pp. 14–21.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov (2012). "Improving neural networks by preventing co-adaptation of feature detectors". *arXiv preprint arXiv:1207.0580*.

Hossain, M. S., J. M. Betts, and A. P. Paplinski (2021). "Dual focal loss to address class imbalance in semantic segmentation". *Neurocomputing* **462**, pp. 69–87.

Huber, P. J. (1992). "Robust estimation of a location parameter". In: *Breakthroughs in statistics: Methodology and distribution*. Springer, pp. 492–518.

Ioffe, S. and C. Szegedy (2015). "Batch normalization: accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr, pp. 448–456.

Jiang, P., D. Ergu, F. Liu, Y. Cai, and B. Ma (2022). "A review of yolo algorithm developments". *Procedia computer science* **199**, pp. 1066–1073.

Jocher, G., A. Chaurasia, and J. Qiu (2023). *Ultralytics YOLO*. Version 8.0.0. URL: `https://github.com/ultralytics/ultralytics`.

Kingma, D. P. and J. Ba (2014). "Adam: a method for stochastic optimization". *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks". *Advances in neural information processing systems* **25**.

Lin, T.-Y., P. Goyal, R. Girshick, K. He, and P. Dollár (2017). "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988.

Lin, T., M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick (2014). "Microsoft COCO: common objects in context". *CoRR* **abs/1405.0312**. arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

Liu, W., D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg (2016). "Ssd: single shot multibox detector". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, pp. 21–37.

Mafla-Endara, P. M. (2023). *Encounters at the microscale: Unraveling soil microbial interactions with nanoplastics*. PhD thesis. Lund University.

Radford, A., L. Metz, and S. Chintala (2016). *Unsupervised representation learning with deep convolutional generative adversarial networks*. arXiv: 1511.06434 [cs.LG].

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (2016). "You only look once: unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Ren, S., K. He, R. Girshick, and J. Sun (2015). "Faster r-cnn: towards real-time object detection with region proposal networks". *Advances in neural information processing systems* **28**.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors". *nature* **323**:6088, pp. 533–536.

Sharma, P. (2023). *Understanding transfer learning for deep learning*. URL: https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/.

Singleton, P., D. Sainsbury, et al. (2001). *Dictionary of microbiology and molecular biology*. Wiley.

Springenberg, J. T., A. Dosovitskiy, T. Brox, and M. Riedmiller (2014). "Striving for simplicity: the all convolutional net". *arXiv preprint arXiv:1412.6806*.

Stanley, C. E., G. Grossmann, X. C. i Solvas, and A. J. deMello (2016). "Soil-on-a-chip: microfluidic platforms for environmental organismal studies". *Lab on a Chip* **16**:2, pp. 228–241.

Tong, Z., Y. Chen, Z. Xu, and R. Yu (2023). "Wise-iou: bounding box regression loss with dynamic focusing mechanism". *arXiv preprint arXiv:2301.10051*.

Uijlings, J. R., K. E. Van De Sande, T. Gevers, and A. W. Smeulders (2013). "Selective search for object recognition". *International journal of computer vision* **104**, pp. 154–171.

Van Loan, C. (1992). *Computational frameworks for the fast Fourier transform*. SIAM.

Wikipedia contributors (2024a). *Fast fourier transform — Wikipedia, the free encyclopedia*. https://en.wikipedia.org/w/index.php?title=Fast_Fourier_transform&oldid=1221693816. [Online; accessed 14-May-2024].

Wikipedia contributors (2024b). *Plastic — Wikipedia, the free encyclopedia*. [Online; accessed 1-May-2024]. URL: https://en.wikipedia.org/w/index.php?title=Plastic&oldid=1218609708.

Wu, C., Y. Chao, L. Shu, and R. Qiu (2022). "Interactions between soil protists and pollutants: an unsolved puzzle". *Journal of Hazardous Materials* **429**, p. 128297.

Zhai, X., Z. Huang, T. Li, H. Liu, and S. Wang (2023). "Yolo-drone: an optimized yolov8 network for tiny uav object detection". *Electronics* **12**:17, p. 3664.

Zhao, Y., W. Lv, S. Xu, J. Wei, G. Wang, Q. Dang, Y. Liu, and J. Chen (2023). "Detrs beat yolos on real-time object detection". *arXiv preprint arXiv:2304.08069*.

Zhao, Z.-Q., P. Zheng, S.-t. Xu, and X. Wu (2019). "Object detection with deep learning: a review". *IEEE transactions on neural networks and learning systems* **30**:11, pp. 3212–3232.

Zhuang, F., Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He (2021). "A comprehensive survey on transfer learning". *Proceedings of the IEEE* **109**:1, pp. 43–76. DOI: 10.1109/JPROC.2020.3004555.

A

YOLO Model Architectures with Different Scales

Table A.1 The architectures of different scales of YOLO models.

Model Scales	[depth, width, max_channels]	Layers	Gradients	FLOPs	Params
n	[0.33, 0.25, 1024]	225	3157184	8.9	3157200
s	[0.33, 0.50, 1024]	225	11166544	28.8	11166560
m	[0.67, 0.75, 768]	298	25902624	79.3	25902640
l	[1.00, 1.00, 512]	365	43691504	165.7	43691520