UNIVERSITÀ DI PISA

Dipartimento di Informatica

Corso di Laurea in Data Science and Business Informatics

A Deep Learning System for Detecting Microplastics Particles

Relatori:

Prof. Davide Bacciu

Prof. Modesto Castrillon Santana

Prof. Jose Javier Lorenzo Navarro

Presentata da:

Ema Ilic

Anno Accademico 2021/2022

Contents

Acknowledgements

There is no way I would have walked this path if it were not for the wonderful people and institutions who made the journey possible.

I would first of all like to thank my family for their support. Both of my parents contributed in their own unique way, shaping who I am today and supporting me throughout these years. Their help was immense and absolutely indispensable in this process. I would like to thank my grandparents, for their unconditional love, support and courage. I'd also like to thank Sabina Lenac, Milka Juranic, Nada Ilic, Sabina Lenac, Ivana Juranic Ilic, Sanjin Ilic, Robert Beletic, Marko Ilic, Dominik Ilic and Jan Ilic in no specific order for being the best and most supportive family one could ever wish for.

I also want to give thanks to my amazing friends, altruistic colleagues and kind acquaintances who have helped me throughout the years in infinite ways. My friends' loyalty, patience, and contribution has been fundamental to the achievement of this degree. And if it weren't for the kindness of colleagues and acquaintances, this journey would have been quite different.

I'd like to thank the professors who invested their time and effort in helping me create this work, namely professor Davide Bacciu from the University of Pisa, and professors Modesto Castrillon-Santana and Jose Javier Lorenzo Navarro from the University of Las Palmas de Gran Canaria. Their devotion to their vocation as well as to individual students and apprentices is something that greatly inspired an image of my future self in the professional world that I'm now embarking on.

Moreover, I would like to give thanks to the management, professors, and the staff of the University of Pisa for believing in me throughout the years. I'm deeply grateful for the support I was provided.

Even though I can not cite here each and every person and institution that helped me along this path, I'd just like to collectively say thanks for everything. I feel deeply humbled for the privilege of higher education and I vow to use it for further advancement of this world.

Hopefully I can someday return to the community a tiny speck of what the community has given me.

Abstract

Classification and quantification of microplastics particles is a crucial task in tracking plastics contamination and understanding its impact on the environment. However, it has traditionally been a tedious and laborious task when done manually, or it required special equipment when automatized, thus raising the need for a faster and more affordable process.

Previous research on automatized microplastics classification using only digital camera images relied on instance segmentation, with the premise that the shape of the particle is a cue on its origin. This might be true while classifying only three types of microplastics: lines, pellets and fragments, which differentiate significantly in their shape. However, when also considering organic particles and tar, which are oftentimes a byproduct of microplastics collection and display shapes similar to microplastics particles, the segmentation based approaches face a significant drop in performance.

This work introduces a Deep Learning approach for counting and classifying microplastics particles from an image taken with a digital camera or a mobile phone with resolution of 16M pixels or higher. The suggested approach comprises two steps: image processing using the Mosaic with Overlap technique, and object detection using YOLOv5 convolutional neural network. Five experiments are done, one concerning the image processing step, and four concerning data augmentation and transfer learning.

The state of the art system for automatic classification and quantification of microplastics particles not relying on special equipment was compared to the model devised in this work in terms of precision, recall and F1 metric, and the results were extremely satisfactory: YOLOv5-based model shows 303% higher Precision value, 490% increase in Recall, and 391% higher F1 value for all classes. Thus, this is the very first deep learning-based system which can suceessfully classify the three microplastics classes, as well as tar and organic particles with high Precision, Recall and F1 rates (>98%).

Chapter 1

Introduction

Artificial intelligence has made significant progress in the recent years in terms of object detection. Learning approaches that use Deep Learning models have surpassed traditional Computer Vision techniques for object detection based on geometrical features, pixel attributes, or statistical metrics. These methods typically deliver improved accuracy, model stability, and task generalization while needing less hardware in the process. As a result, many scientific and industrial processes can benefit from using artificial intelligence approaches to increase efficiency and cut costs. An object detection model could be integrated into the work biologists traditionally did manually, which is time consuming and tedious, with the goal of improving the effectiveness of their work, both in terms of accuracy as well as speed.

The purpose of this research is to explore whether a state of the art YOLOv5 model improves the performance of the previous methods used in the aforementioned papers in automatically counting and classifying microplastics particles without special equipment. This work builds upon [Lorenzo-Navarro et al., 2021] and
[Lorenzo-Navarro et al., 2020]. Whereas the former deployed hybrid approach between a bounding-box based approach and an instance segmentation approach,
[Lorenzo-Navarro et al., 2020] used traditional classifiers such as KNN, Random Forest, SVM in combination with special equipment (a high resolution scanner).

In this work, Deep Learning background is discussed in in chapter 2. Artificial and convolutional neural networks are presented and explained, keeping in mind the most important features of these models. Following that, overview of automatized microplastics counting and classification is presented. The problem of plastics contamination and its impact on the environment is presented, with stress on the marine environment. The concept of 'microplastics' is introduced and defined, and the lenghty process of plastics desintegration is addressed. Various types of microplastics are introduced, and so is their origin. This section also describes methods traditionally used for microplastics counting and classification and addresses their shortcomings, thus highlighting the need for automatizing the process. The rest of the section discusses the most important achievements and papers in the field of automatic microplastics classification.

Next, in chapter 3 a high level architecture of the approach is presented, and the selected computer vision model described. YOLOv5's structure is explained in detail, and so is its performance, and it is compared to its predecessors. Software and Hardware tools used in this work are presented in the same chapter.

In chapter 4, all technical details behind developing the system are elaborated. Namely, the process for data pre-processing and data post-processing is described, and so are all the nits and grits of the dataset, both the images as well as the annotations. The reasoning behind various dataset modifications is explained through sections on dataset problems, and the process for obtaining the new dataset is introduced in section 4.5. Final dataset is presented and so is the training, validation and test split which was necessary in order to train and test our model. Finally, in section 4.7 the post-processing of our data is explained and elaborated. This section also showcases several problems which were faced during the end phase, such as duplicate detections and repositioning bounding boxes on the original image.

Additionally, chapter 5 presents the settings which are going to be experimented with, namely image and dataset size, data augmentation tecnhiques, and transfer learning.

The penultimate chapter, chapter 6 discusses results achieved with this work. The results

of the experiment regarding dataset and image size are presented, and the following four experiments are elaborated with the dataset selected in Experiment 1. The following four experiments concert different levels of transfer learning and data augmentation, or a lack thereof. Next, metrics described in section 1.1 are compared, and the post-processing results of the best performing experiment are presented. Finally, state of the art model is trained and tested on the same dataset as was the YOLOv5 model, and the results are presented. YOLOv5 results are discussed and compared to the state of the art model, and the resource limitations are discussed.

Finally, chapter 7 states the conclusions of this work, highlights the best performing system and instructs us on the possible further research directions.

1.1 Metrics

Before delving into the work, it is important to first explain the metrics which are going to be mentioned and discussed throughout the course of this thesis.

Precision, Recall and F1. Most metrics used in evaluating object detection systems are based on Precision and Recall, and their formulas can be found on Equation 1.1 and Equation 1.2, respectively. Precision measures how accurate are model's predictions. i.e. the percentage of predictions that are correct. It answers the question: What proportion of positive identifications was actually correct? Recall on the other hand answers the question: What proportion of actual positives was identified correctly?

$$Precision = \frac{TP}{TP + FP}. \tag{1.1}$$

$$Recall = \frac{TP}{TP + FN}. \tag{1.2}$$

In addition to these metrics, one of the most common plots present in this thesis is the Precision-Recall (PR) curve. It shows the trade-off between precision and recall

for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall). In order to have a better understanding of the outcome and the differences between the experiments, other plots have also been included in the analysis. For example, the confusion matrix highlights the percentages of true positives and false positives, and the FN background issue. F1 metric is also useful since it is a harmonic mean between the precision and recall, its formula can be observed on Equation 1.3.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}. \tag{1.3}$$

IoU. Another metric which can be useful in evaluating the model's performance is the Intersection over Union (IoU). IoU measures the overlap between 2 bounding boxes. We use it to measure how much our predicted boundary overlaps with the ground truth (the real object boundary, i.e. it's most well-fitting bounding box). In some datasets, we predefine an IoU threshold (say 0.5) in classifying whether the prediction is a true positive or a false positive.

$$IoU = \frac{area_i}{area_g + area_p - area_i}. \tag{1.4}$$

Where area_g refers to the area of the ground truth bounding box, area_p refers to the area of the predicted bounding box, and area_i refers to the area of intersection.

Average Precision (AP) is the precision averaged across all recall values between 0 and 1 at various IoU thresholds. By interpolating across all points, AP can be interpreted as the area under the curve (AUC) of the precision-recall curve. As AP curves are often zigzag curves going up and down, comparing different curves (different detectors) in

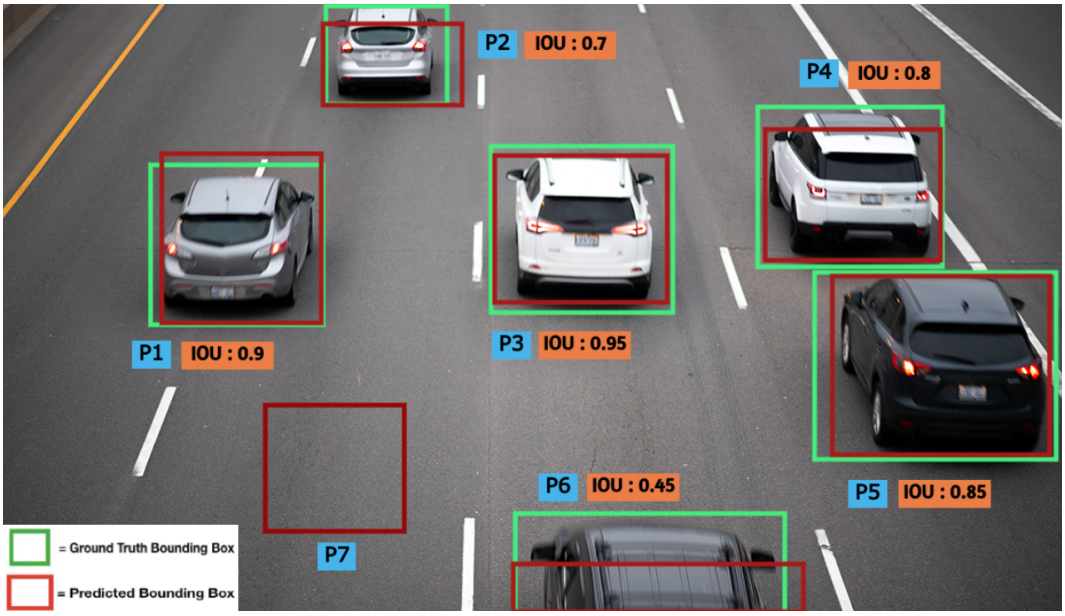

Figure 1.1: An example of IoU calculated on ground truth vs. predicted bounding boxes. Source: [v7,]

the same plot usually is not an easy task - because the curves tend to cross each other frequently. That's why Average Precision (AP), a numerical metric, can also help us compare different detectors. In practice AP is the precision averaged across all recall values between 0 and 1. In Glenn Jocher's implementation of YOLOv5 in Ultralytics library [ult,], the metrics were created with the help of the following repository: [Obj,].

$$\text{AP} = \sum_{n=0} (r_{n+1} - r_n) \rho_{interp}(r_{n+1}). \tag{1.5}$$

Where

$$\rho_{interp}(r_{n+1}) = \max_{\tilde{r}:\tilde{r} \geq \tilde{r}_{n+1}} \rho(\tilde{r}). \tag{1.6}$$

where $\rho(\tilde{r})$ is the measured precision at recall $\tilde{r}$. Thus, the AP is obtained by interpolating the precision at each level (all points), taking the maximum precision whose recall value is greater or equal than $r+1$. This way we calculate the estimated area under the curve.

Mean Average Precision (mAP). The most common metric for evaluating the object detection tasks is mAP (mean Average Precision), so the results of different experiments have been evaluated in terms of this metric, both for IoU = 0.5 (mAP@0.5) and IoU = [0.5 : 0.05 : 0.95] (mAP@[0.5:0.95]). To clarify, AP_{IoU} is the Area Under the Precision Recall curve per each object category, considering a certain IoU (Intersection over Union, Equation 4.4) threshold. Thus, the mAP@0.5 is the mean Average Precision in which the predicted bounding boxes are considered TP (True Positives) if their IoU with the ground-truth is greater than 50%. According to the same reasoning, the mAP@[0.5:0.95], is the mean Average Precision calculated starting from IoU=0.5, with steps of 0.05, to IoU=0.95. Therefore, there are ten computations of AP at ten different IoU threshold values, and a final mean is done to provide a single number.

Complete Intersection over Union (CIoU). Another important metric used in estimating the models is the CIoU Regression Loss (Complete Intersection over Reunion). It incorporates all geometric factors: overlapping area, distance between the predicted box and target box, and aspect ratio of the bounding box. This metric is used to calculate the regression loss and it tells us about how the trained model fits new data. The formula is cited in Equation 1.7.

$$Loss_{CIoU} = 1 - IoU + \frac{\rho^2(p, p^{gt})}{c^2} + \alpha v. \tag{1.7}$$

FLOPs. Floating point operations are often used to describe how many operations are required to run a single instance of a given model. Floating-point is a method of encoding real numbers within the limits of finite precision available on computers. Using floating-point encoding, extremely long numbers can be handled relatively easily. A floating-point number is expressed as a basic number or mantissa, an exponent, and a number base. The number base is usually ten but may also be 2. Floating-point operations require computers with floating-point registers. The computation of floating-point numbers is often required in scientific or real-time processing applications and

FLOPS is a common measure for any computer that runs these applications.

Time. Finally, several aspects of model validation have been measured to assure comparability in terms of time it takes for the model to detect an image in the test set. First of all, pre-processing average time has been measured. This is the time it takes to prepare the test dataset and the model for detection task. Inference time, on the other hand, is the average time it takes to detect and classify all the objects in a single image. Lastly, Non Maximum Suppression (NMS) time is the time to to select one entity (e.g. bounding box) out of many overlapping entities.

Chapter 2

Background

In this chapter we first explain the Deep Learning as a concept. Complementary to that concept, the Artificial Neural Networks are explained, followed by Convolutional Neural Networks, traditionally used for image classification. Next, we select the most prominent and relevant inventions and architectures in the field of computer vision and explain them, as they are going to be important for understanding YOLO models, which were finally chosen for this task, and described in section 3.2 and section 3.3. Lastly, this chapter ends with an overview of the literature of Automatic Microplastics Classification.

2.1 Introduction to Deep Learning

Before getting into the details of the work, it is important to first explain the current state of the art in the world of computer vision, also known as machine vision, and why deep learning has been chosen as the main tool to solve this problem.

As mentioned above, the field of computer vision is a wide one, and has a multitude of strategies to solve problems such as the one posed in this work. Among the most widely used tools today are deep learning and neural networks. Despite their recent popularity, these techniques were developed decades ago, in the the 1980s, though they were not implemented as commonly until a few years ago. Pioneering examples of these systems include

the backpropagation learning model by Rumelhart [Rumelhart et al., 1986] or the convolutional network model for detection of numbers elaborated by LeCun [Russakovsky et al., 2014]. Neural networks require a large computational capacity and large amounts of data to perform optimally. Unfortunately, given the high complexity of these systems, it was impossible to apply them to real problems. Nowadays, the situation has changed quite a bit. Technology has advanced enough to allow the use of these intelligent models. The increase in computer power and the availability of large amounts of data has allowed the development of these powerful techniques.

2.1.1 Artificial Neural Network

An artificial neural network (ANN) is a data-driven learning algorithm inspired by biological nervous system. Its origins are dated to 1943, to a study of mathematical representations of information processing in biological systems by [Mcculloch and Pitts, 1943]. The structure of a neural network is represented by a directed acyclic graph, embodied of several connected layers, each consisting of nodes (or neurons), as can be seen on Figure 2.1.

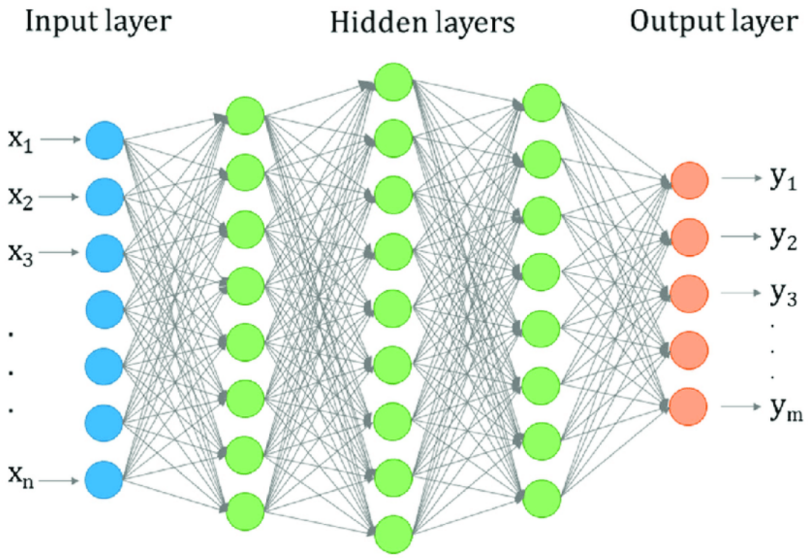

Figure 2.1: Basic scheme of a neural network. Source[de la Fuente Stranger et al., 2019]

Output of a node in each layer is weighted and fed to nodes of the subsequent layer.

Inputs of each node are being weighted and transformed by a function, specific for the node, called activation function. A neural network can be seen as a mapping, where input, fed to the network through the input layer, is mapped to the output layer. Output of each node is produced by the node's activation function σ that takes weighted inputs of the node as parameters transformed by a transfer function (see Figure 2.2). The transfer function creates a linear combination of weighted inputs in order to feed them to the activation function. To approximate complicated functions, nonlinear activations are often used. The most popular are:

Hyperbolic tangent:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{2.1}$$

this function works most effectively on inputs in range $(0;1)$, producing outputs in interval $(-1,1)$;

Sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{2.2}$$

one of the reasons the sigmoid function is broadly used is the fact, the sigmoid function is differentiable at every point;

ReLU:

$$f(x) = \max(0, x), \tag{2.3}$$

rectified linear unit's function is used with the purpose to increase non-linearity of the network. They benefit from their simplicity, resulting in faster training and performance improvements in particular cases, and therefore often used in deep neural networks and convolutional neural networks.

Softmax:

$$p\left(c_k \mid x\right) = \frac{e^{a_k}}{\sum_{i=1}^{m} e^{a_i}}, \tag{2.4}$$

the softmax activation function is usually used in the last network layer, converting an arbitrary real value to posterior probability of the class c_k in range $(0;1)$, where m corresponds the number of output nodes (classes) and a_k is the activation value of k-th node.

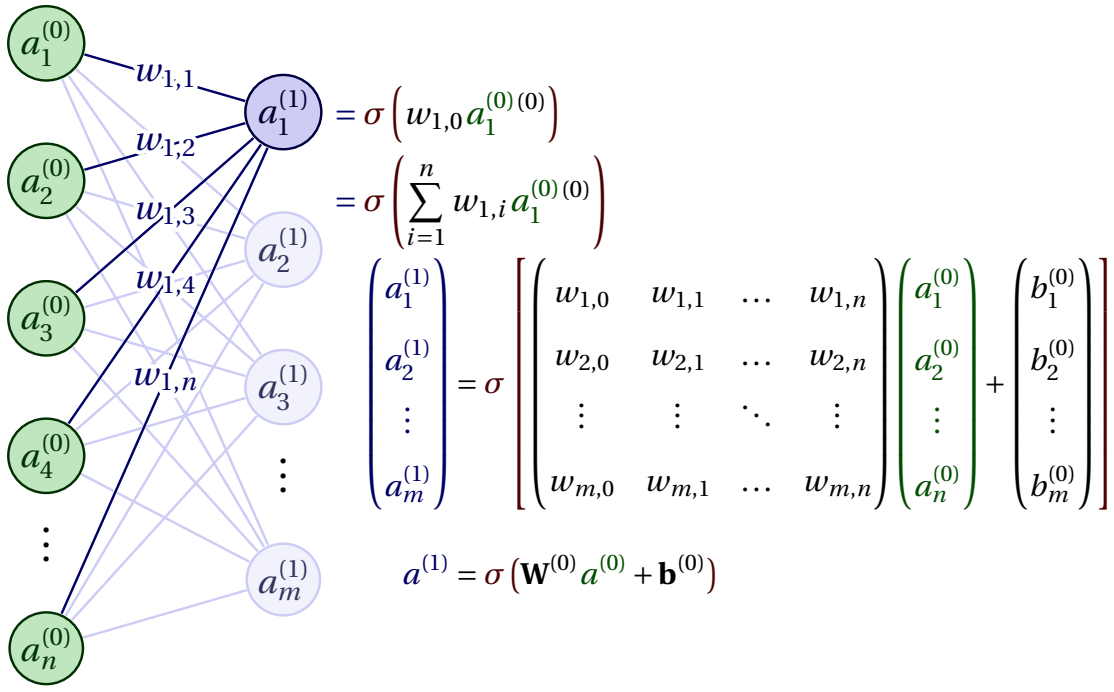

Figure 2.2: Activation function in one neuron and one layer in matrix notation

Training

Loss Function: To measure a precision of the network outcome, a loss (cost or objective) function is used. It expresses how much the prediction differs from expected value. The output of the loss function is a real value referred to as the cost or the penalty. An example of a loss function that outputs probabilities, thus often used in visual classification problems is the cross-entropy loss function:

$$L = -\sum_{i}^{m} y_i \log p_i, \tag{2.5}$$

where m is the number of possible classes (nodes) in the output layer, y the target vector and p the a posterior probability for each class predicted by the network. Evaluated derivatives of a loss function are used in the training phase.

Backpropagation: is a neural network training algorithm. For supervised learning, target classes are essential for error calculation. The error is afterwards backpropagated to every node in previous layers. This error e (Equation 2.6) is obtained as a gradient of the loss

function L with respect to each layer's weights w_{kj} given input of the node x and activation function ϕ.

$$e = \frac{\partial L}{\partial w_{kj}} = \frac{\partial L}{\partial a_j}\frac{\partial a_j}{\partial w_{kj}}, \quad \text{where} \quad a_j = \phi\left(\sum_{k=1}^{n} w_{kj}x_k\right). \tag{2.6}$$

Gradient computation demands application of the chain rule in order to compute partial derivative of the loss function L with respect to particular weight w_{kj}. Using the error, weights are updated by an optimization algorithm such as gradient descent.

Gradient descent: The most common function optimization algorithm used for neural networks is the gradient descent, a first order approximation algorithm that updates weights of the model. The algorithm approaches a local minimum in the direction of the negative gradient of the loss function with respect to the weights. The size of the step is called learning rate. It is a scalar in the range $(0;1)$, controlling magnitude of network's parameters (weights) change. To perform one update of the weights, the whole training set has to be used. For large training sets, this method might be computationally expensive. A more time efficient gradient descent based optimization method is the stochastic gradient descent or SGD (Equation 2.7). SGD needs only one observation (or subset of the training set) to update model parameters w. As the name suggests, at each weight update a random observation is used. Furthermore, SGD does not tend to end up stuck in a local minima such as ordinary gradient descent (also called batch gradient descent). A disadvantage of SGD is a slower convergence rate than convergence rate of batch gradient descent. Due to its stochasticity, a wrong choice of starting observations may cause algorithm to move further from global minima and make convergence problematic.

$$w^{(t+1)} = w^{(t)} - \eta\nabla_w L\left(w^{(t)}\right). \tag{2.7}$$

Weights w are being updated by the negative of the gradient of the loss function with respect to the weights. This change is limited by the learning rate η.

Momentum: Numerous improvements for gradient descent were proposed. One of the most frequently used enhancements is the momentum. Momentum helps to prevent from

convergence to a local minima and also speeds up the convergence process by preserving a fraction of previous weight adjustments. Previous weight adjustment is used in current update, multiplied by factor μ, the momentum, obtaining:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w L\left(w^{(t)}\right) + \mu w^{(t)}. \tag{2.8}$$

Dropout: Dropout is a regularization method presented in [48], which prevents network from overfitting by dropping out random nodes with their connections in each training iteration. This method can be applied to one or more layers. For each layer, a different probability may be used to skip a node. Optimal probability to skip a node was found to be 50% for a hidden layer, and 20% for an input layer. In each training iteration, a different thinned network is trained. Output from all the thinned networks can be approximated by using the whole network with activations scaled by probability the node is used in the training phase.

2.1.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are commonly used in the detection of objects and image processing [CNN,]. They are a specialized kind of deep neural network most commonly used to analyze images, or in general for processing data that has a grid-like topology. As can be deducted from the very name of the network, a mathematical operation called convolution is deployed here. In line with [Goodfellow et al., 2016], convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers, developed and deployed for the first time in 1980s. Currently, they are used mainly for image processing, classification, segmentation, but also for other types of data such as time series, which can be of use in in voice recognition.

It can be said that the CNNs are split up into two parts, each one with a specific objective. An illustration is provided at Figure 2.3.

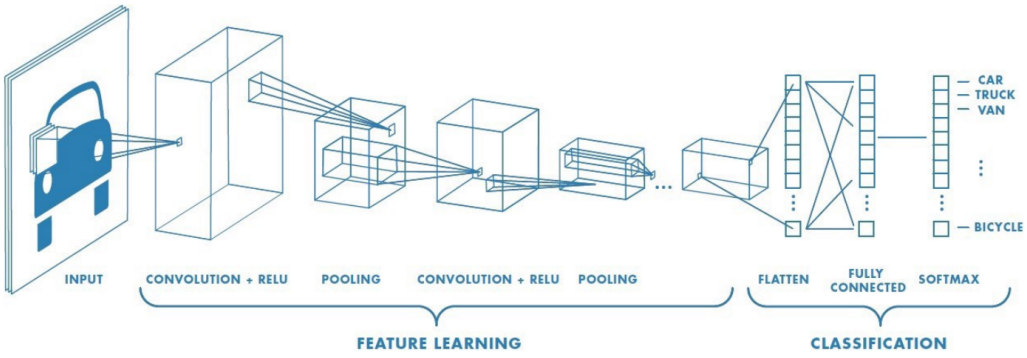

Figure 2.3: Layers of a Convolutional Neural Network. Source: [mor,]

The network receives an image and its objective is to analyze its features. This is known as feature learning. To achieve this, two types of layers are used: the convolutional layer and the pooling layer. The convolutional layer is in charge of extracting these features. Unlike fully connected networks, this layer would not be formed by neurons. The units that form it are called kernels or filters. These filters analyze the image on a pixel-by-pixel basis and extract different characteristics of the image, such as shadows, colors or edges. CNN's weights are designed to form a convolutional filter that is replicated over the whole visual field (Figure 2.4).

In mathematical notation: given two functions x and w the convolution operation is defined as:

$$y(t) = \int x(a)\, w(t-a)\, dt. \tag{2.9}$$

In the CNN terminology x is the input, y the output, also called feature map and w is the convolution kernel. CNN usually employs the discrete convolution, a discrete variant of the convolution operation defined as:

$$y(t) = \sum_{a=-\infty}^{+\infty} x(a)\, w(t-a). \tag{2.10}$$

and this can be easily extended to multiple dimensions:

$$\sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} x(i,j)\, w(m-i, n-j). \tag{2.11}$$

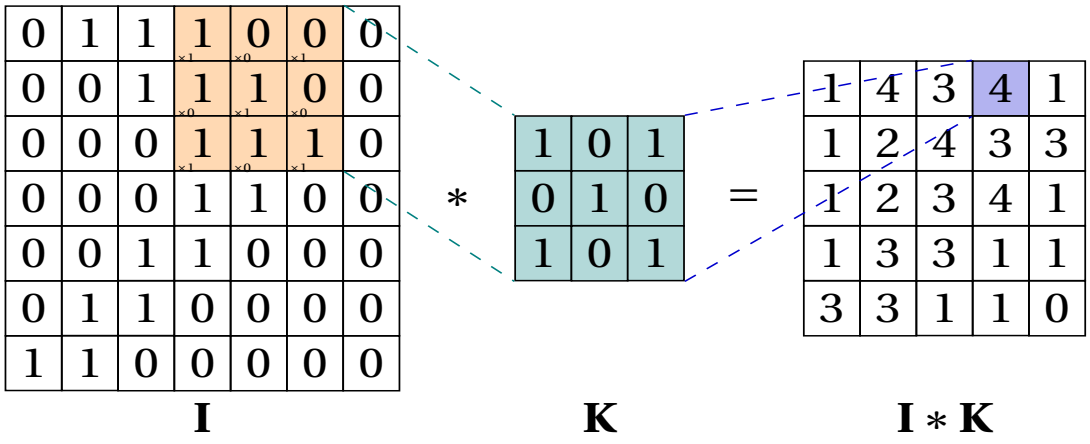

Figure 2.4: Convolutional layer (I), kernel (K) and the feature map or output of the convolution (I*K)

On the other hand, the pooling layer reduces the image size by applying different compression techniques. In this way, the computational capacity required for the filters of a layer to convolve is reduced, as shown on Figure 2.5.

In addition, as the image is compressed, its predominant features are accentuated, allowing successive layers to extract those characteristics more easily. This is why a convolutional layer is typically followed by a pooling layer. The output of the convolutional network would be a tensor formed by matrices containing a number of image features. A matrix is obtained in each convolutional layer. This tensor is called Feature Map. This result is commonly passed as input to a classifier. This is a fully connected conventional neural network. It is in charge of generating an output based on the features obtained in the first module, as shown in Figure 2.3.

Convolutional Neural Network Architectures

As previously mentioned, most of the CNN architectures follow the same structure: the early layers of the network extract local visual features, while later layers would combine them in order to form higher-order features. Thanks to the importance of CNNs in various types of problems, numerous tech companies have developed new architectures. LeNet-

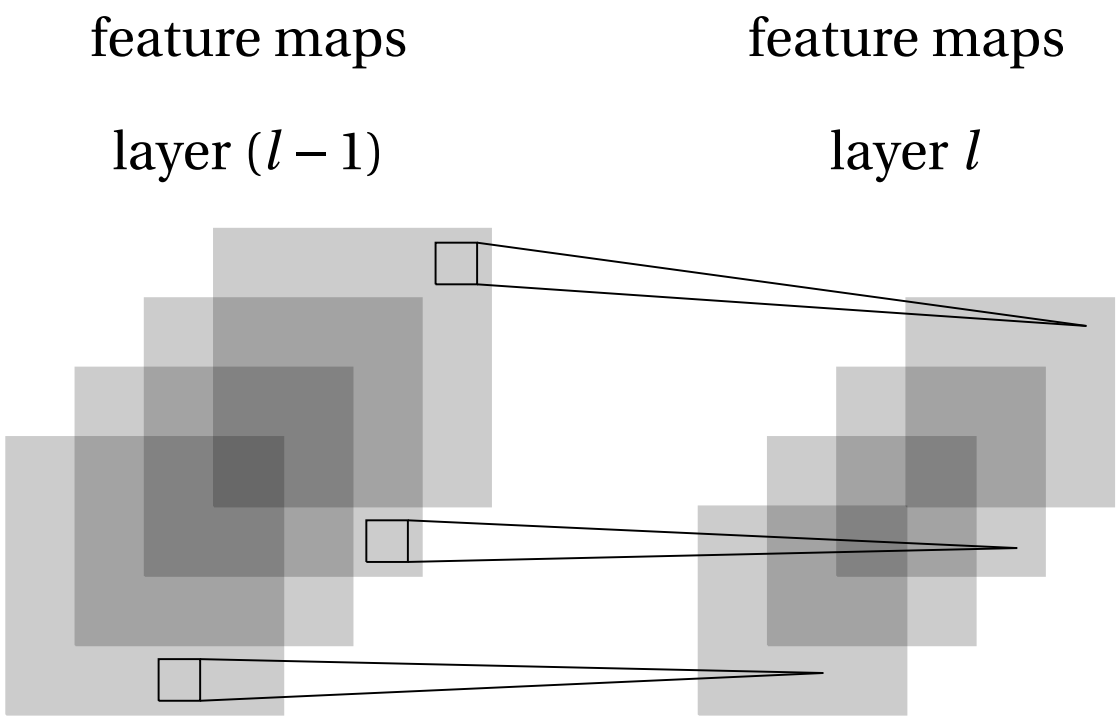

Figure 2.5: Illustration of a pooling and subsampling layer. If layer l is a pooling and subsampling layer and given $m_1^{(l-1)} = 4$ feature maps of the previous layer, all feature maps are pooled and subsampled individually. Each unit in one of the $m_1^{(l)} = 4$ output feature maps represents the average or the maximum within a fixed window of the corresponding feature map in layer $(l-1)$.

5 ([Rawat and Wang, 2017]) was developed in 1998 by Yann LeCun to identify handwritten digits and it defined the basic components of CNN that we know today. Indeed, the architecture AlexNet [Krizhevsky et al., 2012], that is developed in 2012, is similar to LeNet-5 but it is considerably larger and its performance outperformed other models available at that moment. In 2014 VGG-16 ([Simonyan and Zisserman, 2014]) network was introduced as a deeper variant of CNN. VGG networks use a lot of parameters, but more parameters do not always lead to better performance. In 2014, researchers at Google introduced the Inception network, which obtained good performance in both classification and detection tasks and that is different from previous architectures. For the sake of brevity, we are only going to tackle the most relevant architectures.

2.1.3 R-CNN

Regions with CNN features (R-CNN) ([Girshick et al., 2013]) is the first object detection to show that a CNN can lead to dramatically higher object detection performance. Unlike previous methods that tackle the localization problem by building a sliding-window detector, R-CNN solves the problem by operating within the 'recognition using regions' paradigm [Gu et al.,], which has been successful for both object detection and semantic segmentation. An example of detection result using R-CNN in shown in Figure 2.6.

Figure 2.6: Example of object detection with R-CNN

Step 1: Feature Extraction using CNN and object detection using SVM

The R-CNN finds out where bounding boxes are by proposing numerous possible boxes in the image and seeing if any of them actually correspond to an object. R-CNN creates these bounding boxes, or region proposals, using a process called selective search ([Uijlings et al., 2013]). At a high level, selective search looks at the image through windows of different sizes, and for each size tries to group together adjacent pixels by texture, color, or intensity to identify objects. Once the proposals are created, R-CNN warps the region to a standard square size and passes it through to a modified version of AlexNet [Krizhevsky et al., 2012] or VGG-16 [Simonyan and Zisserman, 2014]. At this point follows the fundamental features extraction operation: AlexNet or VGG-16 extract a fixed-length feature vector from each region proposal.

On the final layer of the CNN, R-CNN adds a Support Vector Machine ([Cristianini and Ricci, 2008]) that simply classifies whether a feature vector corresponds to an object, and if so what object.

Finally, In order to improve localization performance, the authors include a bounding-box regression step to learn corrections in the predicted bounding box location and size and generate the final result: tighter bounding box coordinates to fit the true dimensions of the object. So the inputs of this regression model are sub-regions of the image corresponding to objects while the outputs are new bounding box coordinates for the object itself in the sub-region. Figure 2.7 shows the high level architecture of R-CNN.

Step 2: Bounding Box Regression

The aim of this task is to learn a target transformation between our predicted proposal P and the target proposal G. The variables x, y, w, and h stand for the coordinates of the center (x, y) and the width w and height h of the proposal.

$$\begin{aligned} P^i &= (P^i_x, P^i_y, P^i_w, P^i_h) \\ G &= (G_x, G_y, G_w, G_h). \end{aligned} \tag{2.12}$$

The ground-truth transformations that need to be learnt are shown in Equation 2.13. The first two transformations specify a scale-invariant translation of the center of P — x and y, and the second two specify log space transformations of the width w and height h.

$$\begin{aligned} t_x &= (G_x - P_x)/P_w \\ t_y &= (G_y - P_y)/P_h \\ t_w &= log(G_w/P_w) \\ t_h &= log(G_h/P_h). \end{aligned} \tag{2.13}$$

$d_k(P)$ where k can belong to (x, y, h or w) is the predicted transformation. $\hat{G}$ signifies the corrected predicted box calculated using the original predicted box P and the predicted trans-

formation $d_k(P)$).

$$\begin{aligned} \hat{G}_x &= P_w d_x(P) + P_x \\ \hat{G}_y &= P_h d_y(P) + P_y \\ \hat{G_w} &= P_w exp(d_w(P)) \\ \hat{G}_h &= P_h exp(d_h(P)). \end{aligned} \tag{2.14}$$

The predicted transformation $d_k(P)$ is modeled as a linear function of the $pool_5$ features — ϕ_5. Hence, $d_k(P) = w_k^T \phi_5(P)$ where w_k is the vector of learnable model parameters. Note that ϕ_5 is dependent on the actual image features. w_k is learnt by optimizing the regularized least-squares objective function shown in the second line of Equation 2.15.

$$\begin{aligned} d_*(P) &= w_*^T \phi_5(P) \\ w_* &= \underset{\hat{w}_*}{\operatorname{argmin}} \sum_i^N (t_*^i - \hat{w}_*^T \phi_5(P^i))^2 + \lambda \| \hat{w}_* \|^2 \end{aligned} \tag{2.15}$$

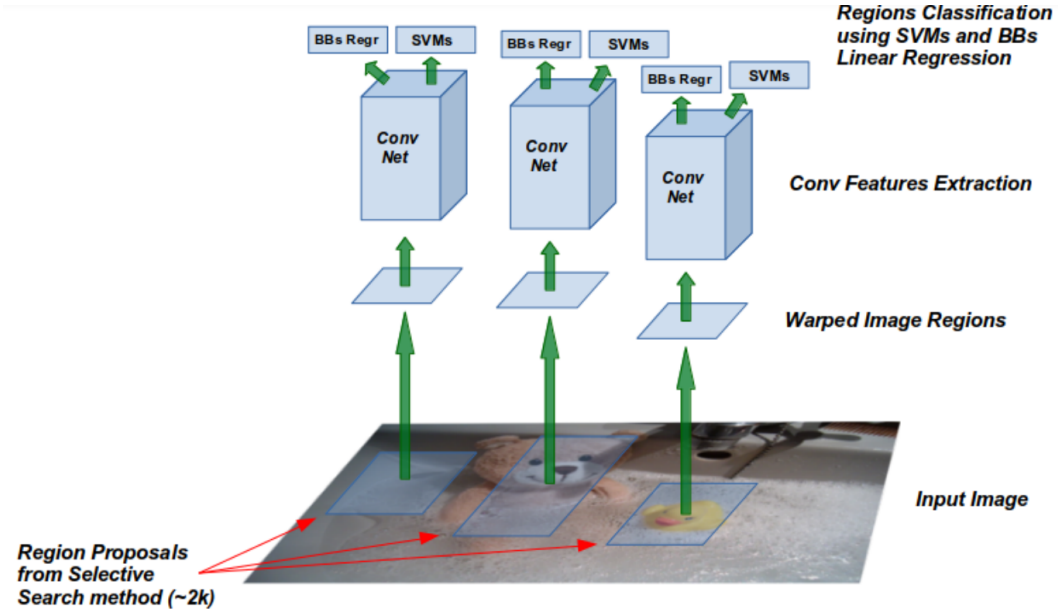

Figure 2.7: High Level Architecture of R-CNN

Finally, note the R-CNN shares some similarities with YOLO described in section 3.2, proposing bounding boxes and scoring them. However, R-CNN does not put spatial constraints when proposing potential bounding boxes. Furthermore, the entire detection pipeline

is more complex than the YOLO framework, that frames object detection as a single regression problem using just one CNN.

2.1.4 Fast R-CNN

While R-CNN works well, it is quite slow for a few simple reasons:

1. It requires a forward pass of the CNN (AlexNet or VGG-16) for every single region proposal for every single image (that's around 2000 forward passes per image).

2. It has to train three different models separately - the CNN to generate image features, the classifier that predicts the class, and the regression model to tighten the bounding boxes. This makes the pipeline extremely hard to train.

Both these problems are solved with Fast R-CNN [Girshick, 2015] exploiting two insights:

1. **RoI (Region of Interest) Pooling.** Using a neural-net layer known as RoIPool (Region of Interest Pooling), Fast R-CNN runs the CNN just once per image and then find a way to share that computation across the about 2000 proposals. What the RoI Pooling actually does is to take in input a fixed-size feature map obtained from a deep convolutional network and an N×5 matrix representing a list of regions of interest, where N is a number of RoIs (the first column represents the image index and the remaining four are the coordinates of the top left and bottom right corners of the region). For every region of interest from the input list, RoIPool takes a section of the input feature map that corresponds to it and scales it to some pre-defined size (e.g., 7×7). The scaling is done by dividing the region proposal into equal-sized sections (the number of which is the same as the dimension of the output), finding the largest value in each section, and copying these max values to the output buffer. The result is that from a list of rectangles with different sizes we can quickly get a list of corresponding fixed-length feature

vector from the feature map. An example is shown in Figure 2.8.

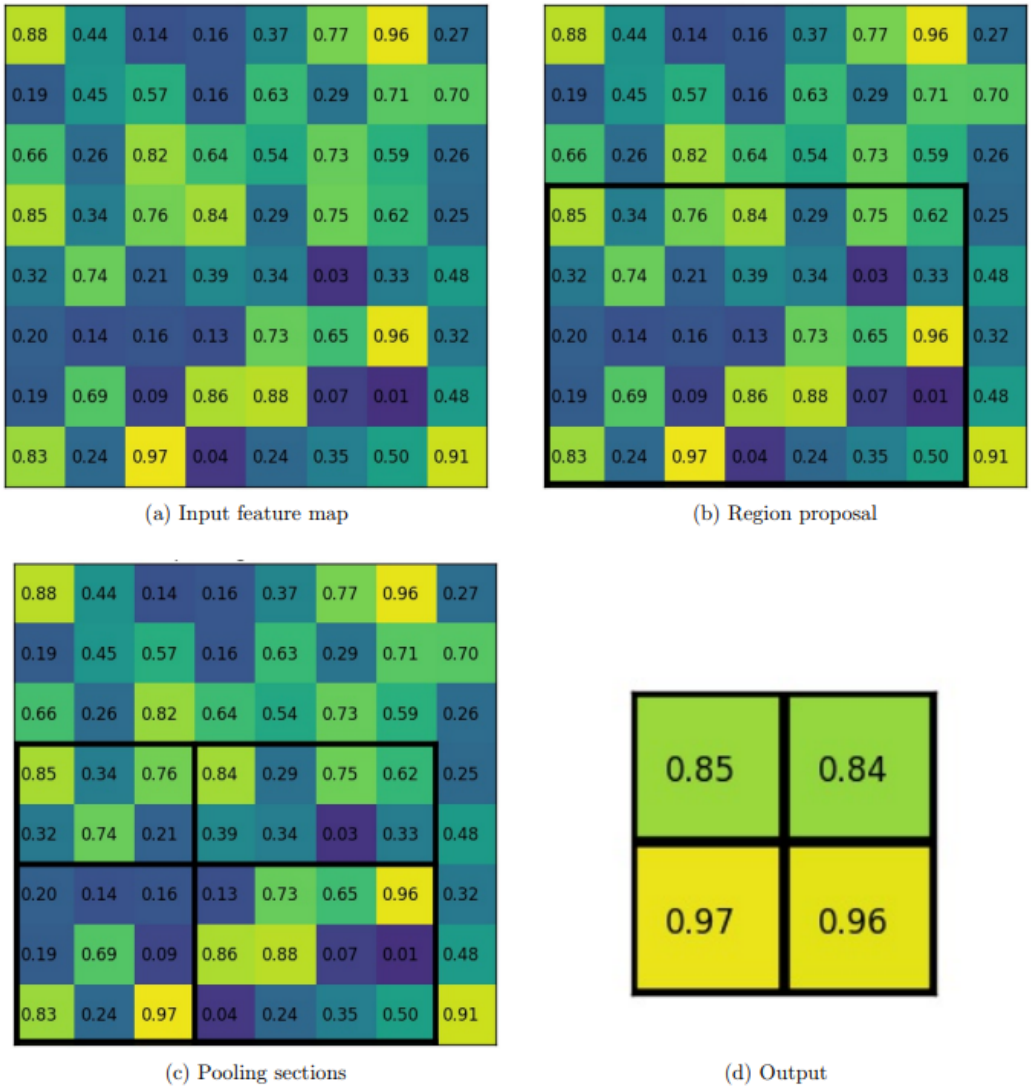

Figure 2.8: Example of RoI Pooling on a single 8×8 feature map, one region of interest (top left, bottom right coordinates: (0, 3), (7, 8)) and an output size of 2×2. Notice that the size of the region of interest doesn't have to be perfectly divisible by the number of pooling sections (in this case our RoI is 7×5 and we have 2×2 pooling sections).

2. **Combine all models into one network.** While in R-CNN we had different models to extract image features (CNN), classify (SVM), and tighten bounding boxes (regressor),

Fast R-CNN instead uses a single network to compute all three. Each feature vector is fed into a sequence of fully connected layers that finally branch into two sibling output layers: a softmax layer, replacing the SVM classifier, that produces probability estimates over K object classes, and another layer that outputs bounding box coordinates for each of the K object classes. Indeed, in this way, all the outputs needed come from one single network. Figure 2.9 shows the high level architecture of Fast R-CNN.

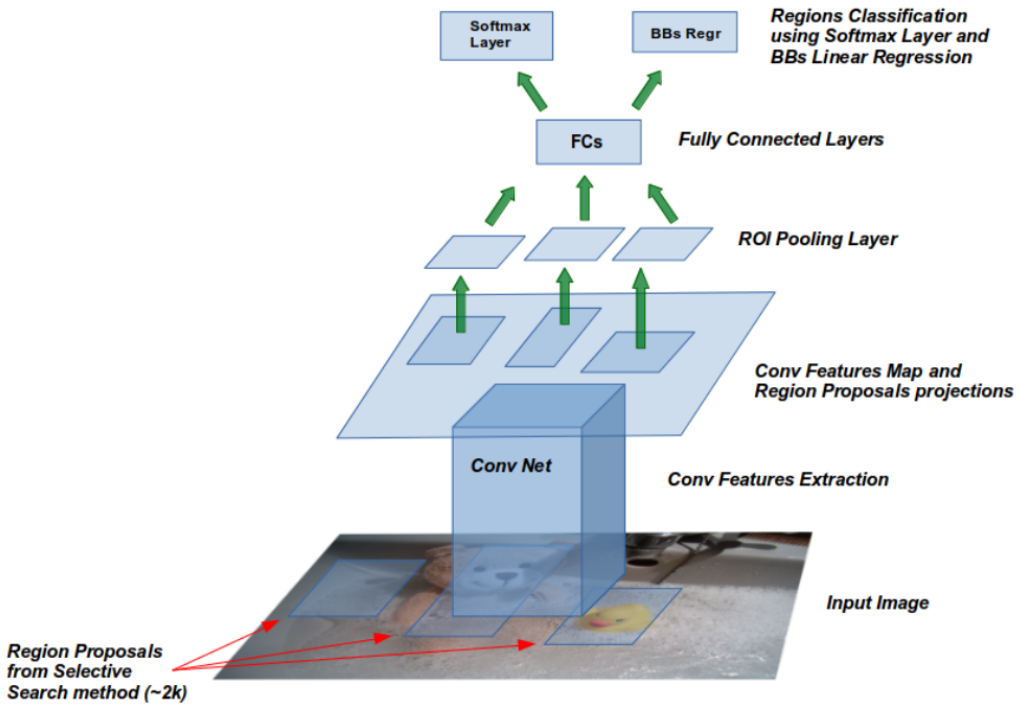

Figure 2.9: High level architecture of Fast R-CNN: CNN, classifier and bounding box regressor are combined into one, single network.

2.1.5 Faster R-CNN

The Faster R-CNN architecture [Ren et al., 2015] was an architecture released in 2015 that aims to go further and improve the detection speed of its predecessor, Fast R-CNN. To achieve this, it adds a Fully Convolutional Network on top of the features of the CNN creating what's known as the Region Proposal Network (RPN). It is a convolutional neural network whose objective is not only to generate the map of the characteristics of an image, but also to propose

regions in which the object is more likely to be found. In this way, the regions that are taken as input in the RoI pooling layer are no longer fixed, but are determined by a network. The classifier continues to take as input the same feature map generated by the convolutional network. Therefore, the improvement of this architecture over its predecessor is based primarily on how well the proposed regions are predicted. In this way, it not only reduces the number of false detections, but it also decreases the time required for detection, since it does not propose a fixed number of zones of invariable size, but they are generated according to each image.

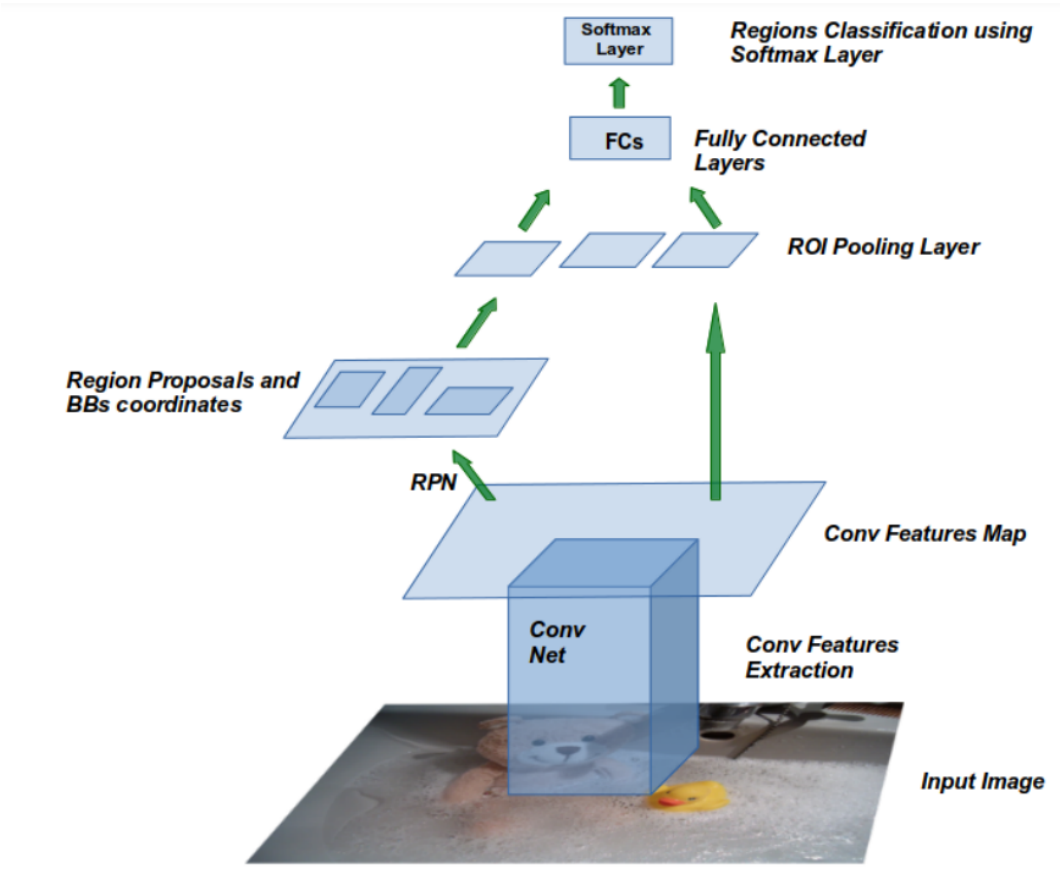

Figure 2.10: High level architecture of Faster R-CNN: a single CNN is used for region proposals and classifications.

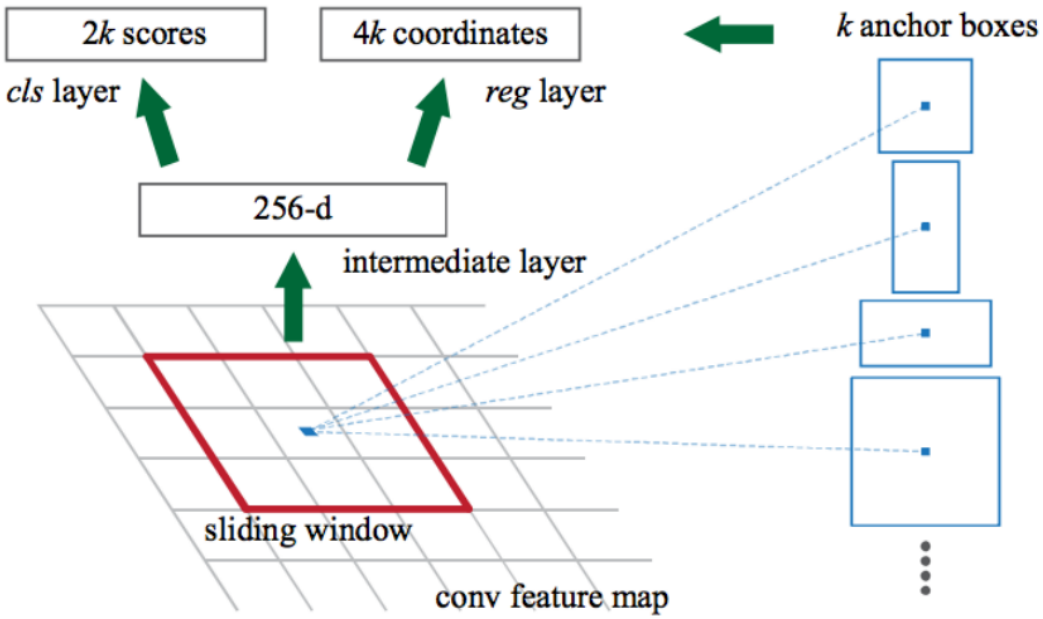

Figure 2.11: The Region Proposal Network slides a window over the convolutional feature map output by the last shared convolutional layer. At each window location, this network takes as input an 3 × 3 spatial window of the conv feature map, mapping this to a lower-dimensional feature. This feature is fed into two sibling fully connected layers - a box-regression layer (reg) and a box-classification layer (cls). Since at each sliding-window location we simultaneously predict k possible proposals (anchors), the reg layer has 4k outputs encoding the coordinates of k boxes, and the cls layer outputs 2k scores that estimate an 'objectness' probability for each proposal .

2.1.6 Mask R-CNN

So far we have analyzed networks able to use CNN features to effectively locate and categorize different objects in an image with bounding boxes. Mask R-CNN [He et al., 2017a] extend such techniques to go one step further and locate exact pixels of each object instead of just bounding boxes (i.e. performing instance segmentation). Mask R-CNN does this by adding a branch to Faster R-CNN that outputs a binary mask that says whether or not a given pixel is part of an object. The branch, as before for the RPN, is just a Fully Convolutional Network

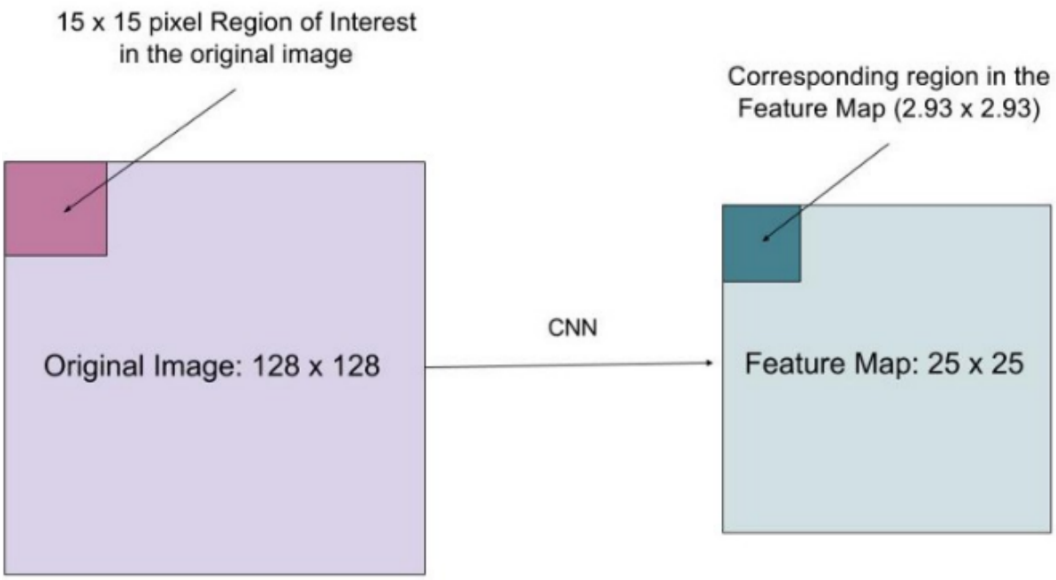

Figure 2.12: Example of mapping from a features map to a smaller one.

(FCN) on top of a CNN based feature map. Even if in principle Mask R-CNN in an extension of Faster-RCNN, the authors had to make one small adjustment to make this pipeline work as expected. Indeed Faster R-CNN is not designed for pixel-to-pixel alignment between network inputs and outputs, and this is evident in how RoIPool performs coarse spatial quantization for feature extraction. To fix this misalignment(seen in Figure 2.8), Mask R-CNN proposes a layer, called RoIAlign, that faithfully preserves exact spatial locations. Specifically, Mask R-CNN predicts an m × m mask from each RoI using a FCN, allowing each layer in the mask branch to maintain the explicit m×m object spatial layout without collapsing it into a vector representation having fewer dimensions. Note indeed that a mask encodes an input object's spatial layout, and thus, unlike class labels or box offsets that are inevitably collapsed into short output vectors by fully-connected layers, extracting the spatial structure of mask can be addressed naturally by the pixel-to-pixel correspondence provided by convolutions. This pixel-to-pixel behavior requires that the RoI features to be well aligned to preserve the explicit per-pixel spatial correspondence, but they are small features maps, as we saw in the previous section. This is the reason why the RoIAlign layer is essential. To understand the RoIAlign layer, imagine we have an image of size 128 × 128 and a feature map of size 25 × 25

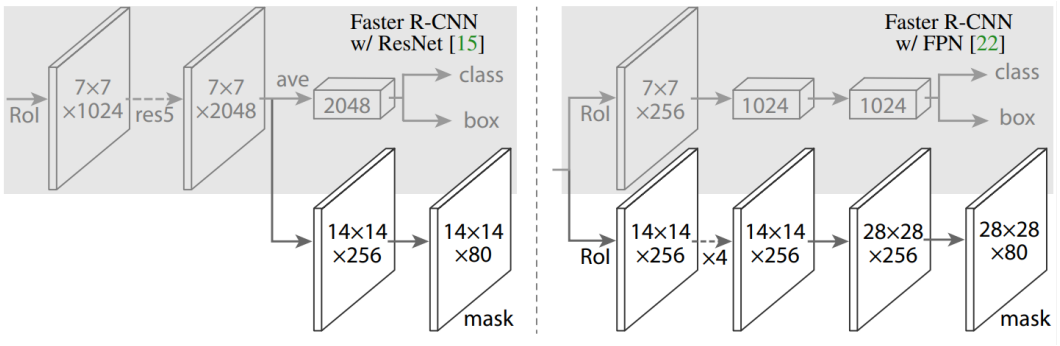

Figure 2.13: Two different head architectures of Mask R-CNN. Source: [He et al., 2017a]

and that we want features the region corresponding to the top-left 15×15 pixels in the original image (see Figure 2.12. The problem here is how to select these pixels from the feature map. We know each pixel in the original image corresponds to about $\frac{25}{128}$ pixels in the feature map. To select 15 pixels from the original image, we just select $15 \times \frac{25}{128} = 2.93$ pixels. When the RoIPool operation is performed, this value is rounded down and we select 2 pixels, causing a slight misalignment. RoIAlign avoids such rounding and uses bilinear interpolation instead, in order to get a precise idea of what would be at pixel 2.93. This, at a high level, is what allows us to avoid the misalignments caused by RoIPool. Mask R-CNN is implemented using multiple architectures that extend new implementations. Figure Figure 2.13 shows the two head architecture of Mask R-CNN.

2.1.7 Residual Neural Networks

When deeper networks are able to start converging, a degradation problem of training accuracy has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error. Figure 2.14 shows a typical example.

In their paper, the authors address the degradation problem by introducing a deep residual learning framework. Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping. Formally, denoting

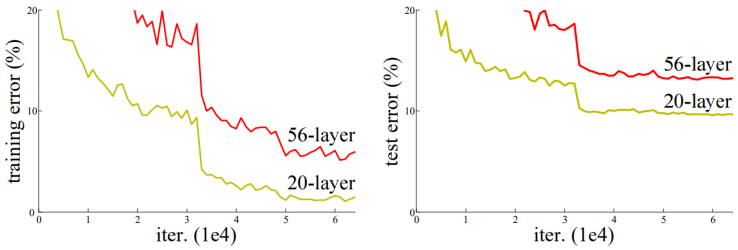

Figure 2.14: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Source: [He et al., 2015]

the desired underlying mapping as $H(x)$, we let the stacked non-linear layers fit another mapping of

$$F(x) := H(x) - x. \tag{2.16}$$

The original mapping is recast into

$$F(x) + x \tag{2.17}$$

The authors hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers. The formulation in Equation 2.17 can be realized by feedforward neural networks with "shortcut connections", as shown in Figure 2.15.

This building block is formally defined as:

$$y = F(x, W_i) + x \tag{2.18}$$

Here x and y are the input and output vectors of the layers considered. The function $F(x, W_i)$ represents the residual mapping to be learned. The shortcut connections in Equation 2.18 introduce neither extra parameter nor computation complexity. This is not only attractive in practice but also important in comparisons between plain and residual networks.

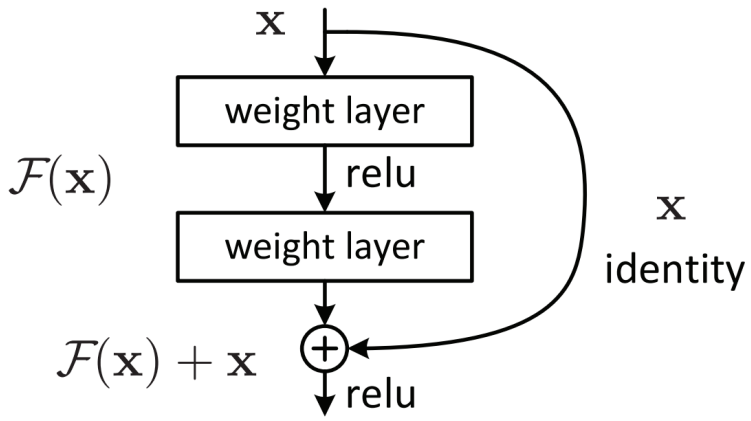

Figure 2.15: Residual Learning: A building block. Source: [He et al., 2015]

The dimensions of x and F must be equal in Equation 2.18. If this is not the case (e.g., when changing the input/output channels), we can perform a linear projection W_s by the shortcut connections to match the dimensions:

$$y = F(x, \{W_i\}) + W_s x \tag{2.19}$$

Several different network architectures are compared on the figure Figure 2.16:

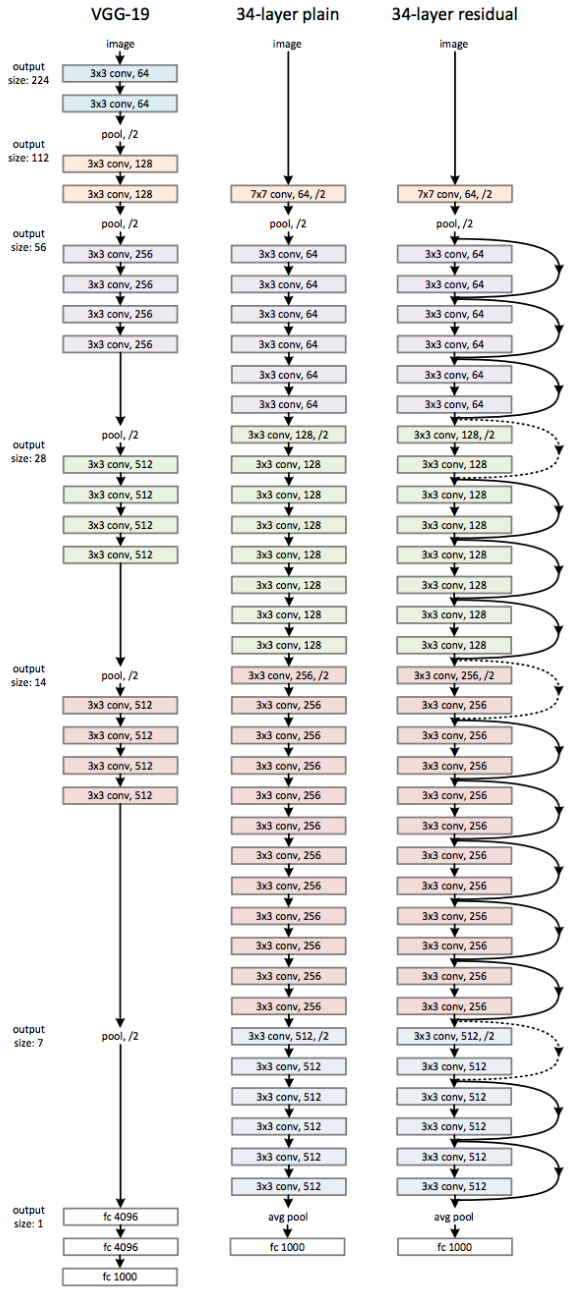

Figure 2.16: Example network architectures for ImageNet. Left: the VGG-19 model [Simonyan and Zisserman, 2015] (19.6 billion FLOPs) as a reference. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Right: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. Table 1 shows more details and other variants. Source: [He et al., 2015]

2.2 Overview of Automatized Microplastics Counting and Classification

Plastic contamination is one of the foremost far reaching issues affecting the marine environment. Moreover, it debilitates sea health, food safety and quality, human well-being, coastal tourism, and contributes to climate change [Lorenzo-Navarro et al., 2021]. In 2015, [Jambeck et al., 2015] estimated that between 4.8 and 12.7 million tons of plastic end up in our seas each year. In addition to the timing related to the plastic disintegration process, another imperative issue is the fracture of them into smaller plastic parts. Those classified as microplastics, smaller than 5 mm, are of increased interest for the biologists since due to their small size they can be ingested by fish and other marine organisms and exchanged through the marine nourishment chains [Carbery et al., 2018]. In this setting, the shape of the microplastic particles within the range of 1–5 mm may indicate the origin. For example, the particles originating from intentional production (primary) tend to exhibit rounded shapes. On the other hand, while the ones originating from the fragmentation of larger pieces normally display an irregular shape. In addition to those microplastic particles, it is fundamental to include those originating from the engineered textures strands, or lines and ropes from angling activities.

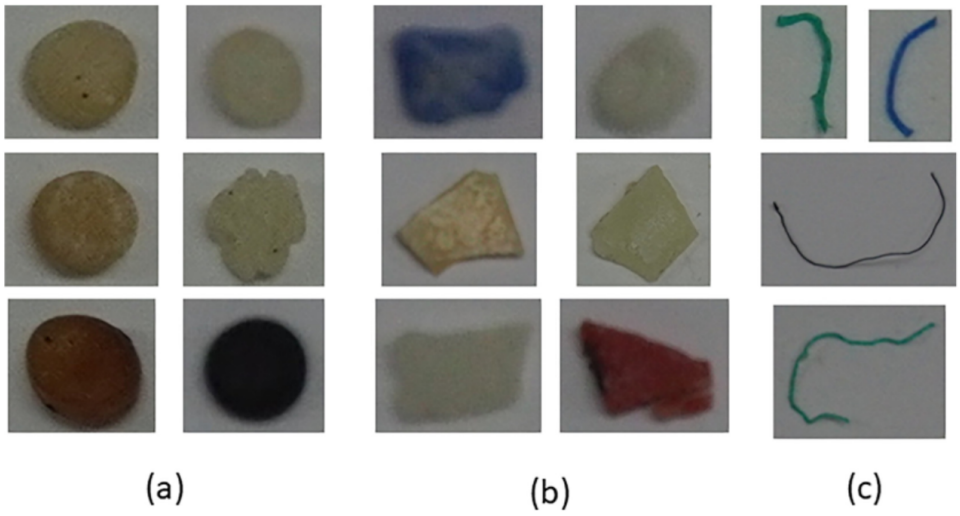

Figure 2.17: Samples of particles cropped images: (a) Pellets, (b) Fragments, (c) Lines. Source [Lorenzo-Navarro et al., 2021]

In a recent study by [Kane et al., 2020], it has been discovered that investigating ocean plastics and microplastics involves not only floating artefacts, but also a large amount of the plastics on the ocean floor due to the transport. This happens due to turbidity currents. Thus, investigating microplastics in various parts of the environment has become a crucial challenge in the understanding of their sources. Therefore, various techniques have been developed to estimate the quantity of plastics in the oceans. One of the methods is observing the quantity and quality of microplastics on the beach proposed by [Pham et al., 2020]. In addition to the the microplastics particles in the oceans, its presence in the freshwater and terrestrial environments is a topic worth the investigation, though it still has not received deserved academic attention [Wong et al., 2020]. In the case of freshwater environment, microplastics quantities are affected and influenced mostly by the population density as well as the waste management systems, whereas for the case of terrestrial ecosystems, the main sources are the agriculture, landfill and water treatment sludge. However, one of the main problems of plastics counting on the beach is that this task is generally labor-intensive in terms of time and other resources relevant to sample collection and processing later on in the laboratory. It should also be born in mind that many of those research

studies require expensive equipment to be conducted, such as various niches of microscopes. The use of stereo microscope in particular for counting and sorting particles based on color, size, brightness, and morphology is the most common identification technique in sediment studies [Hanvey et al., 2017] relevant to the particle size of 1–5 mm. Thus, it was shown that the automatization of image analysis-based identification methods is an important issue in counting, classification, and measuring of particles since the quantification of key dimensions such as particle type or size requires a tedious and lenghty manual labor[Lorenzo-Navarro et al., 2021]. The very same study by [Lorenzo-Navarro et al., 2021] observed that one important limitation in monitoring microplastics is that their visual identification/screening process is a labor-intensive task that must be carried out by trained personnel. Therefore, in that research paper it was proposed that it would be highly recommendable to automatize this task to result in faster sample processing and enabling this working time to be better applied to other relevant tasks. It should be noted that techniques to automatically count and classify microplastic particles with size between 1 mm and 5 mm have not been studied widely.

Several studies have emerged recently showing increasing interest in the image analysis techniques for microplastics characterization. [Mukhanov et al., 2019] classified the microplastic particles into four distinct classes: rounded, irregular, elongated and fiber; leveraging on the ImageJ software to compute three metrics of shape (Feret's diameter, circularity, and area). ImageJ software was also used by [Prata et al., 2019] to develop the Microplastics Visual Analysis Tool (MP-VAT) which had as its aim the automatical counting of fluorescent microplastics stained with Nile Red and specific illumination of wavelength. Similarly to [Mukhanov et al., 2019], the authors proposed the circularity as the property of the particle to use in order to classify it as fiber, fragment or particles, in addition to Feret and MinFeret diameters. The objective was to give an approximation of the largest and smallest dimension of the particle. [Gauci et al., 2019] proposed the use of Matlab software to analyze the microplastic particles collected at four beaches in Malta, computing the following descriptors for each particle: size, roughness, and color. This study used the method of fitting an

ellipse and then using the major and minor axis in order to determine the size of the particle. With respect to the roughness, it was computed as the ratio between the difference of the particle and fitted ellipse areas and the ellipse area. Finally, the color was assigned to the closest of a set of 10 predefined colors in the RGB space.

In their 2020 paper, [Lorenzo-Navarro et al., 2020] presented their System for Microplastics Automatic Counting and Classification (SMACC). Its purpose was to classify and count microplastics particles into five categories using two images as input. The images were taken with a high resolution flat scanner. Prior to the classification of the fragments, an adaptive thresholding was carried out in order for each detected particle to be classified using a cascade classifier; the process achieving 91.1% of accuracy. In recent years, deep learning approaches have been gaining increasing attention due to their impressive performance in many computer vision tasks [Liu et al., 2018a]. Up until 2020, the most important work in the application of deep learning to microplastics analysis has been the work of [Wegmayr et al., 2020] to segment microplastics fibers in microscopy images. In that work, the scientists compared the performance of Mask-RCNN [He et al., 2017b] and Deep Pixel Embeddings [De Brabandere et al., 2017], obtaining the best results with the latter method combined with a heuristic approach for fiber merging in the intersections. All the previous methods used for plastics classification and counting based on image analysis have required special equipment as staining dyes and light [Prata et al., 2019], flat scanner [Mukhanov et al., 2019]; [Gauci et al., 2019]; [Lorenzo-Navarro et al., 2020] or microscopy images [Wegmayr et al., 2020].

On the other hand, the methodology proposed in [Lorenzo-Navarro et al., 2021] is based on images taken with cameras or mobile phones with a resolution of at least 16 million pixels. A further reason for this research, according to the scientists, is the recognition that introducing deep learning techniques improves both the performance of traditional computer vision methods and even the use of manual classification, which is prone to mistakes due to being a tedious task. Thus, with respect to the previous works, the work by [Lorenzo-Navarro et al., 2021] presented a novel method for automatizing the processes of counting and classifying microplastic particles without complex and expensive laboratory

equipment. The scientist quote the main contributions of the paper to be threefold: the use of novel techniques based on deep learning for this problem, possibility of using simple phone camera images instead of expensive equipment, and reduce processing time.

Chapter 3

An object detection system for microplastics

In order to come up with an efficient software for detection and classification of microplastics particles, several decisions needed to be made. First and foremost, the choice of a computer vision model for the problem at hand was crucial. The model and the reasoning for its choice are discussed in section 3.2.

Once this decision was taken, it was necessary to adapt the dataset to fit the input requirements of the chosen model. The decision to decompose the image into smaller ones was taken, as described in the section section 4.5. This was a solution proposed to the twofold problem of the dataset at hand: fistly, the problem of small quantity of data in the initial dataset discussed in section 4.3, and secondly, the problem of high-resolution large images with small objects, described in section section 4.4. The process for deciding on the size of the output images was described in section 5.1. At last, the final dataset is described in section section 4.6, together with the train/validation/test split.

Next issue was the choice of hyperparameter settings for training the model on our dataset, addressed in the last two sections of the chapter 5. The results of five experiments are described in the section chapter 6.

After that, it was time to reunite the detections of the cropped up images, to yield the

information relevant to the input test image. The relevant information is the number of particles per image, as well as the bounding box coordinates and labels for detected particles. This process is described in section 4.7.

3.1 Architecture of the Approach

The system takes as input an image of microplastics particles arranged on a white background, of size 3456*4608 pixels. The particles shouldn't mutually overlap. In the preprocessing step the input image is cropped into smaller ones, in order to fit the requirements of the classifier. The cropped images are classified, and a CSV file with bounding box coordinates, detected class, confidence, and coordinates of the cropped input image relative to the original image is stored. In the postprocessing step, this CSV file is processed in order to eliminate duplicate detections and keep the one with the highest confidence. This step leads to the output of the system, which is a CSV file with all the detected particles, detected class, confidence and bounding box coordinates, as well as an output image with bounding boxes with class name and confidence.

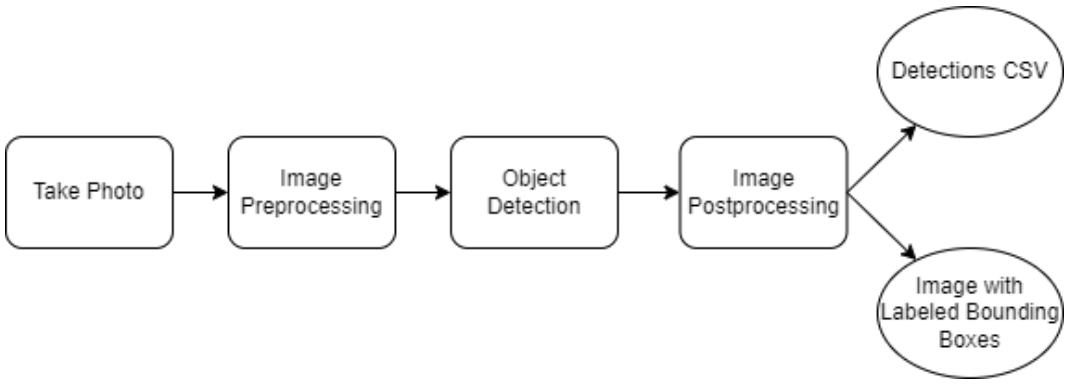

Figure 3.1: Pictorial representation of the System

3.2 YOLO

With respect to what was said in subsection 2.1.2, this architecture approaches object detection from another point of view. It is a single convolutional neural network that predicts bounding boxes and detection probabilities directly from the entire image in a single evaluation. This is what is used to reduce the time needed for this network to perform the detections. In fact, its name, You Only Look Once [Redmon et al., 2015], comes from the ability of this network to resemble human behavior by performing detections with a single glance at the image. YOLO has several versions, it is a model that has been improved over time. Each version of YOLO has added new features that keep it as one of the most popular architectures in its field. At the date of the beginning of this work, the last version was the fifth one. However, it is necessary to explain each of the versions in order to fully appreciate the operation of this network.

3.2.1 YOLOv1

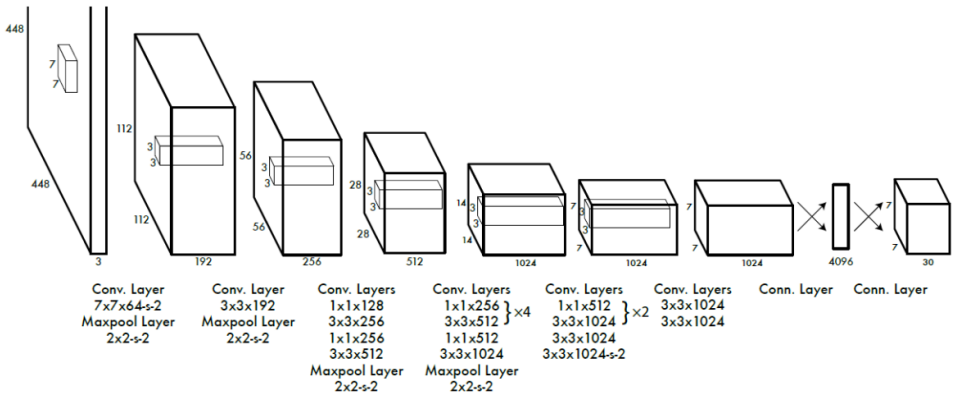

Figure 3.2: YOLO Structure as shown in the original paper. Source: [Redmon et al., 2015]

To achieve these detections in fairly short times, YOLO divides the image into a number of cells. In each cell, an attempt is made to predict a single object, namely, to detect whether the "center of the object" is in that cell. Once this point is obtained, an attempt is made to

predict a fixed number of regions that delimit the object. This is done by multiplying the conditional class probabilities and the individual box confidence predictions,

$$P(Class_i \mid Object) * P(Object) * IoU_{truth}^{pred} = P(Class_i) * IoU_{truth}^{pred} \tag{3.1}$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object. This process is repeated for each cell of the image. Therefore, for each cell, the network will generate a vector indicating the position of each possible bounding box, together with the probability that this box is the correct one, which can be seen on Figure 3.3.

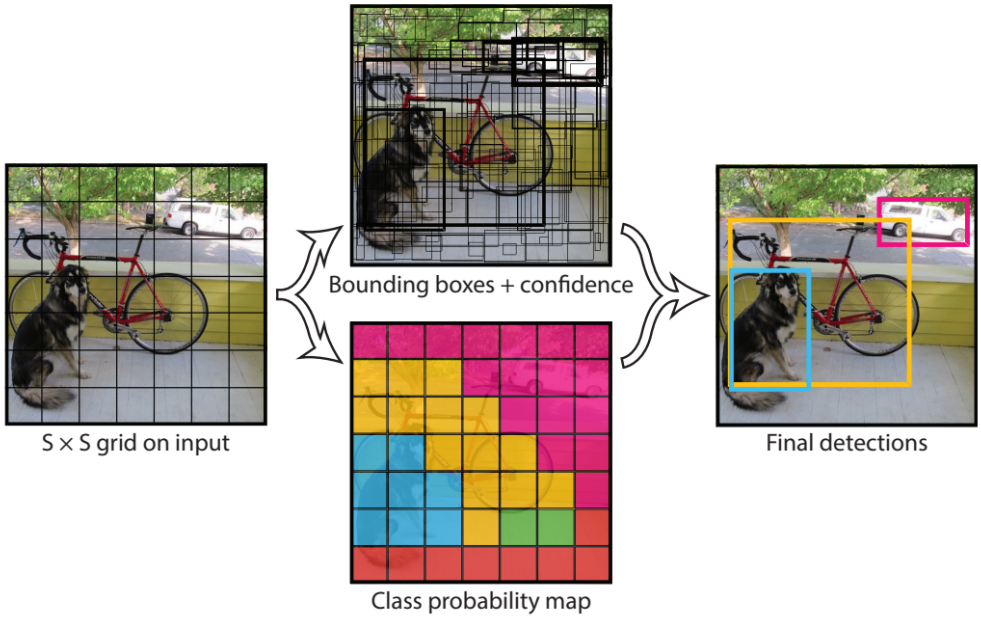

Figure 3.3: The Model. The system models detection as a regression problem. It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × (B * 5 + C) tensor. Source: [Redmon et al., 2015]

This list with probabilities that the detected object belongs to each of the possible classes is well reflected in the Figure 3.4

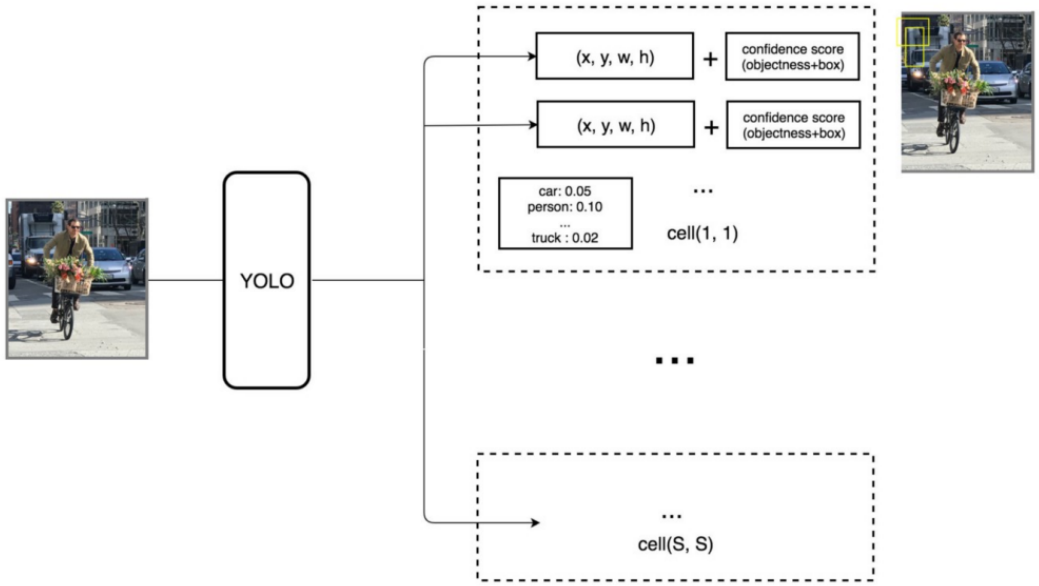

Figure 3.4: YOLO Outputs. Source: [Cerezo, 2020]

Training

The authors use a linear activation function for the final layer and all other layers use the following leaky rectified linear activation:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise.} \end{cases} \tag{3.2}$$

The authors optimize for sum squared error in the output of their model. They modify it by increasing the loss from bounding box coordinate predictions and decreasing the loss from confidence predictions for boxes that don't contain objects. They use two parameters, λ_{coord} and λ_{noobj} to accomplish this. They set $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$.

As has been mentioned already, YOLO predicts multiple bounding boxes per grid cell. At training time the authors only wanted one bounding box predictor to be responsible for each object. We assign one predictor to be "responsible" for predicting an object based on which prediction has the highest current IoU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at predicting certain sizes,

aspect ratios, or classes of object, improving overall recall. During training we optimize the following, multi-part loss function:

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2.
\end{aligned}
\tag{3.3}
$$

where $\mathbb{1}_i^{obj}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{obj}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

Although the detection of a single object per cell gives YOLO its main advantage, the speed of detection, it is also its main disadvantage. The first version of YOLO has quite a few problems in detecting objects when they are overlapping or very close to each other.

3.2.2 YOLOv2 (YOLO9000)

To solve the problem of object overlapping, the second version of this architecture, known as YOLOv2 or YOLO9000 [Redmon and Farhadi, 2017], focuses on improving the accuracy of this network, without losing its characteristic detection speed. The increase in accuracy is achieved by introducing new techniques and changes to the architecture, some of the most important ones discussed below.

Batch Normalization

In simple terms, BN provides every layer of a neural network the capability to learn independently of other layers.

It normalizes the output of the previous activation layer by subtracting the mean of the batch and dividing by the standard deviation of the batch, and this output gets multiplied by a standard deviation parameter $\gamma^{(k)}$, then a mean parameter $\beta^{(k)}$ is added.

For a layer of the network with d-dimensional input, $x = (x^{(1)}, ..., x^{(d)})$, each dimension of its input is then normalized (i.e. re-centered and re-scaled) separately,

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\left(\sigma_B^{(k)}\right)^2 + \epsilon}}, \tag{3.4}$$

where

$k \in [1, d]$ and $i \in [1, m]$;$\mu_B^{(k)}$ and $\sigma_B^{(k)}$ are the per-dimension mean and standard deviation, respectively.

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}. \tag{3.5}$$

This process leads to notable changes in convergence and removes the need for other forms of regularization (such as dropout) without overfitting the model.

Darknet19: High Resolution Classifier

High resolution classifier is deployed, increasing the required resolution of the input image from 256*256 to 448*448. Namely, in Darknet19 authors use mostly 3 × 3 filters and double the number of channels after every pooling step. They use global average pooling to make predictions as well as 1 × 1 filters to compress the feature representation between 3 × 3 convolution. The final model, called Darknet-19, has 19 convolutional layers and 5 maxpooling layers. For a full description see Figure 3.5. Darknet-19 only requires 5.58 billion operations to process an image yet achieves 72.9% top-1 accuracy and 91.2% top-5 accuracy on ImageNet.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 3.5: Darknet-19 Architecture. Source: [Redmon and Farhadi, 2017]

Anchor Boxes

YOLOv2 uses hand-picked priors (anchor boxes) to predict offsets and confidences for bounding boxes at every location in a feature map. This simplifies the problem and makes it easier for the network to learn. This is done by removing the FC layers at the end which predicted the bounding boxes, and using anchor boxes instead.

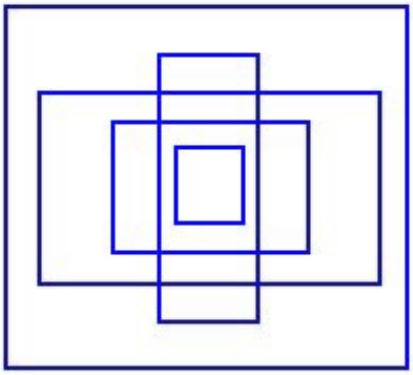

Figure 3.6: Five Anchor Boxes

Thus the network output varies slightly. There is no longer a vector with the probabilities that an object is of a certain class for all the bounding boxes of a cell. Now we obtain the same vector for each bounding box allowing, therefore, the detection of several objects within the same cell. Figure 3.7 allows the visualization of this new configuration.

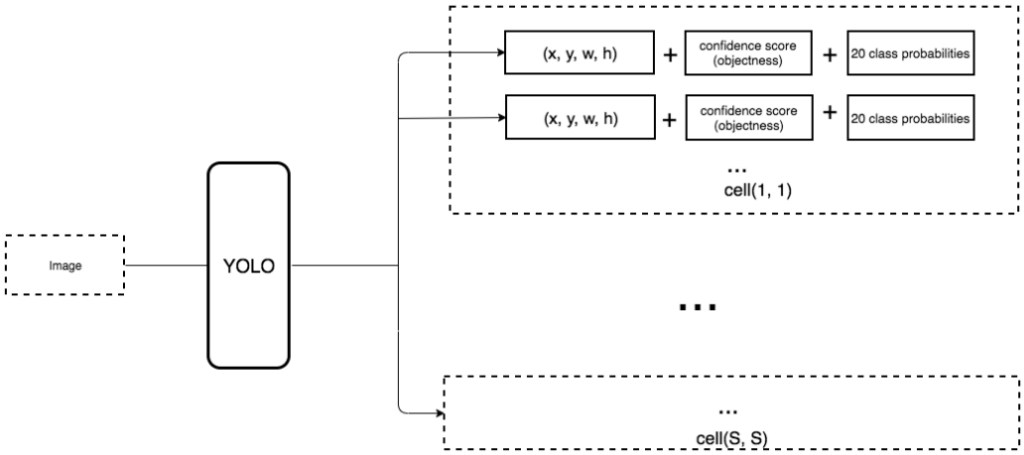

Figure 3.7: YOLOv2 outputs as shown in the original YOLO9000 paper. Source: [yol,]

Dimension Clusters

This method suggests running K-means clustering on the training set bounding boxes to automatically find good priors, instead of starting with hand-picked anchor box dimensions.

The distance metric chosen is described in Equation 3.6

$$d(box, centroid) = 1 - IoU(box, centroid) \tag{3.6}$$

Direct Location Prediction

Direct prediction of localization from the proposed bounding boxes is implemented. A problem that results from directly predicting (x, y) locations of the bounding box is model instability. What region proposal networks do instead is predict values t_x and t_y and the (x, y) coordinates are calculated as follows:

$$\begin{aligned} x &= (t_x * w_a) - x_a \\ y &= (t_y * h_a) - y_a \end{aligned} \tag{3.7}$$

Due to the offsets not being constrained here, any anchor box can end up at any point in the image, regardless of what location predicted it — the model takes a long time to stabilize for predicting sensible offsets.

An alternative is to predict location coordinates relative to the location of the grid cell, just like in YOLO. The network predicts 5 coordinates for every bounding box — t_x, t_y, t_w, t_h, and t_0, and a logistic function is applied to constrain the values of these coordinates to [0, 1].If the cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height p_w, p_h, then the predictions correspond to:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w * e^{t_w} \\ b_h &= p_h * e^{t_h} \\ P(object) * IoU(b, object) &= \sigma(t_0) \end{aligned} \tag{3.8}$$

Illustration of this can be seen in Figure 3.8.

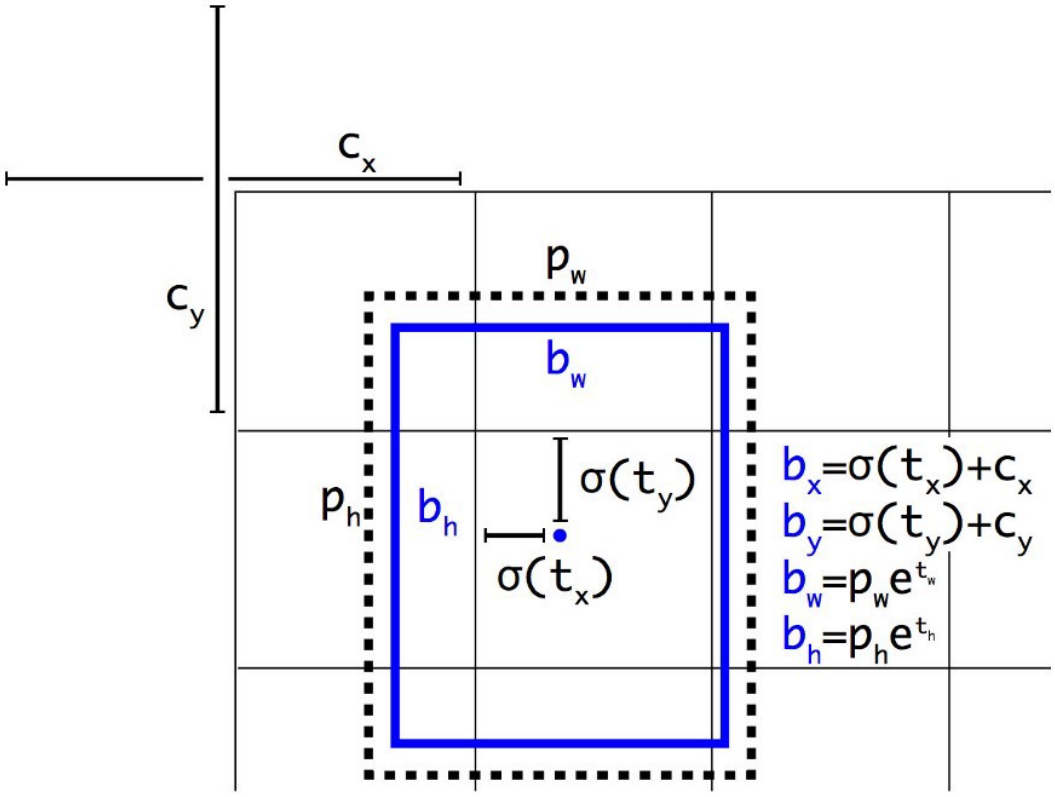

Figure 3.8: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. Source: [Redmon and Farhadi, 2017]

As can be observed in the Figure 3.9, all these changes yield a refinement of the system.

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figure 3.9: Comparison of the precision as shown in the original YOLO9000 paper. Source: [Redmon and Farhadi, 2017]

In addition, the improvement in accuracy is accompanied by an increase in detection speed. This is achieved through a change in the main structure of the network to a darknet architecture [Redmon, 2016] that requires fewer operations in order to perform the detections.

As a result of previous changes (and some other ones, not listed here for the sake of brevity), the new version of YOLO generates better detection and reduced detection times than its predecessor.

3.2.3 YOLOv3

In its third version [Redmon and Farhadi, 2018], YOLO becomes an architecture that, in exchange for the reduction of its speed, gains quite a lot of robustness. Network's capacity of recognizing small objects is improved.

One of its main differences with respect to its predecessor is the change in structure. Previously, YOLO used a 19-layer darknet to which it added 11 more layers. In In the third version, it uses a 53-layer darknet to which it adds another 53 layers, bringing the total to a staggering 106 convolutional layers. This is the main reason for the speed loss of YOLO. However, this allows for a unique strategy: three-scale detection. As can be seen in an impressive illustration of its architecture on figure Figure 3.10 , as the image progresses through

the network, the size of the cells is reduced and consequently the number of cells increases.

This is precisely what allows this version of YOLO to detect small objects. The higher the number of cells, the more objects it can detect (note that only one object was allowed per cell). Therefore, the more objects it can detect, the more accurate it is. This allows YOLO to catch up again with the rest of the contemporary architectures in terms of accuracy.

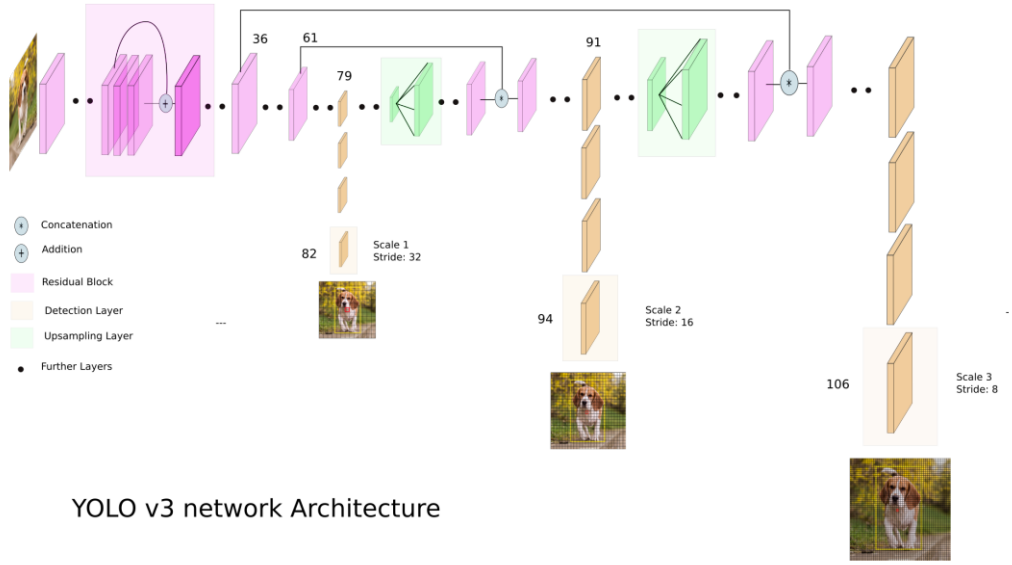

Figure 3.10: YOLOv3 structure as shown in the original paper. Source: [Redmon and Farhadi, 2018]

3.3 YOLO Enhancements

The first three versions were developed by Joseph Redmon, but later on other researchers have studied how to improve YOLO algorithms in terms of both accuracy and speed. This paragraph presents the most important YOLO improvements, highlighting the main changes and the increases in performance compared to the previous versions.

3.3.1 YOLOv4

The fourth version of YOLO, dubbed YOLOv4, which was introduced by Alexey Bochkovskiy in 2020 and is faster and more precise than the earlier models, was described in [Bochkovskiy et al., 2020a]. Specifically, the primary objective was to increase the real-time object detector's precision and develop a CNN that can be used with a standard GPU (Graphics Processing Unit). YOLOv4 is intricate and there have been many improvements over the original YOLO, but ultimately it can still be trained on a single GPU. Finding the ideal balance between the input network resolution, the number of convolutional layers, the number of parameters, and the number of filters is the goal of the architecture selection process. Here are some of the most prominent features of YOLOv4:

CSP Darknet53 as the Backbone

Although CSPDarknet53 is built on Darknet53 and has 29 convolutional layers, it uses the CSPNet method [Wang et al., 2019] to lower computation costs, use less memory, and lessen the issue of heavy inference computations.

DenseBlock and DenseNet. To improve accuracy, we can design a deeper network to extend the receptive field and to increase model complexity. And to ease the training difficulty, skip-connections can be applied. We can expand this concept further with highly interconnected layers.

A Dense Block contains multiple convolution layers with each layer H_i composed of batch normalization, ReLU, and followed by convolution. Instead of using the output of the last layer only, H_i takes the output of all previous layers as well as the original as its input. i.e. $x_0, x_1, ...$ and x_{i-1}. Each H_i below outputs four feature maps. Therefore, at each layer, the number of feature maps is increased by four — the growth rate.

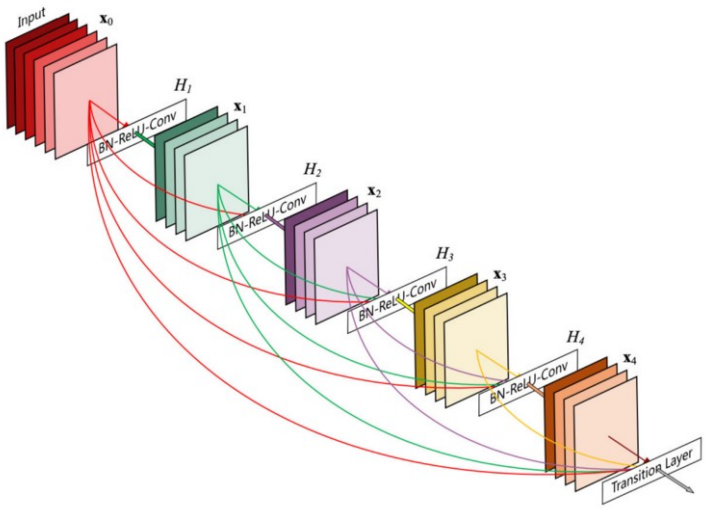

Figure 3.11: A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input. Source: [Huang et al., 2017]

Then a DenseNet can be formed by composing multiple Dense Block with a transition layer in between that composed of convolution and pooling.

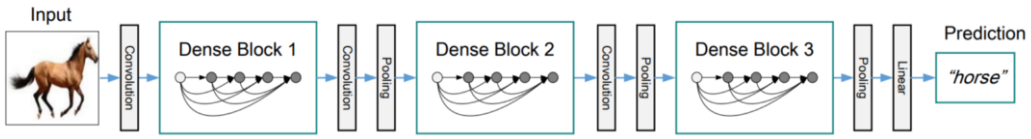

Figure 3.12: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling. Source: [Huang et al., 2017]

Detailed architectural design can be seen on Figure 3.13.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 3.13: DenseNet architectures for ImageNet. The growth rate for all the networks is k = 32. Source: [Huang et al., 2017]

Cross-Stage-Partial Conections (CSP) The mainstream CNN architectures' output is usually a linear or non-linear combination of the outputs of intermediate layers. Therefore, the output of a k-layer CNN can be expressed as follows:

$$\begin{aligned} y &= F(x_0) = x_k \\ &= H_k(x_{k-1}), H_{k-1}(x_{k-2}), H_{k-2}(x_{k-3}), ..., H_1(x_0), x_0) \end{aligned} \tag{3.9}$$

where F is the mapping function from input x_0 to target y, which is also the model of the entire CNN. As for H_k, it is the operation function of the k^{th} layer of the CNN.

The state-of-the-art methods put their emphasis on optimizing the H_i function at each layer, and the authors propose that CSPNet directly optimizes the F function as follows:

$$y = M(x[x_0^{'}, T(F(x_0^{''})]) \tag{3.10}$$

where x_0 is split into two parts along channel and it can be represented as $x_0 = x_0^{'} + x_0^{''}$. T is the transition function used to truncate the gradient flows of H_1, H_2, ..., H_k, and M is the transition function used to mix the two segmented parts.

CSPNet [Wang et al., 2019] thus separates the input feature maps of the DenseBlock into two parts: $x_0 = x_0^{'} + x_0^{''}$. The first part $x_0^{'}$ bypasses the DenseBlock and becomes part of the input to the next transition layer. The second part $x_0^{''}$ will go throughout the Dense block as below.

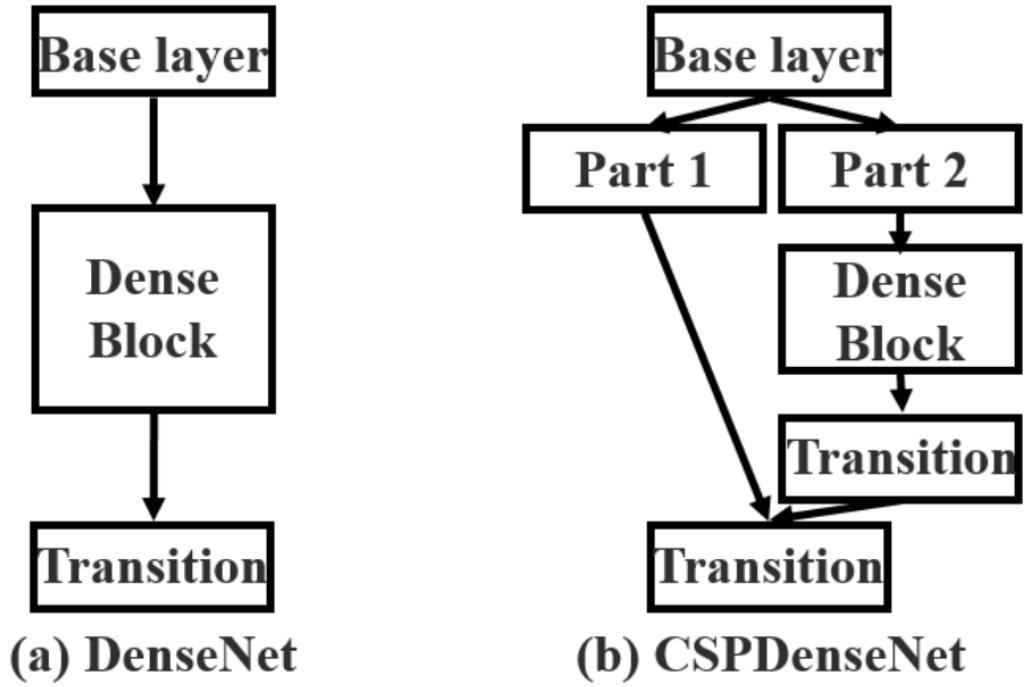

Figure 3.14: Different kind of feature fusion strategies. (a) single path DenseNet, (b) proposed CSPDenseNet: transition → concatenation → transition. Source: [Wang et al., 2019]

This new design reduces the computational complexity by separating the input into two parts — with only one going through the Dense Block.

CSPDarknet53. YOLOv4 utilizes the CSP connections above with the Darknet-53 below as the backbone in feature extraction. The CSPDarknet53 model has higher accuracy in object detection compared with ResNet based designs even though they have a better classification performance. The final choice for YOLOv4 is therefore CSPDarknet53.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.15: Darknet53. Source: [Redmon and Farhadi, 2018]

SPP (Spatial Pyramid Pooling Layer)

The addition of the SPP block (Spatial Pyramid Pooling) [He et al., 2014] helps to improve the receptive field by separating out the most important context information. SPP replaces the last pooling layer (after the last convolutional layer) with a spatial pyramid pooling layer. The feature maps are spatially divided into m×m bins with m, say, equals 1, 2, and 4 respectively. Then a maximum pool is applied to each bin for each channel. This forms a fixed-length representation that can be further analyzed with FC-layers.

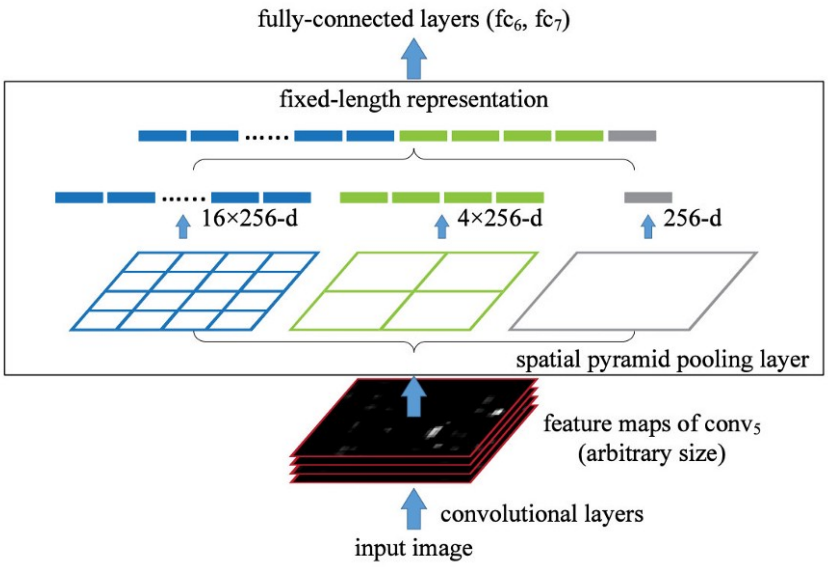

Figure 3.16: A network structure with a spatial pyramid pooling layer. Here 256 is the filter number of the $conv_5$ layer, and $conv_5$ is the last convolutional layer. Source: [He et al., 2014]

Many CNN-based models containing FC-layers and therefore, accepts input images of specific dimensions only. In contrast, SPP accepts images of different sizes. Nevertheless, there are technologies like fully convolutional networks (FCN) that contain no FC-layers and accept images of different dimensions. This type of design is particularly useful for image segmentation which spatial information is important. Therefore, for YOLO, converting 2-D feature maps into a fixed-size 1-D vector is not necessarily desirable. In YOLO, the SPP is modified to retain the output spatial dimension. A maximum pool is applied to a sliding kernel of size say, 1×1, 5×5, 9×9, 13×13. The spatial dimension is preserved. The features maps from different kernel sizes are then concatenated together as output.

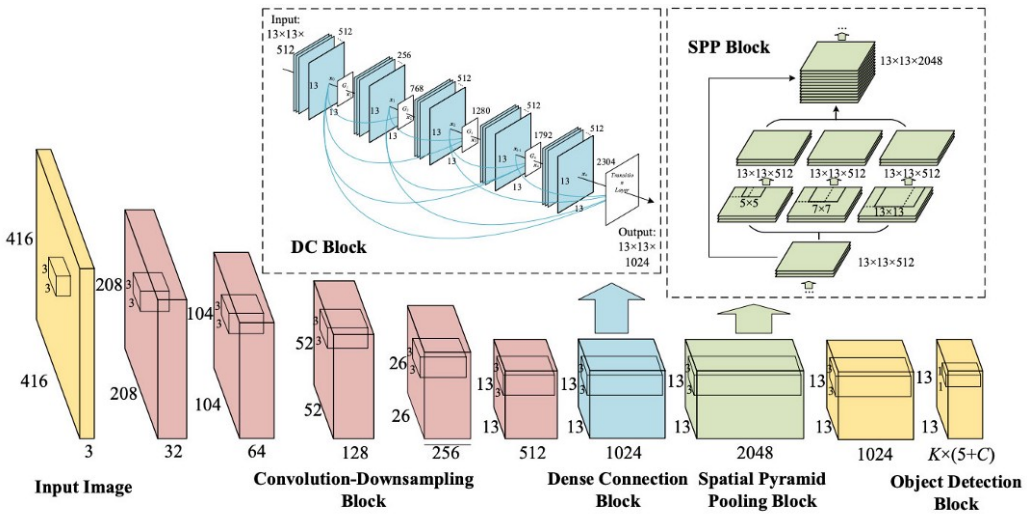

Figure 3.17: Dense Connection and Spatial Pyramid Pooling Based YOLO. Source: [Huang et al., 2020]

Path Aggregation Network (PAN or PANet)

PANet is present in the neck of the YOLOv4 model and it is mainly incorporated in the model to enhance the process of instance segmentation by preserving spatial information. The reason why PANet is chosen for instance segmentation in YOLOv4 is because of its ability to preserve spatial information accurately which helps in proper localization of pixels for mask formation. For the sake of brevity of this thesis, it cannot be discussed in detail here, though the details can be seen in [Liu et al., 2018b]. Full architecture of this network can be seen in Figure 3.18.

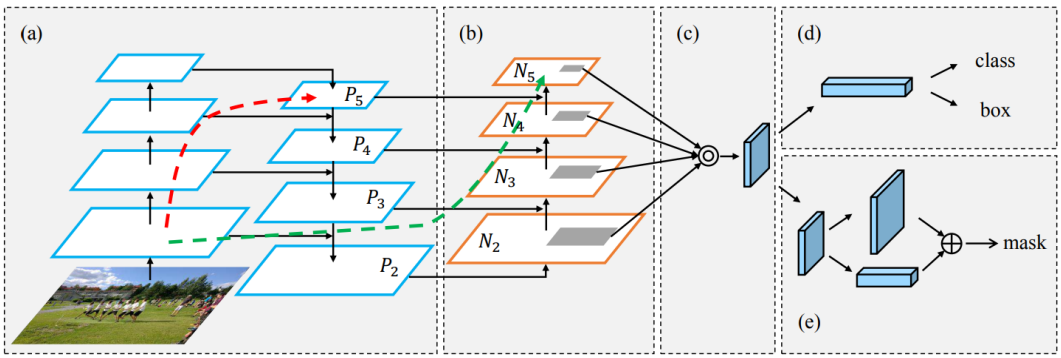

Figure 3.18: Illustration of PANet framework. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. (d) Box branch. (e) Fully-connected fusion. Note that we omit channel dimension of feature maps in (a) and (b) for brevity. Source: [Liu et al., 2018c]

YOLOv3 as the head

The head of the architecture, sometimes referred to as the object detector, is anchor-based and is identical to YOLOv3.

Mosaic Data Augmentation

To create a new training image, four different training images are mixed together. This method is intriguing because it combines various contexts into one input image, allowing for the recognition of items outside of their typical environment. When it was used, YOLOv4 outperformed all other alternative detectors in terms of speed and precision.

Figure 3.19: Mosaic represents a new method of data augmentation. Source: [Redmon and Farhadi, 2018]

Eliminating Grid Sensitivity

Grid Sensitive technique [Bochkovskiy et al., 2020b] is used to enhance the prediction of bounding boxes located on grid boundaries. The boundary box b is computed as in Figure 3.8. The equations

$$\begin{aligned} b_x &= \sigma(t_x) + c_x, \\ b_y &= \sigma(t_y) + c_y, \end{aligned} \tag{3.11}$$

where c_x and c_y are always whole numbers, are used in YOLOv2 and YOLOv3 for evaluating the object coordinates, therefore, extremely high t_x absolute values are required for the b_x value approaching the c_x or $c_x + 1$ values. We solve this problem through multiplying the sigmoid by a factor exceeding 1.0, so eliminating the effect of grid on which the object is undetectable.

Additionally, YOLOv4 can be trained and operated on a single standard GPU, allowing for widespread application. YOLOv4 obtains an AP value of 43.5% on the COCO dataset [Lin et al., 2014], becoming the best object detector model at that time, in terms of both speed and accuracy. As can be observed in the Figure 3.20 compared to YOLOv3, the AP and FPS have increased by 10% and 12% respectively.

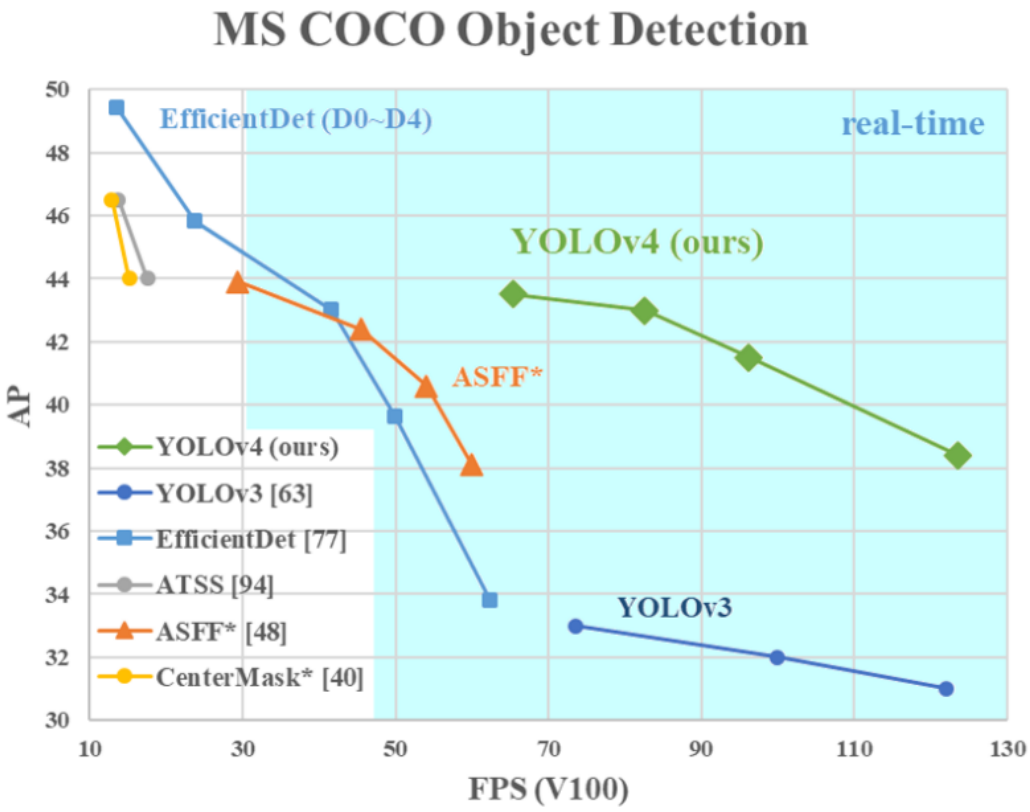

Figure 3.20: Comparison of YOLOv4 and other state-of-the-art object detectors. Source: [Lin et al., 2014]

3.3.2 PP-YOLO

Other researchers proposed the PP-YOLO object detector after the introduction of YOLOv4 [Long et al., 2020]. The name alludes to the fact that all tests are carried out using the open source Paddle (PArallel Distributed Deep LEarning) ML architecture. The authors' objective in this instance was to implement an object detector as well. Thus, one of the primary issues is finding a detector with balanced efficacy and efficiency. Unlike YOLOv4, due to the hardware limitations in this field, the authors did not explore different backbone networks, neck blocks or other structural changes. Considering that YOLOv3 was widely used in practice, they created a model based on it and their focus was on stacking some tricks to get better performance. Many of these techniques had to be modified slightly because they could not be used directly on the YOLOv3 structure.

Backbone

Unlike in YOLOv4 paper, the authors of PP-YOLO did not research various backbone architectures. Instead, ResNet-50-vd, a type of residual neural network, was used in place of the original backbone Darknet-53. The reason behind it is that the ResNet has been extensively used and researched.

Batch Size

The authors increased the training batch size from 64 to 192 since doing so makes the training more stable.

Eliminating Grid Sensitivity

Eliminating Grid Sensitivity has been desctibed previously in section 3.3.1.

Loss function

They also alter the loss that is used in bounding box regression. Although L1 loss is used for this task in YOLOv3, it is not fit to the mAP evaluation measure, which is heavily dependent on IOU (these metrics are described in section 1.1). As a result, they employed IOU loss as a solution to this issue.

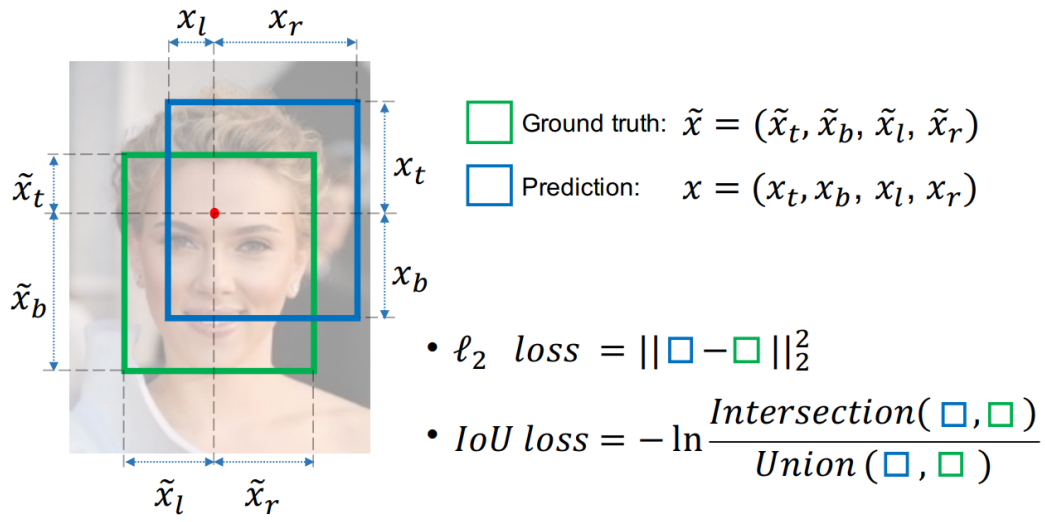

Figure 3.21: Illustration of IoU loss and l_2 loss for pixel-wise bounding box prediction. Source: [Yu et al., 2016]

Figure 3.22 compares the performances of PP-YOLO with other cutting-edge object detectors, particularly with YOLOv4. More specifically, this model can reach a mAP of 45.2% and an inference speed of 72.98% on the COCO dataset.

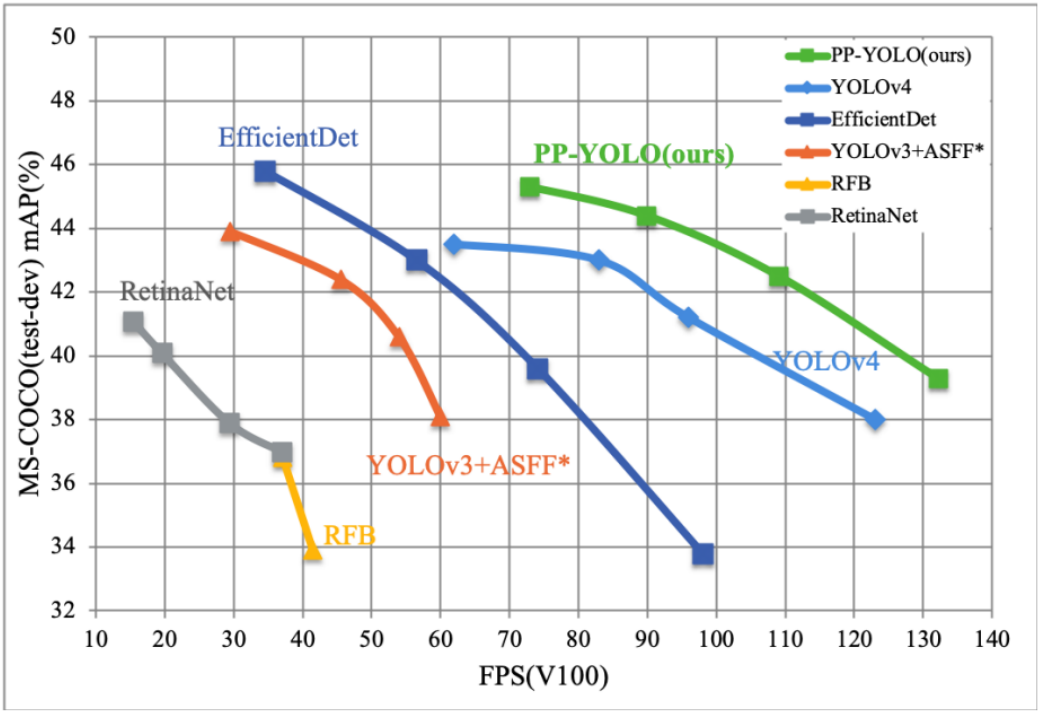

Figure 3.22: Comparison of PP-YOLO and other state-of-the-art object detectors. Source: [Long et al., 2020]

3.3.3 Scaled-YOLOv4

Scaled-YOLOv4 is an additional YOLO variant built on YOLOv4 that was put forth by Wang and Bochkovskiy in 2020 [Bochkovskiy et al., 2020a]. As the name implies, they developed a network scaling strategy that alters the YOLOv4 network's depth, width, resolution, and structure to scale both up and down.

General Principle of Scaling

For the k-layer CNNs with b base layer channels, the computations of ResNet layer is:

$$k * [conv(1 \times 1, b/4) \rightarrow conv(3 \times 3, b/4) \rightarrow conv(1 \times 1, b)]. \tag{3.12}$$

As for the Darknet layer, the amount of computation is:

$$k * [conv(1 \times 1, b/2) \rightarrow conv(3 \times 3, b)]. \tag{3.13}$$

Table 3.1: FLOPs of different computational layers with different model scaling factors

Model	**Original**	Size α	Depth β	Width γ
Res layer	r = 17whkb^2/16	$\alpha^2 r$	βr	$\gamma^2 r$
Dark layer	d = 5whkb^2	$\alpha^2 d$	βd	$\gamma^2 d$

Table 3.2: FLOPs of different computational layers with/without CSP-ization.

Model	**Original**	to CSP
Res layer	r = 17whkb^2/16	whb^2(3/4 + 13k/16)
Dark layer	d = 5whkb^2	whb^2(3/4 + 5k/2)

Let the scaling factors that can be used to adjust the image size, the number of layers, and the number of channels be α, β, and γ, respectively. When these scaling factors vary, the corresponding changes on FLOPs are summarized in Table 3.1.

It can be seen from Table 3.1 that the scaling size, depth, and width cause increase in the computation cost. They respectively show square, linear, and square increase.

The CSPNet [Wang et al., 2019] can be applied to various CNN architectures, while reducing the amount of parameters and computations. In addition, it also improves accuracy and reduces inference time. We apply it to ResNet and Darknet and observe the changes in the amount of computations, as shown in Table 2.

From the figures shown in Table 3.2, we observe that after converting the above CNNs to CSPNet, the new architecture can effectively reduce the amount of computations (FLOPs) on ResNet and Darknet by 23.5% and 50.0%, respectively. Therefore, we use CSP-ized models as the best model for performing model scaling.

This study's primary finding is therefore evidence that an object detection neural network based on the Cross-Stage Partial Connection Block technique [Parico and Ahamed, 2021] performs well on both small and large networks without sacrificing accuracy. This technique achieves good scores on MSCOCO 2017 object detection dataset, as shown in Figure 3.23.

Namely, it achieves the highest accuracy of them all, 56.0% AP.

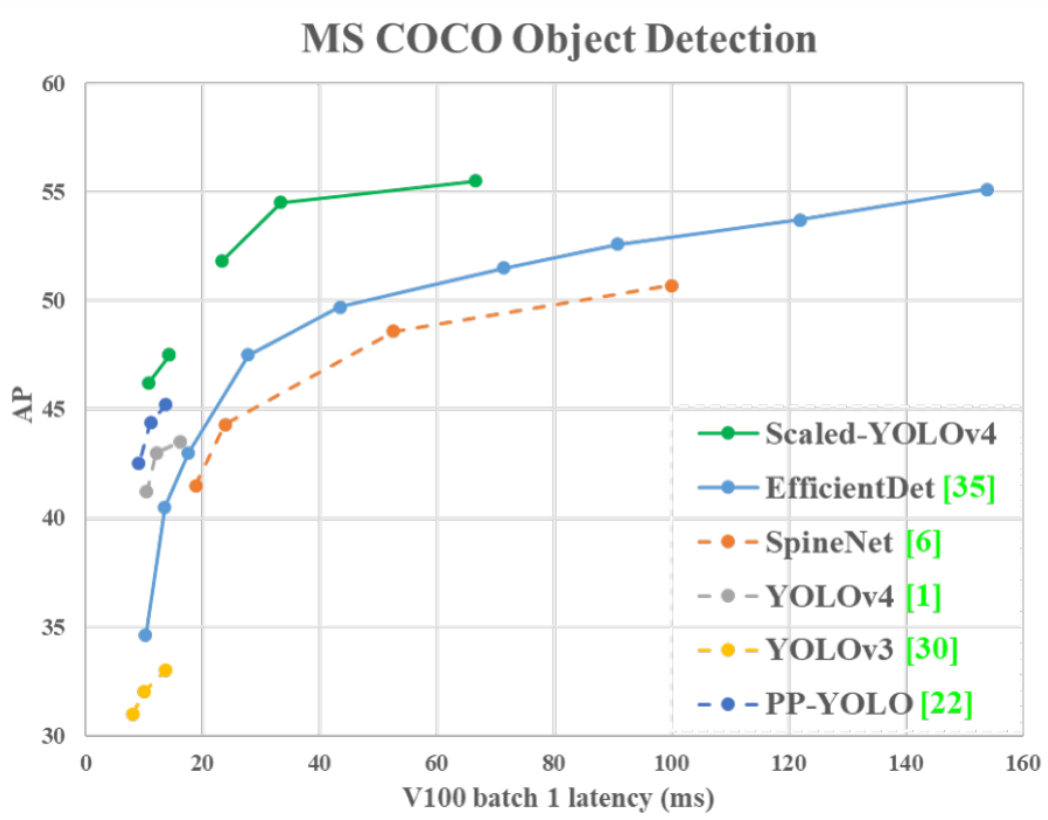

Figure 3.23: Comparison of PP-YOLO and other state-of-the-art object detectors. Source: [Long et al., 2020]

3.3.4 YOLOv5

Glenn Jocher introduced YOLOv5 using the Pytorch framework not long after the release of YOLOv4. However, it has the same design as YOLOv4, with CSPDarknet53 serving as the backbone, SPP and PANet serving as neck blocks, and a YOLOv3 anchor-based head. He did not produce a paper to go along with his release. Consequently, all the remarks concerning YOLOv4 still apply in this instance. YOLOv5 is the newest and lightest version. It employs the PyTorch framework in place of the Darknet architecture.

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

Figure 3.24: YOLOv5 weights. Source: [ult,]

It contains several pretrained models, starting with YOLOv5n, which is the smallest one, all the way to to YOLOv5x that is the largest. Taking these differences into account, the Figure 3.24 shows the quantity of parameters, accuracy, and GPU inference time. The comparison of these models can be observed on Figure 3.25.

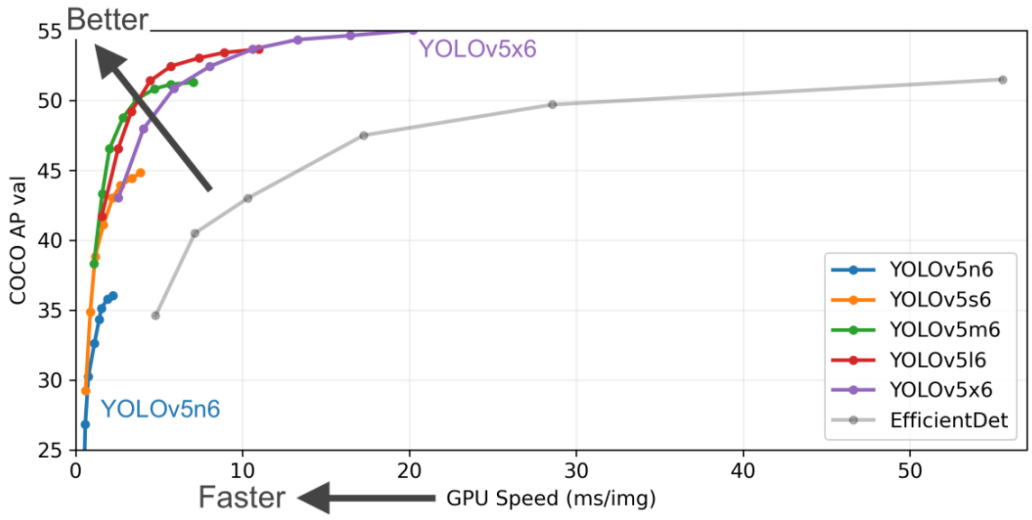

Figure 3.25: YOLOv5 weights. Source: [ult,]

The pretrained checkpoint used further on in training the model with microplastics data is YOLOv5s6, as it was it is deemed to have an optimal balance of speed and accuracy. It is well compatible with the software and hardware tools used to produce the final system for counting and classifying microplastics particles.

The architecture of the YOLOv5 model as implemented in Ultralytics library can be seen on Listing 3.1.The main purpose of the *Focus* layer is to reduce layers, reduce parameters, reduce FLOPS, reduce CUDA memory, increase forward and backward speed while minimally impacting mAP, and it was introduced by Glenn Jocher in Ultralytics library. It replaces the first three layers of YOLOv3. *Conv* represents a convolutional layer, whereas *BottleneckCSP* and *SPP* represent a CSP connection and SPP as introduced in subsection 3.3.1. *nn.Upsample* upsamples a given multi-channel 1D (temporal), 2D (spatial) or 3D (volumetric) data. *Concat* concatenates the given sequence of tensors in the given dimension. All tensors must either have the same shape (except in the concatenating dimension) or be empty.

Listing 3.1: Backbone and Head of VOLOv5 as implemented in [ult,]

```
1 # YOLOv5 backbone
2 backbone:
```

```
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]], # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]], # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]], # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]], # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]], # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]], # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]], # cat head P4
   [-1, 3, BottleneckCSP, [512, False]], # 20 (P4/16-medium)
```

```
31
32      [-1, 1, Conv, [512, 3, 2]],
33      [[-1, 10], 1, Concat, [1]], # cat head P5
34      [-1, 3, BottleneckCSP, [1024, False]], # 23 (P5/32-large)
35
36      [[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
37    ]
```

3.4 Tools

3.4.1 Software

The experiments were done in the free version of Google Colab [col,], using the GPU available. The quantity of GPU in the free version fluctuates and is therefore not strictly defined. The paid version of Google Colab is not available in most European countries at the time of writing this work. In addition, PyCharm was of use in order to write certain snippets of code [pyc,]. The following Python libraries and packets were used:

- **PyTorch** is an open source ML framework that accelerates the path from research prototyping to production deployment. Thanks to a strong GPU support it is used for applications such as Computer Vision and natural language processing and many DL software are built on top of it[PyT,].
- **Tensorflow** is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. [tf,]
- **Keras** is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. [ker,]

- **PIL** is a free and open-source library that is useful for opening, modifying, displaying and saving images since it supports many different file formats. It includes various standard functions for images, such as geometric transformations, per-pixel manipulations, distructive transformations (blurring, smoothing, . . .), adding text or symbols to images and much more [PIL,].

- **OpenCV** is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.[ope,]

- **Ultralytics** is a library implemented by Glenn Jocher for easy and user-friendly deployment of YOLOv5 model [ult,].

- **PyYAML** is a YAML parser and emitter for Python [PyY,].

- **Matplotlib** is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible [Mat,].

- **OS** provides a portable way of using operating system dependent functionality [os,].

- **Pandas** is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language [Pan,].

- **LabelMe** was used in order to annotate the images [lab,].

3.4.2 Hardware

The laptop used in order to create the work is Dell Vostro 14 5401. The cameras used for capturing photos used in constructing the dataset are Olympus E-M10MarkII [oly,] and Sony cameras [son,].

Chapter 4

Development

This chapter describes the dataset used in training of the models. The original dataset is discussed and described, and so is the process of annotating the images in the dataset. The chapter proceeds to describe the changes that were done upon images and labels, as well as the reasoning for these changes. A final output dataset is described, including both the images and labels. Following that, data post-processing is described in detail, including some problems faced, as well as their solutions.

4.1 Images

The initial dataset was comprised of 19 high resolution (3456*4608) images of microplastics particles of five different classes (categories) on a white background. The classes in question are fragments, pellets, lines, tar, and organic. The information on each of these classes can be found in Table 4.1 and on Figure 2.17.

Some images display shadows whereas the others do not, which, according to the documentation is a desirable feature of the image data [bes,]. Images were taken with Sony digital camera[son,] and Olympus E-M10MarkII digital camera [oly,]. There are some differences between the photos taken by the two cameras with respect to illumination. This was done on purpose in order to verify that the architecture did not present significant variations in

Table 4.1: Classification of Microplastics Particles

Class	**Description**
Fragment	A microplastics particle originating form fragmentation of larger pieces
Pellet	Particles originating from intentional (primary) production. These particles have been subject to a large period of erosion, and this is why they usually display a rounded shape. The color range they are taking on is usually between beige and dark brown.
Line	Small nylon segments originating from fishing nets or fishing lines. They are therefore long but thin elements that are usually greenish or bluish in color.
Tar	It is not a microplastic particle, but it is a residue that researchers want to take into account. It is dark brown or black particle originating from a viscous liquid of hydrocarbons and free carbon, obtained from a wide variety of organic materials through destructive distillation.
Organic	This class is made up of elements that have an organic origin, such as bones, branches, algae or small shells. They are not the object of study but it is interesting to train the models to recognize them for what they are in order to discard them.

spite of changes in the characteristics of the cameras, such as the type of digital camera or the illumination.

An important note to remark is the fact that the photos taken with the Sony camera initially the dimensions of 4000*6000, and were later on reduced to the size of 3456*4608, in order to make sure the net is always fed the images of the same size.

The described dataset needed further processing in order to obtain satisfying training results with it. The reason for that is twofold, and described in section 4.3 and section 4.4. The solution is presented in section 4.5.

4.2 Annotations

The images belonging to the initial dataset comprising 19 large, high resolution images, were labeled using the LabelMe annotation tool, using polygonal shapes to delineate various microplastics fragments in the images. The labels were saved in the JSON format particular to the LabelMe software. Note that in addition to labeling microplastics particles, the particles of organic matter and tar were labeled too. Teaching the model to distinguish between the microplastics particles, organic matter and tar is a crucial part of the task, due to the possible drop of accuracy that might otherwise arise.

Next challenge was transforming the LabelMe JSON labels to YOLO type .txt files. Namely, each of the training, validation and test datasets required for the execution of YOLOv5 by the Ultralytics needed to be distributed in two folders: one for images, and one for text files releavnt to labels. Each image should have a corresponding text file with the same exact name as the image, the only difference being the extension, which in .jpg in the case of the image and .txt in the case of the label file. For YOLOv5 to be successfully executed, the label file should be comprised of as many lines as there are objects in the image, the first element in the line being a number representing the class, and the following four elements representing the normalized x and y coordinate of the center point, and the width and the height of the box, respectively.

Figure 4.1: Box coordinates in the normalized xywh format. Source: [ult,]

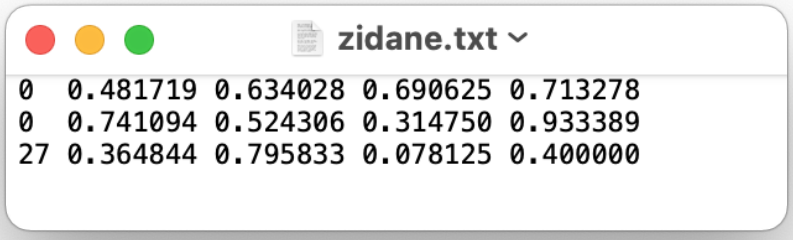

Figure 4.2: Annotation file example. Source: [ult,]

The label files were successfully converted from JSON to YOLO using the 'Conversion.py' file, a script written specifically for the purpose of this work. The file converts the polygon labels to square labels and converts the LabelMe JSON format to the YOLO .txt format.

The files were transferred from the personal computer to the appropriate Google Drive folders, and the data was thus ready for the next step in the process.

4.3 Problem 1: Image Data Quantity

The original number of images was only 19, and even though YOLOv5 model is a pretrained model and thus requires less input images than models requiring training from scratch, the training results were unsatisfactory, as can be observed in Figure 6.1. These results were achieved with default hyperparameter settings for low augmentation training [Hyp,] for yolov5 model, choosing yolov5s6 as weights, and setting the image size to the original one, namely 4608. The result of the detection can be observed on Figure 4.3.The image clearly shows that very few particles of microplastics were indeed detected, and those that were detected were misclassified. Namely, all the detected particles were classified as 'tar'.

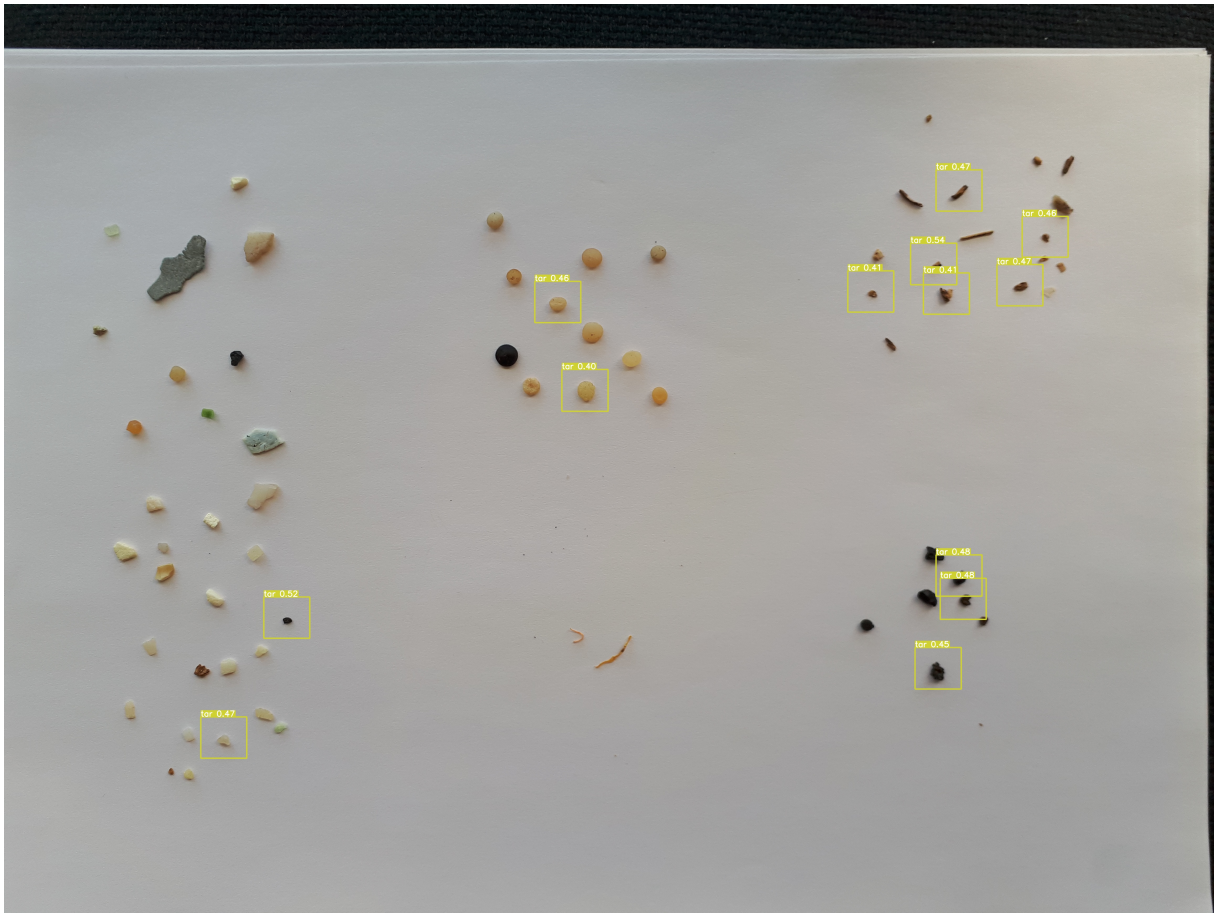

Figure 4.3: Results on the test image after training with the initial dataset with confidence threshold set to 0.1

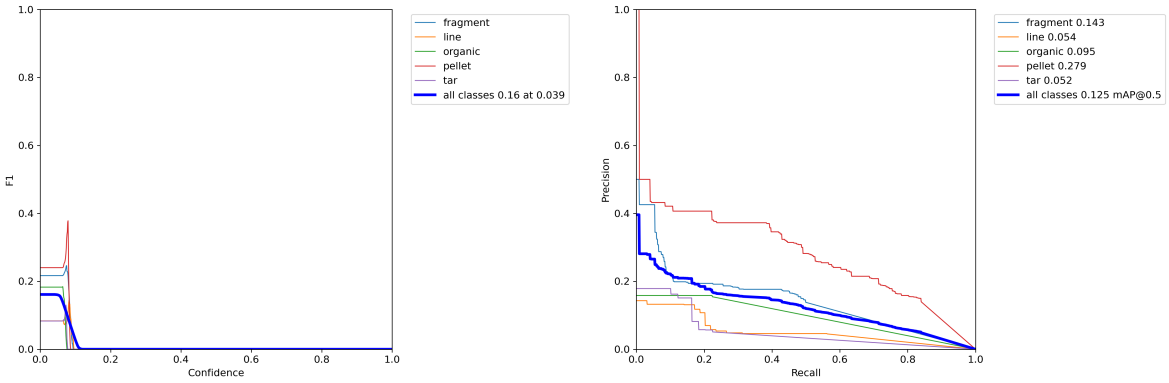

Figure 4.4: F1 (left) and Precision and Recall (right) curves after the initial training of YOLOv5s6 on the images of size 3456*4608 pixels with default hyperparameter settings

4.4 Problem 2: Image Data Size

In addition to the problem of small dataset size, another issue was the size of the images. Namely, the problem at hand has previously been discussed in the literature as the problem of object detection on high resolution images with small objects. Simply enlarging the resolution of input image in the neural network will cause more problems, such as that, it aggravates the large variant of object scale and introduces unbearable computation cost. [Liu et al., 2020] Since the YOLOv5s6 weights were initially obtained on the dataset of images of size 1280*1280 pixels, and the documentation suggested training the model on the images of similar size, it was therefore necessary to crop the images into smaller ones.

4.5 Solution: Mosaic Image Data Enhancement

This method, indeed, solves both the problems the original dataset was presenting: it both increases the data quantity as well as it decreases the image size.

4.5.1 Simple Mosaic

The first version of this system would consist of a simple division of the original image into squares, as shown in Figure 4.5. Each cell of such a mosaic would have a size approximately equal to the size required by the net. In this way, the detection of the elements in the complete image will consist of the sum of the successive detections in each of the generated squares.

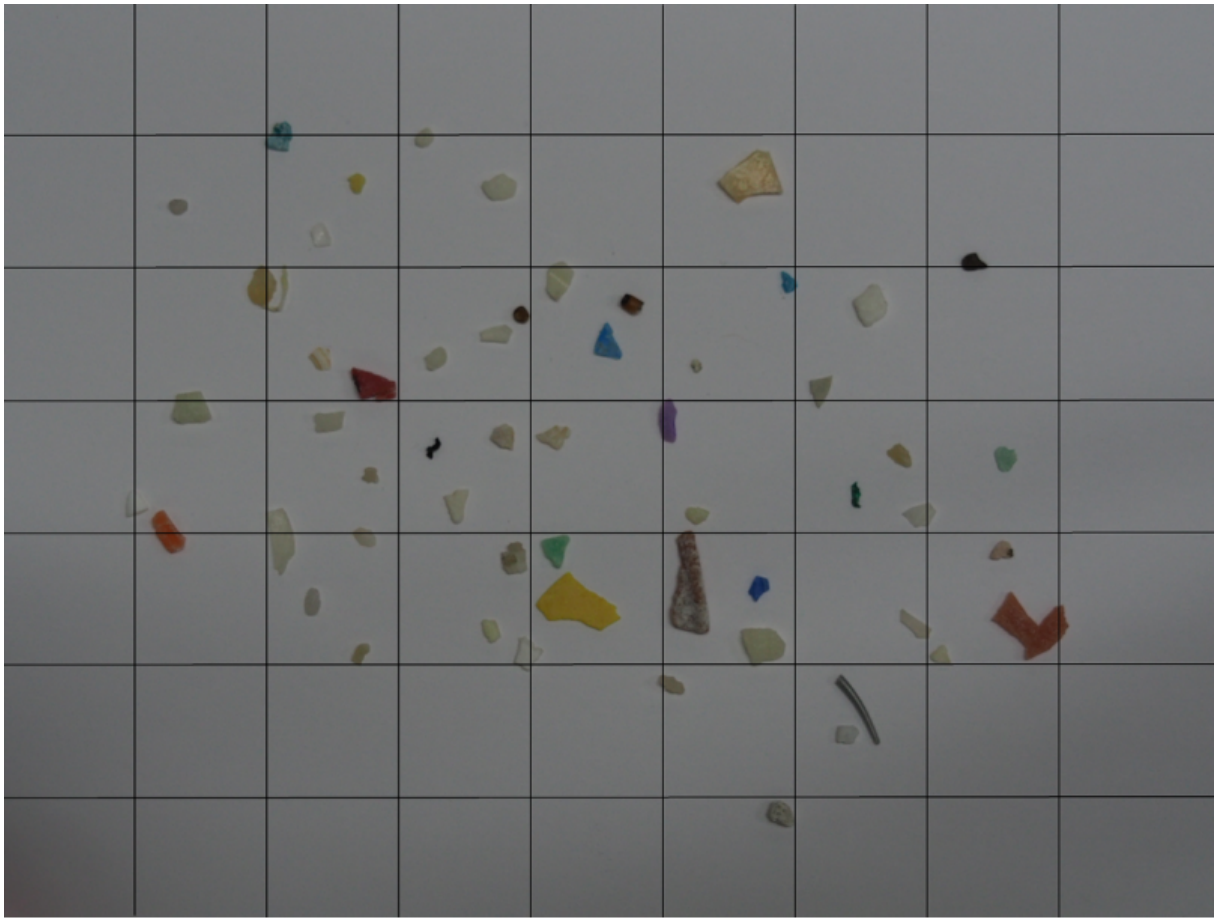

Figure 4.5: Simple Mosaic method done on a sample image. Source: [Cerezo, 2020]

The chosen size would be 1152 pixels in height and width, which would be the obtainable size most similar to the size of images on which the net YOLOv5s6 was trained (1280). By obtainable, it is meant that the chosen image size needs to be a common factor of width(4608) and height(3456) of the original photo.

In addition to decreasing the images to the size similar to the size of images on which the net was trained without losing important information, this solution also generates a larger number of images.

Despite the apparent suitability of this methodology, it presents a major problem to be considered. Namely, an object which just so happens to be on the grid, and thus split in two images has two possibilities of detection (or a lack of it): the object might be detected multiple times, or it simply might not even be detected because the model does not have a

complete "view" of it.

As shown in Figure 4.6, the blue particle marked with a red circle falls right in the middle of the partition grid between two cells. Thus, two images are generated in which in each one there is a part of the blue particle. This, as is logical, can lead to the duplicate detection of the blue particle or it might happen that the network does not even detect it because it is not visible in its entirety to the network. This is a problem during testing, when the full-resolution image is inserted into the system and later cropped up into smaller images.

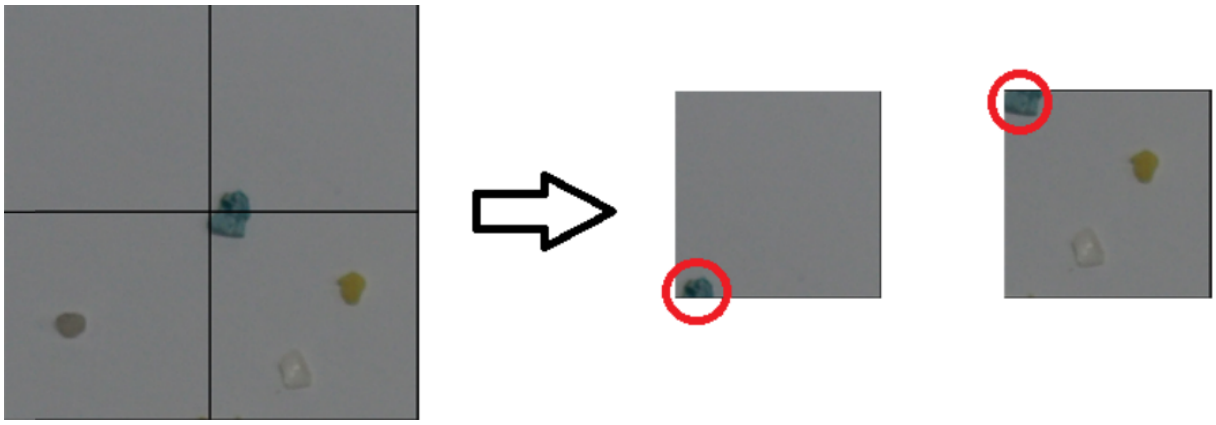

Figure 4.6: Problematics of the simple mosaic method Source: [Cerezo, 2020]

This methodology, even though it might work well for classifying complete pieces of microplastics on a given image, is not appropriate for counting the particles of microplastics on the original, full sized input image.

4.5.2 Mosaic with Overlap

In order to address this problem, a system with overlaps was chosen as a solution. In other words, the contiguous cells of the mosaic will overlap, so that on multiple images there will be regions coming from the same area of the original image. For this purpose, the photograph was divided into a grid of smaller squares. These squares are then combined to generate a fragment of a certain size. As shown in Figure 4.7 when the overlap is performed, two images are generated that have an area where they overlap. Likewise, it can be observed that

in the second image the particle appears almost in its entirety, but in the first one, it is not. practically in its entirety, something that would not occur in the system without overlapping.

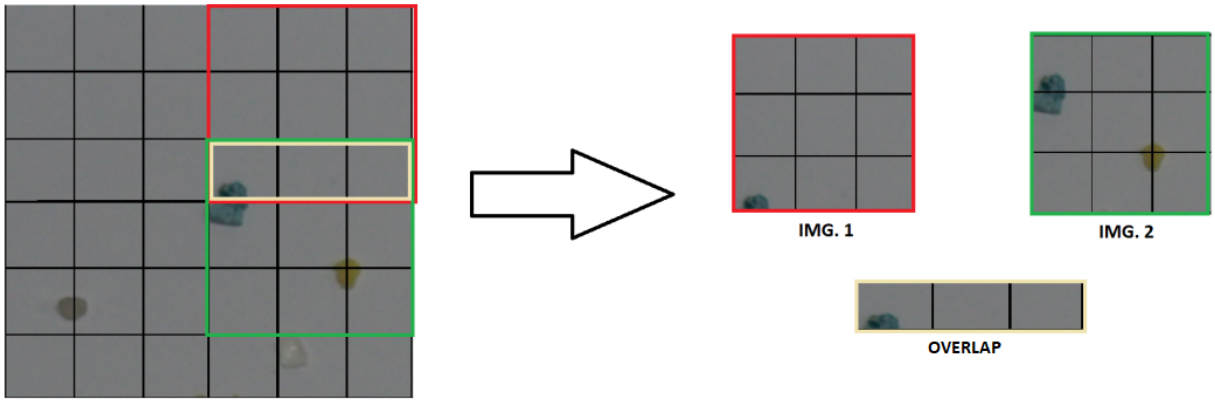

Figure 4.7: overlap Source: [Cerezo, 2020]

This method clearly results in the generation of an even larger number of images than the simple mosaic method described in subsection 4.5.1, which increases the dataset size, as well as leads to an increase in the detection time of the complete image. However, this is a cost that must be assumed since without this solution, the effectiveness of the system would be drastically reduced.

Cropping Images

The algorithm used to accomplish this task is implemented in the code shown in the Listing 4.1. As can be observed, the name of the output images is generated so that it keeps the information of a) original image, b) top left coordinate of the rectangle: 20220516_113208.jpg_grid_2304_3456.jpg. This is going to be useful in the latter stages of the process, when it's time to recompose the detection coordinates on a full sized input image.

Listing 4.1: Algorithm for generating cropped images, storing the information on the positioning in the original image in the image name

```
1 def split_image(src_dir, filename, out_dir):
2     grid_image = open_image(os.path.join(src_dir, filename))
```

```
3    grid_width, grid_height = grid_image.size
4
5    img_sz=1152 #desired image size
6    overlap = img_sz/3
7
8    for h in range(1, int(grid_height/overlap) - 1):
9        for w in range(1, int(grid_width/overlap) - 1):
10           left = overlap * (w - 1)
11           right = left + (overlap * 3)
12           top = overlap * (h - 1)
13           bot = top + (overlap * 3)
14           crop_img = grid_image.crop((left, top, right, bot))
15           crop_img.save(os.path.join(out_dir, filename + "_grid_" + str(top) + "_" + str(left)
                 + ".jpg"))
```

Firstly, the dimensions of the original input image are obtained in line 3 of Listing 4.1, after which both the width and the height of the input image are divided by overlap value, and the grid iteration begins in a nested for loop on lines 8-15. In the preprocessing architecture that has been elaborated, the overlap value corresponds to a third of 1152, so 384 ppi. On each step of the iteration, we are moving the pointer by one third of the output image and creating a new image starting at exactly that coordinate as the top left corner. When all the iterations have been done in the width, we lower in the height axis by one overlap value, and begin iterating over new images in that row.

The experiments were done with two different potential sizes of the generated images: 576 and 1152. YOLOv5s6 model with default settings was deployed on a training, validation and test set, first of size 576 and then size 1152. The chosen image size was 1152, and the experiment is documented in section 5.1.

Cropping Labels

Once the images have been cropped, it was time to create appropriate label files for each of the newly cropped images. This process is twofold: we need to divide the lines of label files corresponding to the full sized original images in distinct files corresponding to the new, cropped images. Next, we need to normalize these lines with label coordinates to the dimensions of the newly created images.

First important step was to iterate over the lines of the original label file and check if the line at hand is a label relevant to the cropped image at hand. A crucial information here was stored in the image name during the image cropping step described in section 4.5.2. The code snippet for this step can be observed in Listing 4.2. The most important part of this snippet is the condition used for checking whether the row in question is located in the part of the original image corresponding to the cropped image at hand, which can be seen on line 20 of Listing 4.2. This is done by making sure that every angle of the rectangle is within the given coordinates in the original image.

Listing 4.2: Code snippet for checking if a label line from the original image is present in the cropped image

```
1  def isInTheImage(label_dir, row, overlap):
2      n = label_dir.find('grid_')
3      pcords = label_dir[(n+5):].strip('.jpg').split('_')
4
5      #boundaries of the image in pixels
6      top = int(pcords[0])
7      left = int(pcords[1])
8      right = left + (overlap * 3)
9      bottom = top + (overlap * 3)
10
11     #bounding box (label) coordinates
```

```
12      bbcords=row.split()[1:]
13
14      #list of bounding box coordinates
15      lbbcords=[]
16      for x in bbcords:
17        lbbcords.append(float(x))
18
19      #check if the bounding box from the original image is present it the new one
20      if 3456*(lbbcords[1]+ 0.5*lbbcords[3])>top and 3456*(lbbcords[1]- 0.5*lbbcords[3])<
            bottom and 4608*((lbbcords[2]/2+lbbcords[0]))<right and 4608*((lbbcords[0]-
            lbbcords[2]/2))>left:
21        return True
22      else:
23        return False
```

Once the row that belongs to the cropped image has been identified, the next step is to convert the coordinates belonging to the input label, into coordinates belonging to the output label. As was described previously in chapter section 4.2 and can be observed in Figure 4.2 and Figure 4.1, the lines in the label file are comprised of a string separated by whitespace character comprised of 5 values: class value, x center, y center width and height of the bounding box, as described in section 4.2. These coordinates are normalized with respect to the relevant size of the image. Thus, these values needed to be "re-normalized" to the new size of the image, since the relevant particle of microplastics is likely located in a different place in the new image.

The function written with this purpose can be found in Listing 4.3. The width and the height of the bounding box are multiplied by the fraction of width (height) over the size of the output image (lines 15 and 16). The center of the box coordinates, X and Y were obtained in the following way: The original x coordinate was multiplied by the width of the original image to obtain the actual pixel position of this value (instead of a normalized value). From this

number, the left coordinate and the new width coordinate have been subtracted in order to obtain the pixel coordinate for X in the new image. To normalize the value again, the number was divided by the size of the output image. The Y coordinate was obtained in the same way, the difference being the height was used instead of the width, and the top value and the height value was used instead of the left value and the width. This calculation can be found on lines 17 and 18 of Listing 4.3. In lines 20-26 we made sure that no value assigned to X or Y lays outside of [0,1] range, whereas in lines 29-36 we made sure that no bounding box exceeds the boundaries of the image.

Listing 4.3: Code snippet for changing the normalization function applied to coordinates in the label file into a new normalization function in our new output label file

```
1  def convert_coordinates(line,image, overlap, width, height):
2      #get top left corner coordinates
3      idx=image.find('grid_')
4      pcords=image[(idx+5):].strip('.jpg').split('_')
5
6      #boundaries of the image in pixels
7      top=int(pcords[0])
8      left=int(pcords[1])
9      right= left + (overlap * 3)
10     bottom= top + (overlap * 3)
11
12     #original image coordinates converted and normalized to the cropped image
13     line=line.split('␣')
14     w=float(line[3])*(width/3*overlap)
15     h=float(line[4])*(height/3*overlap)
16     x=(float(line[1])*4608-left-w)/3*overlap
17     y=(float(line[2])*3456-top-h)/3*overlap
18     if x<0:
```

```
19          x=0
20      if x>1:
21          x=1
22      if y<0:
23          y=0
24      if y>1:
25          y=1
26
27      #in case the bounding box is outside the picture
28      if x-w/2<0:
29          w=2*x
30      if y-h/2<0:
31          h=2*y
32      if w/2+x>1:
33          w=2*(1-x)
34      if h/2+y>1:
35          h=2*(1-y)
36
37
38      newline=line[0]+' '+str(x)+' '+str(y)+' '+str(w)+' '+str(h)
39      return newline
```

Finally, the main function was written (Listing 4.4), uniting the two previously defined functions, and writing an annotation file relevant to each of the output images of the system. In Listing 4.4 we iterate over the list of points in the original image that are going to become top left corners of the newly generated images in lines 5-10. We then create a new label file for every newly generated image in line 13, we open the newly genrated image in line 16, and we read the label file belonging to the original input image in line 17. For each of the lines (bounding boxes) in that file, we check if it is present in the newly generated image in line 21,

and if so we "re-normalize" the bounding box values to be relevant to the newly generated image in line 22, and finally we write this line into the new label file in line 23.

Listing 4.4: Code snippet for creating new label files out of the original label files

```
1  def crop_labels(overlap, grid_image_path, og_lbl_path, outpath):
2      grid_image = Image.open(grid_image_path)
3      grid_width, grid_height = grid_image.size
4
5      for h in range(1, int(grid_height/overlap) - 1):
6          for w in range(1, int(grid_width/overlap) - 1):
7              left = overlap * (w - 1)
8              right = left + (overlap * 3)
9              top = overlap * (h - 1)
10             bot = top + (overlap * 3)
11
12             for filename in label_name_list:
13                 original= open(og_lbl_path+'/'+filename, "r")
14                 img_path=os.path.join(outpath, filename.strip('.txt') + ".jpg_grid_" + str(
                       top) + "_" + str(left) + ".jpg")
15                 new_path=os.path.join(outpath, filename.strip('.txt') + ".jpg_grid_" + str(
                       top) + "_" + str(left) + ".txt")
16                 new = open(new_path, "a+")
17                 lines = original.read().split('\n')
18
19                 for line in lines:
20                     if line!='':
21                         if isInTheImage(img_path, line):
22                             newline = convert_coordinates(line, grid_image_path)
23                             new.write(newline +'\n')
```

```
24                  print('Saving␣{}␣to␣{}'.format(newline, outpath)
```

4.6 Final Dataset

The final dataset contained 1330 output images of size 1152*1152 ppi. Therefore, each of the 19 images belonging to the original dataset generated 70 new images of size 1152. This clearly implies a significant overlap between the output images, and most of the particles in the original image are going to appear in numerous output images. For this reason, it was crucial to make sure that the same particle does not show up in more than one of the training, validation and test sets. In the following subsection, solution to this issue is described.

4.6.1 Training, Validation and Test Split

Once both the labels and the images have been cropped to the new size, and before executing any machine learning model on the data, it was necessary to split it into training, validation and test set. In order to make sure the split was comparable and fair, it was made sure that the validation and test set only contain cropped images and labels belonging to 2-3 original images, each.

Listing 4.5: Code snippet for dividing the dataset into training, validation and test set

```
1  def data_split(oi, ol, vali, vall, testi, testl):
2
3      #removing the '.jpg' ending
4      l=[f[:-4] for f in os.listdir(oi)]
5
6      for x in l:
7          #move all the and cropped images and labels belonging to these two original images
               into test folder
8          if '20220516_113208' in x or '20220516_113718' in x:
```

```
9               os.rename(oi+'/'+x+'.jpg', testi+'/'+x+'.jpg')
10              os.rename(ol+'/'+x+'.txt', testl+'/'+x+'.txt')
11
12          #do the same for the validation folder, this time with 3 images
13      if '20220516_114747' in x or 'DSC00010'in x or 'line–04–olympus–10–01–2020' in x:
14              os.rename(oi+'/'+x+'.jpg', vali+'/'+x+'.jpg')
15              os.rename(ol+'/'+x+'.txt', vall+'/'+x+'.txt')
```

The function can be found on the Listing 4.5. Initially, all the 1330 images and labels were located in the same directory, 'oi' and 'ol' for images and labels, respectively. Then, the images containing in it the name of one of the two original images were moved to the test folder, as well as the corresponding labels (lines 6-10). The same was done with the validation set: output images and labels corresponding to one of the three original images were moved to the validation folder (lines 13-15). This way the initial, undivided dataset of images and labels became the training folder(s). Finally, the training set contained 980 images (originating from 14 full-sized input images), validation set 210 images (originating from three full sized images) and the test set 140 images (originating from two full sized images).

4.7 Data Post-Processing

As was specified at the beginning of the work, the objective here is to create a software which counts and classifies particles from the original input image, which as was specified in the beginning, is the size 3456*4608 ppi. However, since the training images are smaller in size, a crucial step is to use the information collected during object detection on cropped up small images with overlap, to obtain a number of particles and classes of objects on the large input image. In the sections to come, all the circumstances that had to be taken into account are going to be elaborated.

It should be kept in mind that, after the detection of each original input image, the net-

work returns another one in which the elements detected in the image are marked with colored rectangles, each color corresponding to a particular class. This image has the same name as the image provided, but it's located in the '.../results' folder. In addition, the system provides us with a single CSV file in which all the detections made are annotated, belonging to a certain original sized input test image. This file indicates for each detection, the coordinates extracted from the name of the original image in which this prediction was made, the position and size of the box that delimits the detected element, the class to which this object belongs, and the confidence of the network expressed in decimal number from 0 to 1.

4.7.1 Duplicate Elimination

The overlap system ensures that all the elements of the original image are completely or almost completely present in at least one of the images provided to the detector. However, this also results in the same object appearing in multiple images if it is located in an overlapping area, as shown in Figure 4.8

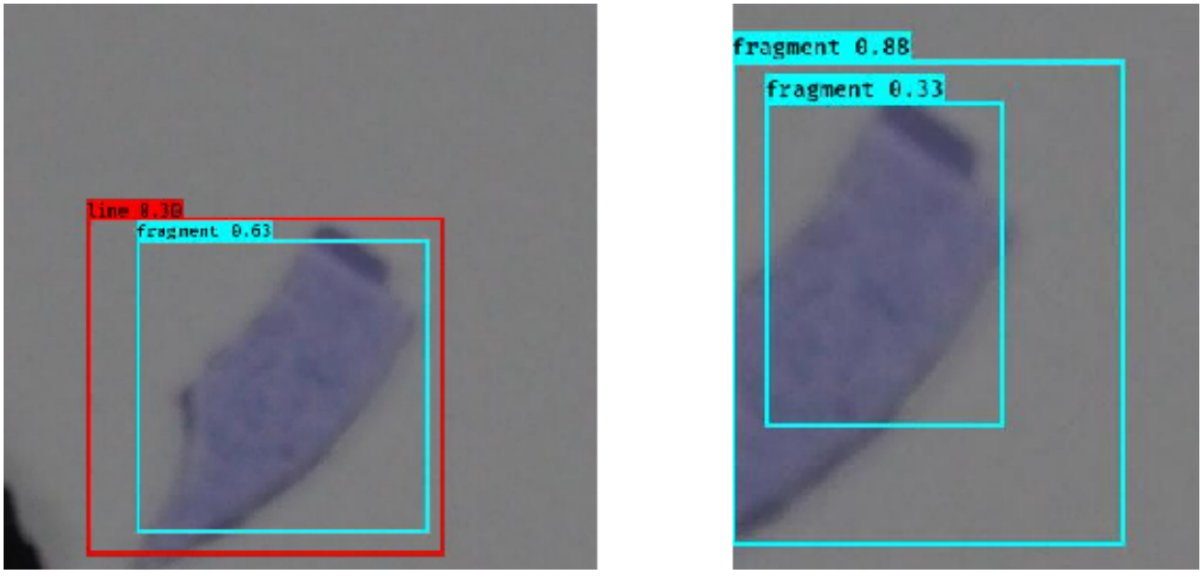

Figure 4.8: Two possible cases of multiple detections of the same particle. Source [Cerezo, 2020]

Therefore, when recombining all the fractions of images it is essential that the repeated

detections of the same particle are eliminated. Only a single detection should be kept for each particle. To solve this problem, a detection system for overlapping detection boxes was developed. This system generates all the bounding boxes of an image and saves them in a provisional list. It then loops through that list, comparing each rectangle of the list with the rest. In each comparison, the intersection over union between the two rectangles is caluclated. Or, in other words, the percentage of the area in which each two rectangles overlap is obtained. In order to obtain the intersection over union for each pair of boxes, first we have to calculate the area of intersection of the two rectangles, as well as the area of each one separately. To perform this operation it is convenient to simplify the representation of the rectangles in four values, left side l_n , right side r$_n$, top t_n and bottom b_n. In addition, we know that the area of intersection between these two figures is a rectangle in itself. Therefore, we have to find the width and height to calculate its area. The problem statement can be observed on Figure 4.9.

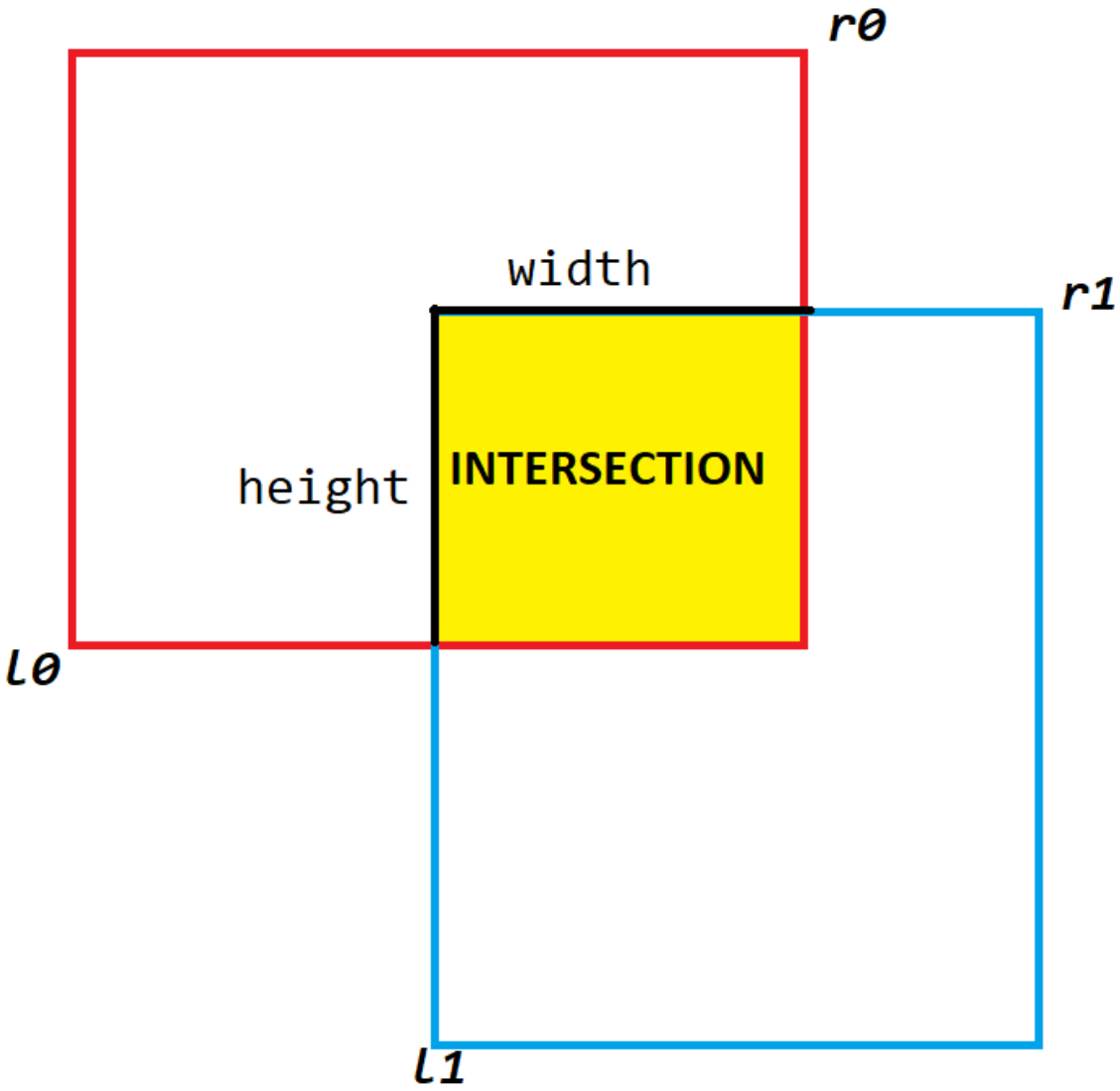

Figure 4.9: visualization of the intersection area

The formulas used in order to find width and height are quoted on Equation 4.1 and Equation 4.2. Namely, in order to find the width of the intersection area, the maximum of the lower left corners coordinates was subtracted from the minimum of the two squares upper right corners.

$$width_{intersection} = min(r_0, r_1) - max(l_0, l_1). \tag{4.1}$$

The process and reasoning for calculating the height are analogous:

$$height_{intersection} = min(t_0, t_1) - max(b_0, b_1). \tag{4.2}$$

Once we have obtained the height and width, we multiply them to obtain the area. It should be noted that if the rectangles do not overlap, the resulting area may be negative or zero, so that needs to be accounted for in the calculations.

$$area_{intersection} = width_{intersection} * height_{intersection}. \tag{4.3}$$

Once the intersection area is obtained, we need to find the percentage of this area with respect to the sum of the areas of both the rectangles, i.e. the intersection over union (IoU). as follows:

$$IoU = \frac{area_{intersection}}{area_0 + area_1 - area_{intersection}}. \tag{4.4}$$

In case the calculated percentage exceeds a certain threshold, set at 25%, it is understood to be a duplicate detection. Among the two detections, the one with the highest confidence value is kept and the other one is discarded. It is not expected that there will be overlapping samples in the photos to be taken, but rather that the particles will be arranged throughout the photo.

4.7.2 Repositioning Bounding Boxes

The following challenge to be addressed was placing all of the bounding boxes from cropped up images back into original, full sized input image.

For each original image name, the croped up generated images' labels were retrieved and the bounding boxes coordinates were inserted into a CSV file together with the coordinates extracted, label, and confidence score. In this way, after passing through the network, it was possible to know from which generated image each detection came, as well as its position, confidence and detected label.

Therefore, in order to place each of the bounding boxes on the original full sized image, it was only necessary to iterate over the CSV file named after the image in question. The coordinates of the generated image were later used as the h_offset (height offset) and w_offset

(width offset) value and were crucial in placing the bounding box on the original image. This can be observed in the Listing 4.6.

The measurements given in the CSV are relevant and normalized to the current size of the image, so in order to reposition these bounding boxes onto the original image, the information on the relative location of a particle was leveraged in order to place the bounding boxes back on the original image. This was done by obtaining the offset in terms of width and height from the cropped image name (line 2 on Listing 4.6), and then calculating the coordinates (lines 3-7).

Listing 4.6: Normalizing the bounding box coordinates to the original full sized image's dimensions

```
1 def generate_rectangles(csv_row): #positions rectangles on a full sized image
2     [h_offset, w_offset]= csv_row["grid_coords"].split(",␣")
3     xmin = int(w_offset) + int(csv_row["xmin"])
4     xmax = int(w_offset) + int(csv_row["xmax"])
5
6     ymin = int(h_offset) + int(csv_row["ymin"])
7     ymax = int(h_offset) + int(csv_row["ymax"])
8
9     return Rectangle(xmin, ymin, xmax, ymax, int(csv_row["class"]), float(csv_row["
          confidence"]))
```

Chapter 5

Experimental Design

Several experiments were done in order to obtain the best settings for the final model.
The experimental design is the following: using a YOLOv5s6 model with default settings, it was first decided on the dataset which is more informative for the model; a dataset of 6688 images of size 576 pixels and a dataset od 1330 images of size 1152 pixels were compared. Following that, four experiments were done with different transfer learning and data augmentation settings.

5.1 Image and Dataset Size

The experiments were done with two different datasets: a dataset comprising images of resolution 576 containing 6688 images, and a dataset comprising images of resolution 1152 with 1330 images. YOLOv5s6 model with default settings was deployed on a training, validation and test set, first of size 576 and then size 1152. These results were achieved with low augmentation hyperparameter settings training [Hyp,], choosing 'yolov5s6' as pre-trained weights.
The following experiments included experimenting with distinct data augmentation settings as well as training only the output layer vs training the whole net.

5.2 Data Augmentation

Data Augmentation is a technique used to generate synthetic data from the real dataset through various transformation and invariant to noise techniques by modifying different parameters. In imaging, the increase of such data is used to reduce overfitting during the training phase, and also to minimize the existing imbalance in some classes when training machine learning systems, i.e. when the different classes used for training an algorithm do not have an equal or similar number of examples [Shijie et al., 2017]. For image classification models, data can be augmented using Deep Learning based approaches which require the training of models to generate the new synthetic samples, namely Generative Adversarial Networks (GAN) [Seibold et al., 2022], Autoencoders (AE) [DeVries and Taylor, 2017], or Variational Autoencoders (VAE) [Norouzi et al., 2020] among others. On the other hand, there are also augmentation methods which do not require the training of these models. They are based on transformations such as geometric, photometric and noise invariant alterations [Zoph et al., 2020] [Delgado de Santos et al., 2020].

- Geometric transformations alter the geometry of the image to ensure that the model is invariant to the position and orientation of the image. Rotation, flipping, shifting, scaling and cropping would be good examples of these, which can be seen in Figure 5.1.

- Photometric transformations modify the color channels in order to make the model invariant to the color and illumination of the image. Some examples would be changing the brightness, contrast, saturation or hue of an image, as shown in Figure 5.2.

- Noise invariant transformations are used in order to help the CNNs learn noise-robust features. Gaussian blur, sharpness or edge detection which are shown in Figure 5.3 are examples of this method.

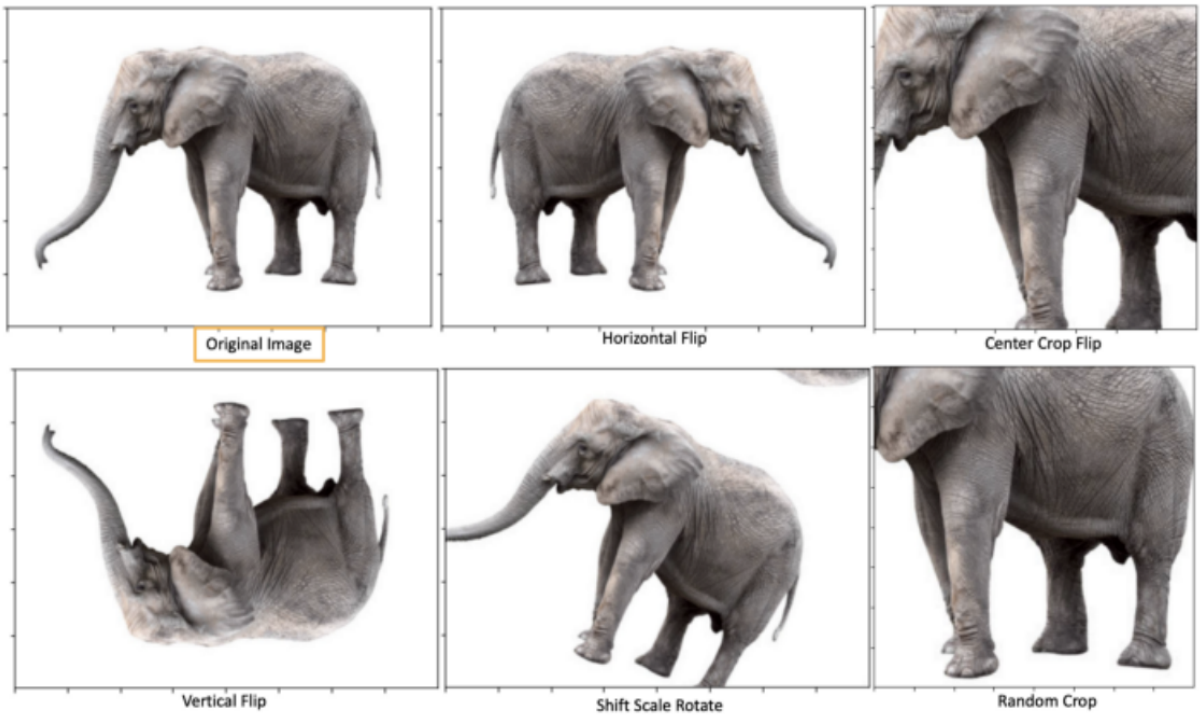

Figure 5.1: Geometric Transformation. Source [Algarabel, 2022]

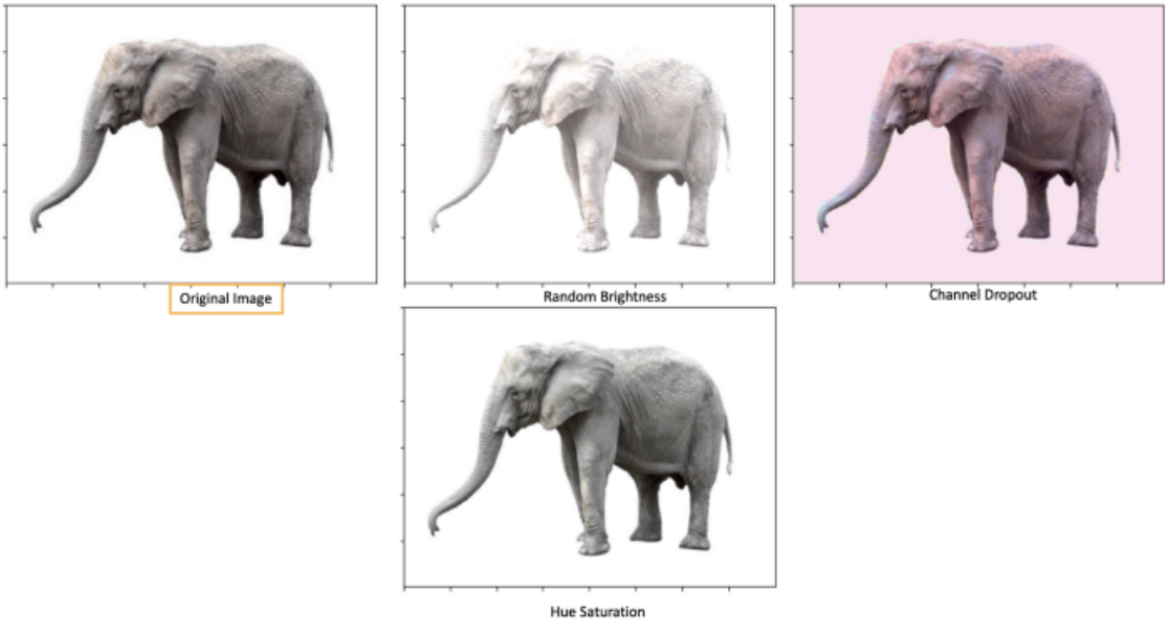

Figure 5.2: Photometric Transformation. Source [Algarabel, 2022]

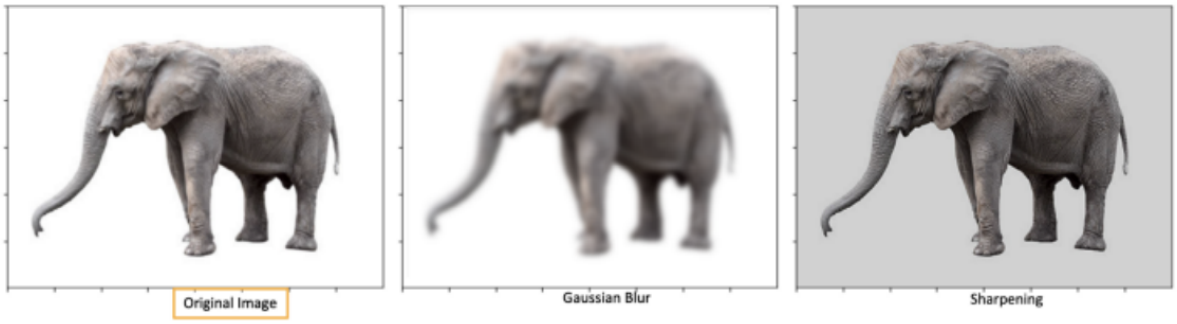

Figure 5.3: Noise Invariant Transformation. Source [Algarabel, 2022]

In this work, the experiments are going to be conducted with Low and High Data Augmentation (DA) settings as provided in the Ultralytics library. In the following chapters, low and high DA settings are sometimes referred to as no DA and DA. Namely, the settings can be observed in Table 5.1. For the sake of brevity of this report, not all of the transformations can not be individually addressed and explained here, and the most important ones have already been mentioned.

Table 5.1: High (left) and Low (right) DA Hyperparameter Settings for the YOLOv5 model as provided by [ult,]

Hyperparameter	**Low Setting**	**High Setting**	**Interpretation**
lr0	0.01	0.01	initial learning rate (SGD=1E-2, Adam=1E-3)
lrf	0.01	0.1	final OneCycleLR learning rate (lr0 * lrf)
momentum	0.937	0.937	SGD momentum/Adam beta1
weight_decay	0.0005	0.0005	optimizer weight decay 5e-4
warmup_epochs	3.0	3.0	warmup epochs (fractions ok)
warmup_momentum	0.8	0.8	warmup initial momentum
warmup_bias_lr	0.1	0.1	warmup initial bias lr
box	0.05	0.05	box loss gain
cls	0.5	0.3	cls loss gain
cls_pw	1.0	1.0	cls BCELoss positive_weight
obj	1.0	0.7	obj loss gain (scale with pixels)
obj_pw	1.0	1.0	obj BCELoss positive_weight
iou_t	0.20	0.20	IoU training threshold
anchor_t	4.0	4.0	anchor-multiple threshold
fl_gamma	0.0	0.0	focal loss gamma (efficientDet default gamma=1.5)
hsv_h	0.015	0.015	image HSV-Hue augmentation (fraction)
hsv_s	0.7	0.7	image HSV-Saturation augmentation (fraction)
hsv_v	0.4	0.4	image HSV-Value augmentation (fraction)
degrees	0.0	0.0	image rotation (+/- deg)
translate	0.1	0.1	image translation (+/- fraction)
scale	0.5	0.9	image scale (+/- gain)
shear	0.0	0.0	image shear (+/- deg)
perspective	0.0	0.0	image perspective (+/- fraction), range 0-0.001
flipud	0.0	0.0	image flip up-down (probability)
fliplr	0.5	0.5	image flip left-right (probability)
mosaic	1.0	1.0	image mosaic (probability)
mixup	0.0	0.1	image mixup (probability)
copy_paste	0.0	0.1	segment copy-paste (probability)

5.3 Transfer Learning

Humans are able to take information from a previously learned task and use it beneficially to learn a related task. This would be the idea behind the Transfer Learning technique.

The definition of transfer learning is given in terms of domains and tasks. A domain D consists of a feature space $\mathscr{X}$ and a marginal probability distribution P(X) where $X = \{x_1, ..., x_n\} \in \mathscr{X}$. Given a specific domain, D=$\{\mathscr{X}$, P(X)$\}$, a task consists of two components: a label space $\mathscr{Y}$ and an objective predictive function $f : \mathscr{X} \rightarrow \mathscr{Y}$. The function f is used to predict the corresponding label f(x) of a new instance x. This task, denoted by $\mathscr{T} = \{\mathscr{Y}, f(x)\}$, is learned from the training data consisting of pairs $\{x_i, y_i\}$, where $x_i \in \mathscr{X}$ and $y_i \in \mathscr{Y}$.[Lin and Jung, 2017]

Given a source domain $\mathscr{D}_S$ and learning task $\mathscr{T}_S$, a target domain $\mathscr{D}_T$ and learning task $\mathscr{T}_T$, where $\mathscr{D}_S \neq \mathscr{D}_T$, or $\mathscr{T}_S \neq \mathscr{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathscr{D}_T$ using the knowledge in $\mathscr{D}_S$ and $\mathscr{T}_S$.[Lin and Jung, 2017]

In Deep Learning, this technique is implemented by pre-training neural networks on various tasks which provide generic features which are then used to build models for new target tasks without training specific neural networks from scratch for the mentioned target task [Delgado de Santos et al., 2020]. There are two Transfer Learning strategies:

- a pre-trained model can be used as feature extractor for the target data by only updating the final layer of the model
- retraining the model with the target data initializing with the learned weights from the training with the large amounts of data, known as fine tuning [Pan and Yang, 2010]

In imaging, Transfer Learning is commonly used to fine-tune the pre-trained networks parameters for feature extraction purposes. These typical neural networks include AlexNet, VGG-16, VGG-19 or ResNet [Shaha and Pawar, 2018].
In the case of YOLOv5, the second type of transfer learning is deployed in all cases(experiments): all our models were initialized with pre-trained weights, namely, YOLOv5s6 weights. However, in the experiments to follow we are going to be experimenting with the first type of

transfer learning, namely, freezing the backbone and the head of the model and updating only the final layer's weights. Listing 3.1 shows the backbone and the head of the model as implemented in Ultralytics library. For the purposes of this thesis, we experimented with freezing all the layers besides the last one, and compared the results to the complete network trained from scratch, without freezing any of the layers.

Chapter 6

Results

6.1 Experiment 1: Image and Dataset Size

Table 6.1: Results of training on a dataset with 1330 images of size 1152ppi

Class	**Precision**	**Recall**	**mAP@0.5**	**mAP@[0.5:0.95]**
all	**0.74**	**0.599**	**0.646**	**0.229**
fragment	0.809	0.717	0.728	0.212
line	0.805	0.352	0.581	0.166
organic	0.738	0.613	0.603	0.194
pellet	0.825	0.763	0.756	0.256
tar	0.523	0.548	0.561	0.272

To restate what was said in section 5.1, the first experiment served to establish which of the two datasets at hand yields better results. Namely, a dataset comprising 1330 images of size 1152 pixels was compared to the dataset comprising 6688 images of size 576ppi. The hyperparameters and settings used in comparison of the models were exactly the same, the only difference being adjusting the img_size flag in the yolov5 model. Batches of 14 were deployed in 150 epochs, and the initial 12 layers of the model backbone were 'frozen', in

Table 6.2: Results of training on a dataset with 6688 images of size 576ppi

Class	**Precision**	**Recall**	**mAP@0.5**	**mAP@[0.5:0.95]**
all	0.693	0.518	0.547	0.202
fragment	0.746	0.566	0.604	0.223
line	0.743	0.262	0.408	0.129
organic	0.643	0.599	0.619	0.203
pellet	0.828	0.68	0.714	0.293
tar	0.506	0.481	0.39	0.163

other words, transfer learning described in section 5.3 was deployed. Low data Augmentation settings were chosen. The results in terms of F1 curves and Precision-Recall curves can be observed on Figure 6.1 and Figure 6.2, respectivelly. First of all, it is important to observe how the 1152 pixels dataset scores higher F1 value, namely an average f1 of 0.65 when confidence is at 0.436 in comparison to 0.52 F1 when confidence is at 0.363 achieved by the 576 pixels dataset.

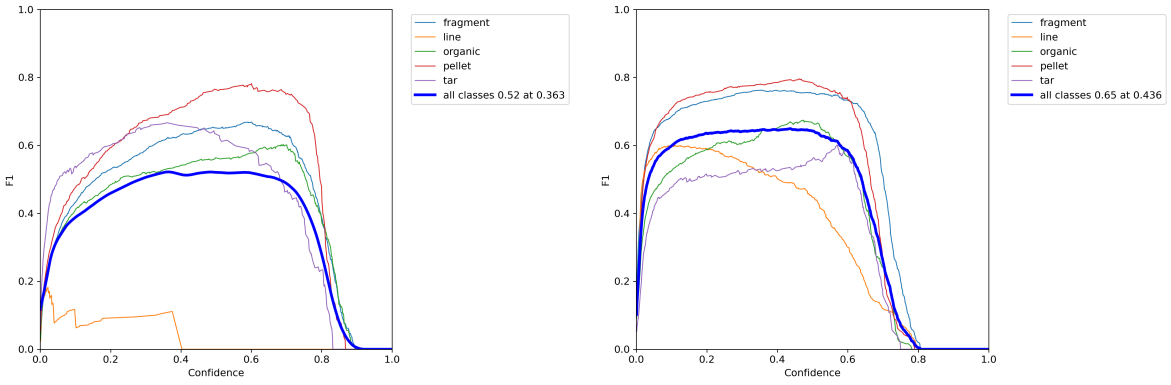

Figure 6.1: F1 curves relevant to the dataset comprising images of size 576 pixels (left) and 1152 (right).

In addition, we can observe how the PR curve relevant to the 1152 pixels dataset displays a higher area under the curve (mAP@0.5), implying it has both a higher precision as well as

a higher recall, though the higher precision is definitely a more predominant hallmark of this model. This implies a significantly lower false positive rate, and somewhat lower false negative rate, which is an improvement with respect to the 576 pixels models.

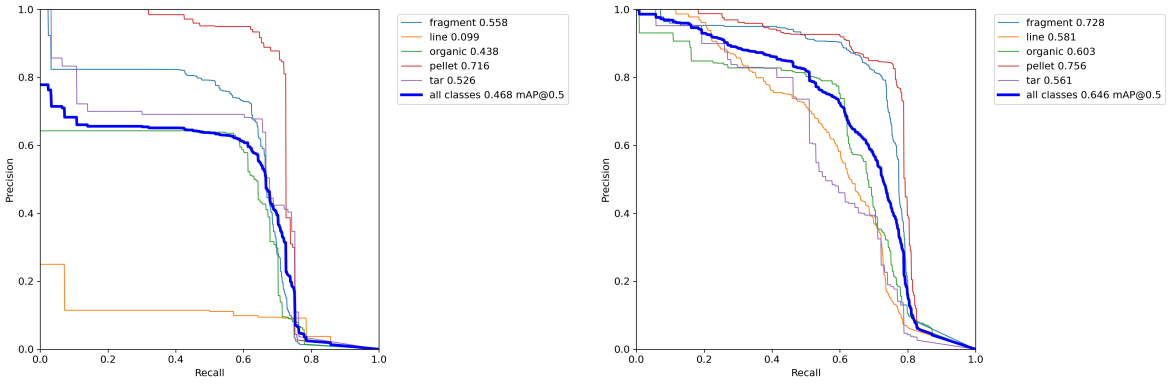

Figure 6.2: Precision-Recall curves relevant to the dataset comprising of images size 576 pixels (left) and 1152 (right).

Confusion Matrices are also interesting to observe when comparing these two datasets. One of the most problematic cases in both datasets is the 'line' class. Namely, we can observe how in the 576 pixels dataset, the line class often used to be predicted as organic class, namely in 71% of the cases, which is a problem that is nearly wiped out in the 1152 pixels dataset. However, the 1152 pixels dataset presents another (though minor) problem with this class: the line is misclassified as a background noise in 51% of the cases. In addition, the TP rate for the line class is only 7% in the 576 dataset, whereas it is 48% in the 1152 pixels dataset. That is an improvement of 41% percentage points, just for this class, though it's worth mentioning that literally all of the classes had an improved TP rate in the 1152 pixels dataset.

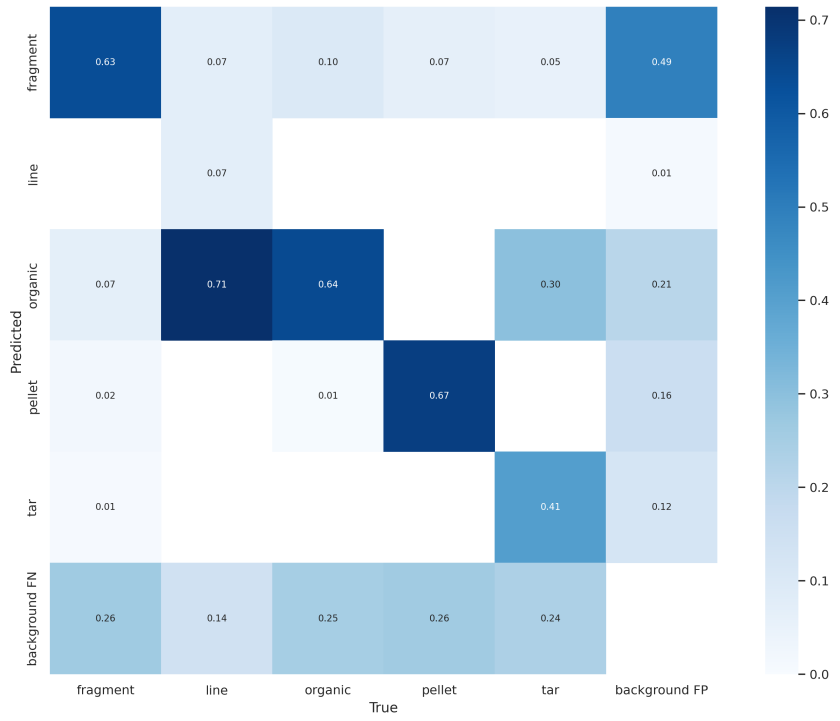

Figure 6.3: Confusion Matrix relevant to the dataset comprised of images size 576 pixels.

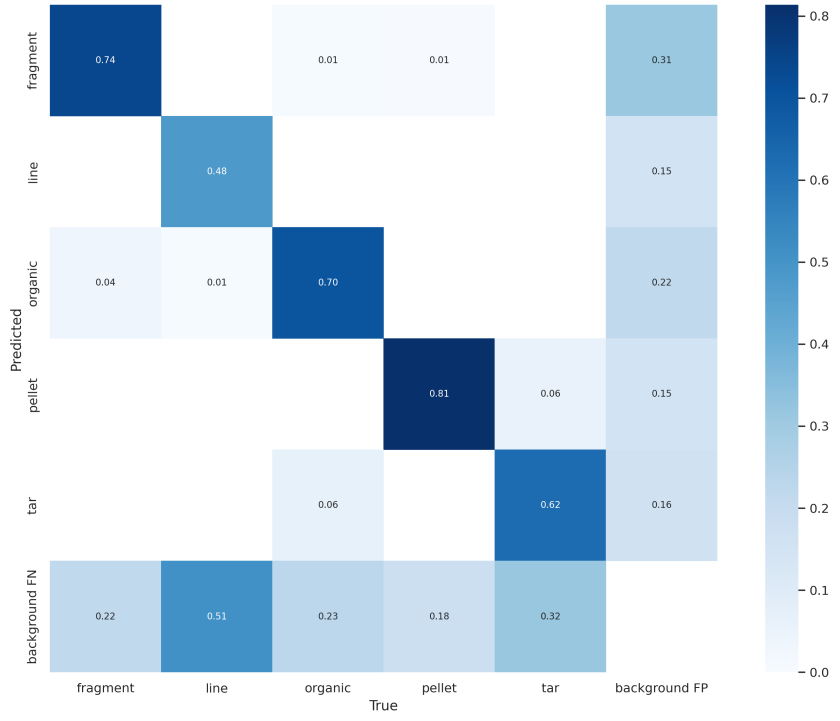

Figure 6.4: Confusion Matrix relevant to the dataset comprised of images size 1152.

Lastly, as can be observed from Table 6.1 and Table 6.2, the dataset with images of size 1152 pixels yields better results for literally every class, even though it contains less images. This was expected, as the YOLOv5s6 weights were trained on images of size 1200ppi, which corresponds more closely to the dataset with images sized 1152 than the dataset with images sized 576.

Therefore, it was decided to proceed with further experiments using the 1152 pixels dataset with 1330 images.

6.2 Experiment 2: Data Augmentation and Transfer Learning

The first experiment conducted once the dataset was established is a combination of high DA settings and TL by freezing the backbone and most of the head of the net (everything except for the final layer).

Observing the precision-recall curve relevant to this experiment (upper left), we can definitely tell that the class with the highest precision is pellet, with mAP of 0.822 and the class with lowest precision is line with mAP of 0.822. However, the class with the lowest recall value is tar, with mAP of 0.499, whereas the class with the highest recall is organic, with mAP of 0.798. The low performance of tar might be attributed to its shape which is oftentimes confused with organic and sometimes confused with fragment. The total mAP of the model in question compares better to the section 6.3 and section 6.5, whereas it compares worse to section 6.4.

In the case of the F1 curve, the pellet definitely achieves the highest score, surpassing 0.80 at its peak, whereas the class with the lowest F1 score is definitely line. The confidence value that optimizes the precision and recall is 0.250, which is the lowest of all of the four experiments, though the very F1 score is the second best of all of the 4 experiments, following the section 6.4. Similarly to the PR curve, the F1 score compares better to the section 6.3 and section 6.5, whereas it compares worse to section 6.4.

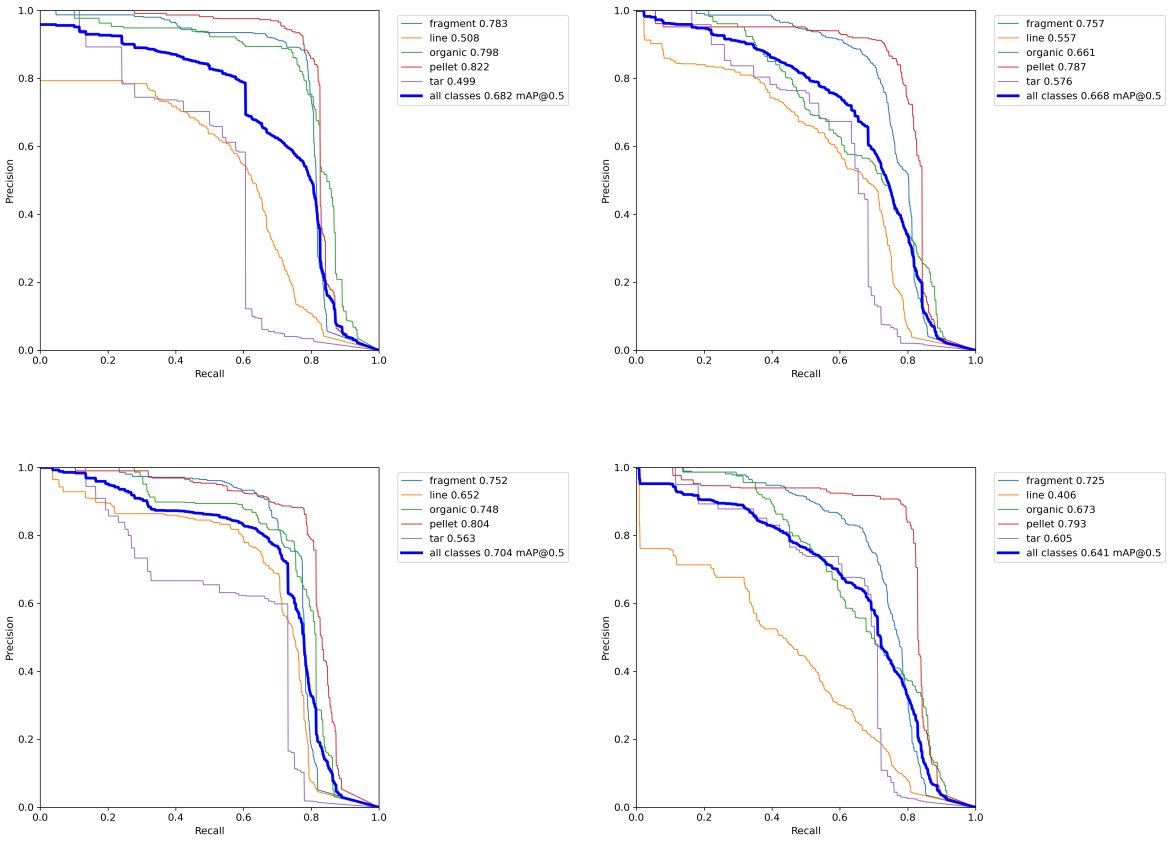

Figure 6.5: Precision-Recall curves relevant for each of the four experiments with Data Augmentation (DA) and Transfer Learning (TL). From left to right and from top to bottom, images are relevant to the Experiment 2: DA and TL, Experiment 3: DA and no TL, Experiment 4: No DA and no TL ,and Experiment 5: No DA and TL.

Finally, as can be seen from the Table 6.3, the High DA and TL experiment does not really excell with respect to the remaining experiments in nearly any metric, nor in any particular class, except for the case of the tar class which in the case of high DA and TL displays the highest precision value of all of the four experiments.

Table 6.3: Results High Data Augmentation and Transfer Learning

Class	**Precision**	**Recall**	**mAP@0.5**	**mAP@[0.5:0.95]**
all	0.704	0.624	0.668	0.233
fragment	0.708	0.744	0.757	0.242
line	0.71	0.447	0.557	0.184
organic	0.712	0.5	0.661	0.212
pellet	0.77	0.795	0.787	0.284
tar	**0.621**	0.635	0.576	0.243

6.3 Experiment 3: Data Augmentation without Transfer Learning

In this experiment, the high DA settings have been combined with training the complete network, without freezing the backbone of the YOLOv5 model.

On the PR curve (upper right) on Figure 6.5, it can be observed that the mAP is slightly lower than in the previously discussed section 6.2 as well as the section 6.4, whereas it is higher than section 6.5 . In this experiment, the line is (again) displaying the lowest mAP value, whereas the pellet is (again) showing the highest mAP value ,this time closely competing with the fragment class.

The F1 curve on the upper right on Figure 6.6, which is relevant to this experiment, is telling us that the worst scoring values are tar and line, though the organic class is not performing too well either. The confidence value that optimizes the F1 value is 0.360, which is the second best confidence of all of the four experiments, following the experiment from section 6.4. Nevertheless, the very F1 score is not impressive, namely it's second worst, following section 6.4 and section 6.2. However, it is still worth noting that it is an interesting trade-off. Sometimes a suboptimal F1 score when coupled with good confidence can be prefered to an optimal F1 score with lower confidence.

To discuss the results of this experiment presented in Table 6.4, it is first of all worth observing that, in this experiment either, none of the relevant metrics for all classes achieves the best results of all the experiments. However, several classes do display interesting results in terms of Recall, mAP@0.5 and mAP[0.5:0.95]: namely, fragment and organic both score the highest values for these three metrics, whereas the pellet is not lagging too much behind by displaying the highest results of all of the four experiments in terms of recall and mAP@0.5.

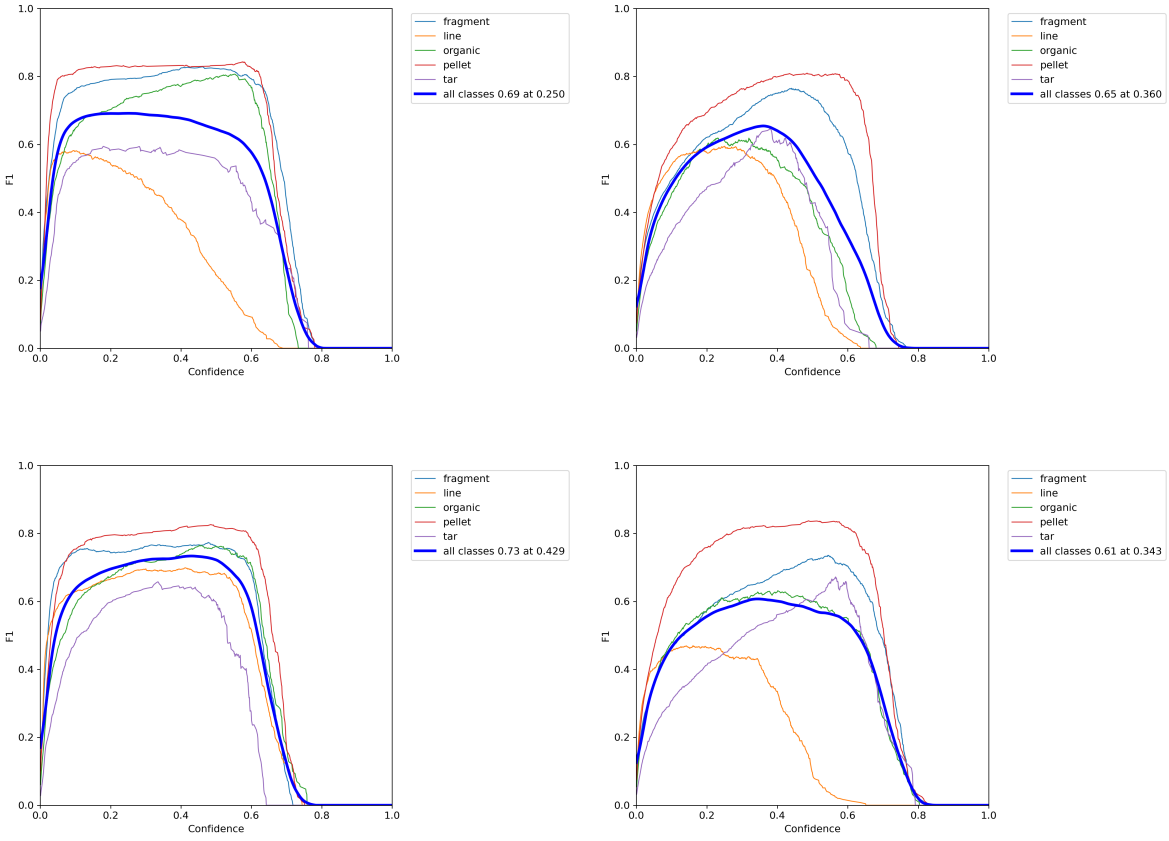

Figure 6.6: F1 curves relevant for each of the four experiments with Data Augmentation (DA) and Transfer Learning (TL). From left to right and from top to bottom, mages are relevant to the Experiment 2: DA and TL, Experiment 3: DA and no TL, Experiment 4: No DA and no TL, and Experiment 5: No DA and TL.

Table 6.4: Results High Data Augmentation and No Transfer Learning

Class	Precision	Recall	mAP@0.5	mAP@[0.5:0.95]
all	0.713	0.689	0.682	0.262
fragment	0.788	**0.796**	**0.783**	**0.306**
line	0.701	0.426	0.508	0.172
organic	0.634	**0.827**	**0.798**	**0.261**
pellet	0.854	**0.809**	**0.822**	0.306
tar	0.586	0.585	0.499	0.263

6.4 Experiment 4: No Data Augmentation and No Transfer Learning

This experiment, deploying low DA settings and no TL, displays the most favorable results from many different perspectives.

Starting with the precision recall curve in Figure 6.5, this experiment displays the best mAP value of 0.704 mAP@0.5, exceeding the second best one(section 6.3) by one percentage point. The class with the highest mAP@0.5 is again pellet, this time with somewhat lower score of 0.804. The worst performing class in the sense of the same metric is tar with a mAP@0.5 of only 0.563.

Next important thing to consider when determining the best performing model is the Precision, Recall and the F1 score. As can be observed from Figure 6.6, this experiment is showing the best results in terms of F1 per confidence, again. It achieves F1 score of 0.73 at confidence of 0.429, which is both the highest F1 as well as the highest confidence score at which the F1 is reached. The lowest F1 values are scored by Tar and the highest by Pellet.
A less obvious feature of the model created in the Experiment 4, is the fact that the confidence values are quite proportionate among the classes. Namely, the remaining experiments usually showcase some classes with a lower confidence (curve peak leaning left) and other

classes with higher confidence (curve peak leaning right). However, the F1 curves relevant to the Experiment 4 are quite concentric, telling us that there is less variance between the differences in values among confidence of the five classes.

Table 6.5: Results Low Data Augmentation and No Transfer Learning

Class	**Precision**	**Recall**	**mAP@0.5**	**mAP@[0.5:0.95]**
all	**0.776**	**0.7**	**0.704**	**0.271**
fragment	**0.877**	0.679	0.752	0.282
line	**0.766**	**0.627**	**0.652**	**0.246**
organic	**0.759**	0.75	0.748	0.233
pellet	**0.866**	0.779	0.804	**0.331**
tar	0.612	**0.663**	0.563	0.261

Finally, to confirm the result discussed above, we refer to the Table 6.5. As intuited from the plots discussed, the Experiment 4 does indeed score the highest of all the five experiments for all of the four relevant metrics. The model trained on our dataset with the given settings yields Precision score of 0.776, Recall of 0.7, mAP@0.5 of 0.704 and mAP@[0.5:0.95] of 0.271. This means, that compared to the other models, this one has is correct almost 78% of the time it makes a prediction, and it correctly identifies 70% of objects it detects.

This model is interesting because it showcases higher Precision than any other model for all of the classes, except for Tar which achieves the highest precision in Experiment 2. With these experimental settings, Line class achieves the highest Recall and mAP values, whereas the Tar achieves the highest Recall value here. mAP@[0.5:0.95] is the highest in this case for Line and Pellet class.

It is quite counter-intuitive that the performance worsens with higher data augmentation settings. This might be so due to several reasons. First of all, as was mentioned in section 5.2, even though we use the terms 'no data augmentation' versus 'data augmentation' here for

the sake of simplicity, we are actually comparing between high and low DA settings. Namely, the high vs. low data augmentation settings differ in the following: high data augmentation settings have a higher learning rate, lower classification loss gain, lower object loss gain, higher image scale, higher probability of image mixup, and higher probability of segment copy paste, compared to the low data augmentation settings. Thus, it might simply be that the higher data augmentation settings are contributing to overfitting the model, whereas the lower DA settings are just the right amount of data augmentation for our dataset.

6.5 Experiment 5: No Data Augmentation and Transfer Learning

This experiment is interesting because of its high mAP@0.5 and mAP[0.5:0.95] value relevant to class Tar, which was one of the worst performing classes in terms of Precision throughout various models.However, Line and Fragment classes achieve the lowest maP@0.5 of all of the five experiments, though the Fragment class does not lag too much behind with respect to the other experiments. Other than that, this model is probably the least interesting one of all the four because of it's relatively low mAP@0.5 value, as well as the F1 value.

The confidence values at which the highest F1 score is reached are not similar among different classes, like they were in Experiment 4. This can be observed from the bottom right plot on Figure 6.6. The curve relevant to tar Line is displaying a peak towards left, meaning it scores the peak F1 value at a relatively low confidence value, whereas Tar, Organic and Pellet peak at higher confidence levels, reaching a higher F1 value at a higher confidence.

6.6 Other Metrics

In this section, the comparison of CIoU losses and detection speeds can be studied.

Table 6.6: Results Low Data Augmentation and Transfer Learning

Class	**Precision**	**Recall**	**mAP@0.5**	**mAP@[0.5:0.95]**
all	0.636	0.625	0.641	0.242
fragment	0.587	0.744	0.725	0.301
line	0.661	0.284	0.406	0.124
organic	0.669	0.577	0.673	0.202
pellet	0.837	0.808	0.793	0.285
tar	0.428	0.712	**0.605**	**0.299**

6.6.1 CIoU Loss

As stated previously, CIoU loss is a regression loss incorporating all geometric factors: overlapping area, distance, and aspect ratio, and it is used to determine how the trained model fits the new data. Table 6.7 shows us that the experimental settings cited in section 6.3 yielded the lowest training loss, meaning the model with settings from Experiment 3 is the one that best fits the training data.More importantly, the settings from the Experiment 4, explained in section 6.4 displayed the lowest validation loss, meaning that is the model that best fits validation data. The worst performing model on the validation set is coincidentally the one from the Experiment 3, which happens to best fit the training data. This is a sign that the model might be overfit.

It is worthwhile noting that the data relevant to the Experiment 1 is skipped in this case because it is not truly comparable to the remaining ones due to its nature.

6.6.2 Detection Speed

For the commercial purposes of microplastics detection and counting software, as well as for the purpose of this study, one of the requests of the object detection model refers to the inference time, since a real-time object detection is necessary. The lowest average pre-processing

Table 6.7: CIoU loss for each experiment

Experiment	Training CIoU Loss	Validation CIoU Loss
Exp2: DA, TL	0.054378	0.060657
Exp3: DA, no TL	**0.049147**	0.068782
Exp4: no DA, no TL	0.053363	**0.060633**
Exp5: no DA, TL	0.060627	0.06285

and inference time per image can be attributed to the Experiment 4, whereas the lowest NMS was obtained for Experiment 3. However, most times appear to be quite similar so the time likely shouldn't be among the deciding factors for model selection.

Table 6.8: Speeds for detecting an image using model from each experiment (ms)

Experiment	Pre-processing	Inference	NMS
Exp2: DA, TL	1.1 ±0.07	20.8 ±1.42	4.4 ±0.26
Exp3: DA, no TL	1.0 ±0.08	21.3 ±1.22	3.1 ±0.18
Exp4: no DA, no TL	0.9 ±0.11	20.9 ±1.60	4.8 ±0.33
Exp5: no DA, TL	1.1 ±0.07	21.9 ±0.98	3.3 ±0.19

6.7 Experimental Settings Selected for Post-Processing

Now that the extensive analysis of results of the four experiments has been conducted, it has become quite clear that the model that exceeds in terms of most metrics is indeed the model with settings described in **Experiment 4**, section 6.4. This model showcased the highest Precision, Recall, mAP[0.5] and mAP[0.5:0.95] for all classes. As discussed in section 6.4, it also bears the highest scoring precision for all classes except for Tar of all the five experiments, whereas Recall, mAP[0.5] and mAP[0.5:0.95] exceed for at least some classes compared to the

other experiments. Moreover, it is also the model which best fits the validation set in terms of its CIoU validation loss, which can be observed from the Table 6.7.

6.8 Post-Processing Results

The selected model was deployed together with the post-processing method described in section 4.7 at a confidence threshold of 0.5. The results yielded are displayed below. First of all, the confusion matrix for the test set on may be observed on Figure 6.7 (which, as mentioned in subsection 4.6.1 contains 140 images derived from two full sized input images). As described in section 4.7, the postprocessing was done on both of the two original input images in the test set. On Figure 6.7 the confusion matrix of the final test set is provided, which is a result of the post processing method deployed with the YOLOv5 model. On the first glance, it is already visible that the true positive rateis everwhelmingly large compared to the false positives and the false negatives.

		Predicted						
		fragment	line	organic	pellet	tar	Background FN	Total
Real	Fragment	46	0	0	0	0	0	46
	Line	1	1	0	0	0	0	2
	Organic	0	1	19	0	0	0	20
	Pellet	0	0	0	19	0	0	19
	Tar	0	0	2	0	9	1	12
	Background FP	1	0	0	0	0	0	1
	Total	48	2	21	19	9	1	100/100

Figure 6.7: Confusion Matrix for the Final System using YOLOv5

Precision, Recall and F1 value relevant to the final system as well as individual classes can be observed on Table 6.9. The model has an overall recall value of 98.9%, meaning it correctly classifies nearly 99% of particles. All of the individual particles have a perfect recall value of 100% on the test set, except for Tar which displays a somewhat lower recall value of 90%, which is still considered high.

Moreover, this architecture resulted in 94% precision for all classes, which means that

when it predicts that a particle belongs to a certain class, it can be expected to estimate correctly 94% of the time. The Precision is equal to 100& for particles Pellet and Tar, whereas it's somewhat lower for Organic and Fragment class, 96% and 90%, respectively. However, the Line class displays the lowest value of precision, namely only 50%. This can be explained by a low number of samples of particles with this class, namely, only 2% of all of the particles tested belongs to the class Line.

Finally, F1 metric for all the classes is 96%, being the harmonic mean of precision and recall. As expected, the class Line presents the lowest F1 score, of 67% whereas the class Pellet has the highest one, 100%.

According to these values, the overall performance of the model is satisfactory.

Table 6.9: Precision, Recall, F1 per class and for all classes

Class	**Precision**	**Recall**	**F1**
Fragment	0.9583	1	0.9787
Line	0.5	1	0.666666667
Organic	0.904761905	1	0.95
Pellet	1	1	1
Tar	1	0.9	0.947368421
All	0.94	0.989473684	0.964102564

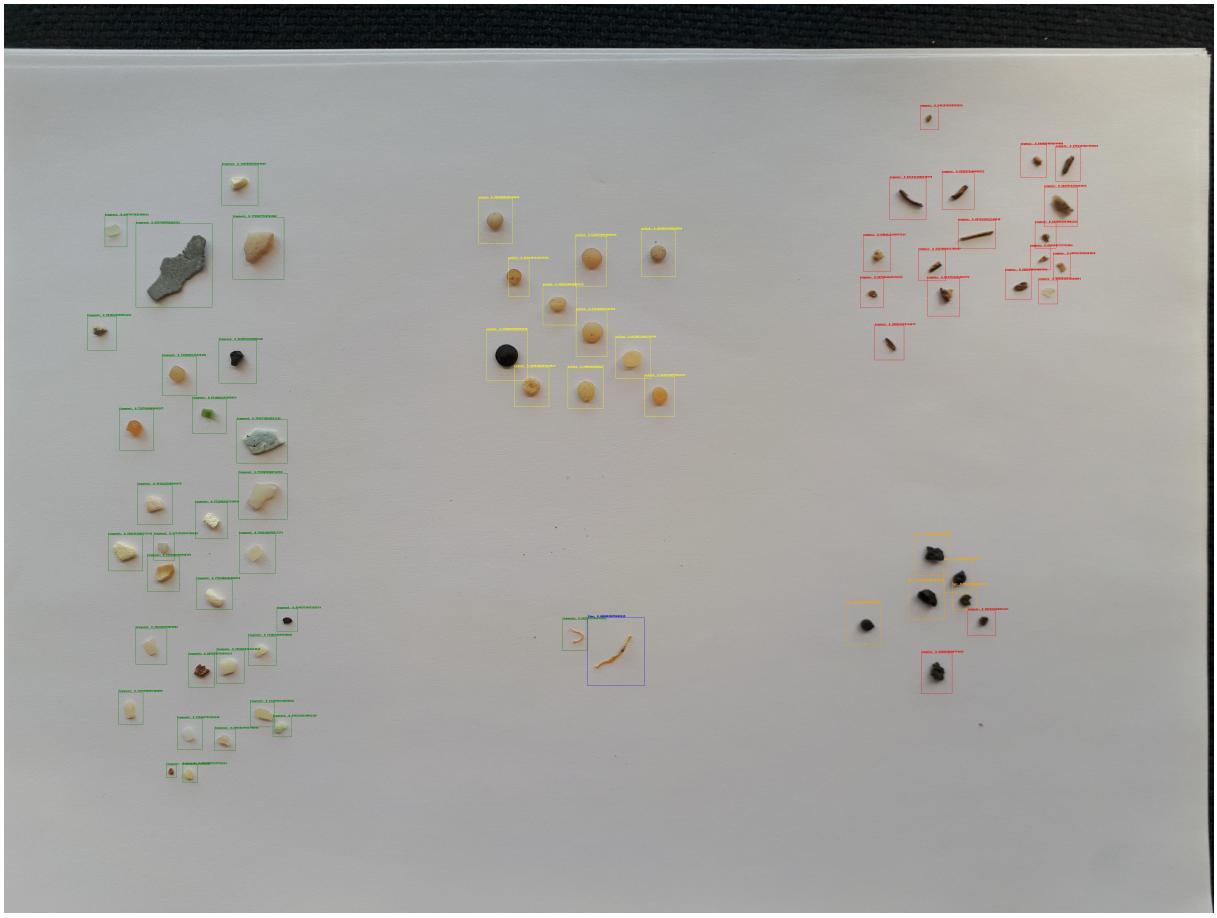

Figure 6.8: Example of a final output image of the system

6.9 State of the Art System for Automatic Microplastics Classification

In this section, the state of the art model for microplastics classification is trained and used for object detection and classification on the same dataset as the one used for deploying YOLOv5. The purpose of this process is thus to compare the performance of YOLOv5 model with the performance of the current best model for microplastics classification, and determine if the pipeline created in this work exceeds the previous best performing model in terms of relevant metrics,or not.

6.9.1 Description of the Architecture

As described in [Lorenzo-Navarro et al., 2021], the method suggested in the work is a hybrid approach between bounding-box based approach and an instance segmentation approach. It is comprised of two consecutive steps: microplastics semantic segmentation, and microplastics classification. In the first phase, an instance segmentation is realized to detect the particles in the sample image encompassing all the types of microplastics in only one, the class particle versus the background. As result of this first phase, the pixels that correspond to particles in the images are obtained. This result is important in microplastics analysis because the shape of the particles is a cue about its origin. After the microplatics has been segmented in the sample image, in the second phase, each particle is classified with a fine-tuned convolutional neural network.

Semantic Segmentation

The first phase of instance segmentation is performed with UNET that was originally proposed for pixel-level segmentation of biomedical images [Ronneberger et al., 2015]. This network has the advantage over other architectures in that it requires a small number of images to successfully train it. The architecture is composed of two pathways: a contracting pathway that is composed of layers of convolutions, and the expansive pathway that is composed of upward convolutions. A characteristic of this architecture is the concatenation of features from the contraction path to those of the expansive path. As stated above, the resolution of the sample images must be high enough to maintain small particle details so that they cannot be reduced to feed the network. The solution was to divide the sample image into smaller overlapping patches, apply instance segmentation to each patch, and finally concatenate the patches to reconstruct the resulting segmented sample image. The use of overlapping patches instead of non-overlapping patches is a way to solve the obvious drawback of considering a single particle as two different ones when when it falls between two adjacent non-overlapping patches. The illustration can be observed on Figure 6.9. It is worthwhile noting that this step is quite similar to what was done in this work, as described

in section 4.5.

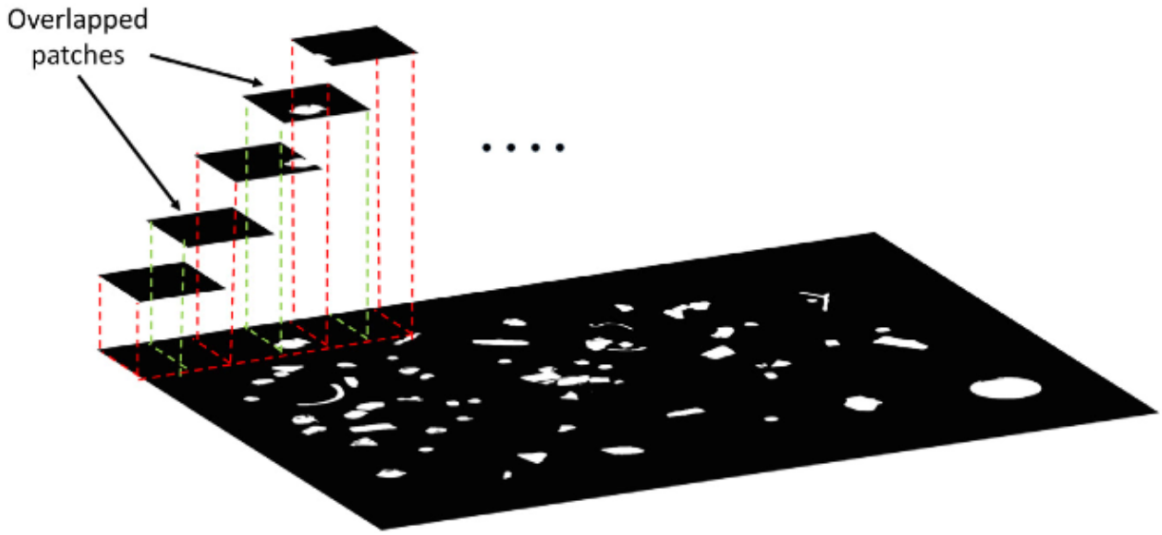

Figure 6.9: Microplastics segmentation with overlapped patches to avoid breaking a particle in two adjacent patches as shown in [Lorenzo-Navarro et al., 2021]

UNET was trained by leveraging on the bounding boxes created for the dataset in order to train the YOLOv5 model. Namely, the bounding box regions were converted to pixels of value 255 (white) whereas the background was converted to 0 (black). When these masks were plugged into UNET model together with the corresponding ground truth images, the output was a well-segmented mask. Thus the model was ready for being deployed on the test set.

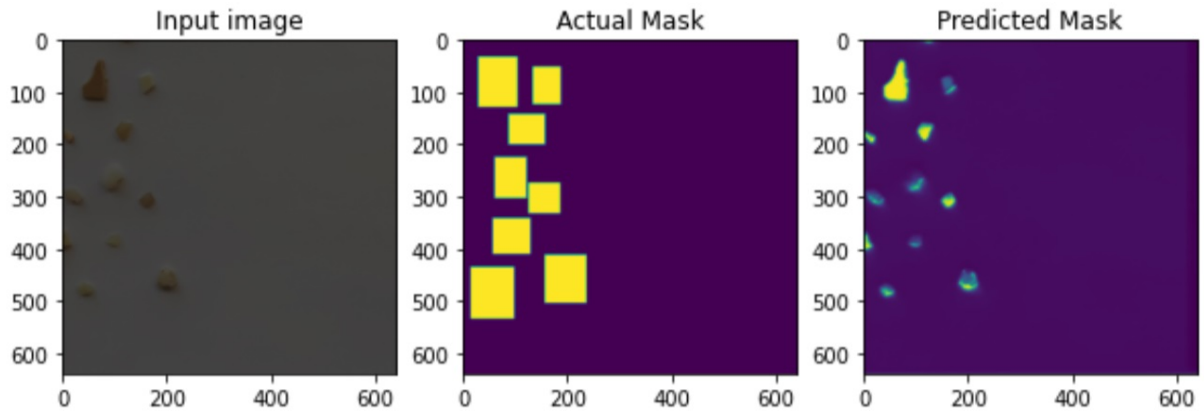

Figure 6.10: Original image, training mask, and the mask predicted by UNET

Classification

VGG16 net [Simonyan and Zisserman, 2014] was fine tuned using the 'Imagenet' weights [Ima,] with SGD optimizer. As described in [Lorenzo-Navarro et al., 2021], some modifications have been introduced into the original VGG16 architecture to adapt it to the microplastics classification task. Firstly, the dimension of the input images has been set to 132 × 132 pixels in RGB color instead of the original 224 × 224 pixels. From the three fully-connected layers, the two first ones have been reduced to 128 and 64 units respectively, and a batch normalization layer is introduced after each layer. The last layer has three outputs, corresponding to the microplastics classes under consideration, with a softmax activation function. To reduce the number of samples necessary in the training process, the convolutional layers of the net- work are initialized with weights that had been pre-trained on ImageNet. Each test image was segmented and the cropped image of each particle was extracted by leveraging on the connected components of the generated testing mask. The relative coordinates of the cropped images of particles were saved in the name of the image. The extracted images were classified, and the bounding box coordinates and the predicted class were saved into the aforementioned .csv file. Ultimately, the full-sized input test image was recomposed using the aforementioned csv file.

		Predicted						
		fragment	line	organic	pellet	tar	Background FN	Total
Real	Fragment	4	0	4	4	2	33	47
	Line	0	1	0	0	0	1	2
	Organic	1	0	1	4	0	14	20
	Pellet	2	0	0	7	1	9	19
	Tar	0	0	0	3	3	6	12
	Background FP	11	1	1	0	1	0	14
	Total	18	2	6	18	7	63	100/110

Figure 6.11: Confusion Matrix for the Final System using VGG16 and UNET suggested by [Lorenzo-Navarro et al., 2021]

6.9.2 Results

The implementation of the system devised in [Lorenzo-Navarro et al., 2021] with the dataset used to implement this work did not deliver results similar to the ones quoted in the paper. In addition, the confidence threshold had to be lowered from 0.5 implemeted with YOLOv5 to 0.3, since most detections' confidence score did not exceed that threshold, and there would be nearly no results to report.

The model has an overall recall value of 20.2%, meaning that is the percentage of the particles it correctly classifies. The highest recall value observed is 50% relevant to the Line class, and the lowest one is for the organic class, being less than 7%. This means the model correctly classifies half of the Line particles and less than 7% of Organic particles. The rest of the classes lie within that range. As mentioned previously, the unusual results connected to the line particle might be explained by a low number of samples of particles with this class, namely, only 2% of all of the particles tested.

Additionally, this architecture resulted in 31.3% precision for all classes, which means that when it predicts that a particle belongs to a certain class, it can be expected to estimate correctly 31.3% of the time. The Precision is the highest for Tar class, namely almost 42%, whereas it's the lowest for Organic class, only 16.7%.

Finally, F1 metric for all the classes is 24.6%, being the harmonic mean of precision and recall. As expected, the class Line presents the highest F1 score, of 50% whereas the class

Organic has the lowest one, less than 10%.

According to these values, the overall performance of the model is not satisfactory. The values differ a lot from the ones suggested in [Lorenzo-Navarro et al., 2021], which can be explained in several ways, highlighted in section 6.10.

Table 6.10: Precision, Recall, F1 per class and for all classes deploying VGG16 and UNET as suggested in [Lorenzo-Navarro et al., 2021]

Class	**Precision**	**Recall**	**F1**
Fragment	0.222222	0.108108	0.145455
Line	0.5	0.5	0.5
Organic	0.166667	0.066667	0.095238
Pellet	0.38888	0.4375	0.411765
Tar	0.428571	0.333333	0.375
All	0.313725	0.202532	0.246154

As can be seen from the resulting sample image, most particles were classified as background noise (FN). The model appears to perform better at detecting and counting particles rather than actually classifying them. A major concern here is the black background at the corners of input images which oftentimes confuses the model and classifies parts of the background as particles. Class Pellet appears to be performing relatively well here, as can be seen from Figure 6.11 and Table 6.10

Figure 6.12: Example of a final output image of the system suggested in [Lorenzo-Navarro et al., 2021]

6.10 Discussion

Now that the results have been observed and commented in terms of several different metrics, both the best performing model leveraging on YOLOv5 as well as the State of the Art System for microplastics detection and classification (UNET and VGG16 as described in [Lorenzo-Navarro et al., 2021]), it is time to compare and discuss the results.

6.10.1 The Two Systems

It should be clear by now that, considering the dataset described and selected in section 6.1, the best performing model is indeed the YOLOv5 pipeline presented in this work. It far exceeds its predecessor, in terms of all metrics and for all classes. Even though the system presented in [Lorenzo-Navarro et al., 2021] showcases significantly better results on their dataset, the system fails on this dataset for several possible reasons.

First of all, the model in [Lorenzo-Navarro et al., 2021] was trained and tested on a dataset comprised of more images, namely 49 images of size 3456*4608 later cropped into images of size 512 pixels, with an overlap. Other than the fact that our dataset contained significantly less images (19), the cropped images are more than twice the size the ones used in [Lorenzo-Navarro et al., 2021] (1152 pixels compared to 512 pixels). Certain models, such as VGG16 and UNET might benefit from smaller resolution images, and this could be one part of the explanation as to why the model which performed so successfully in [Lorenzo-Navarro et al., 2021] failed on the dataset used in this research.

Second reason for the failure of the system presented in [Lorenzo-Navarro et al., 2021] might be the black background behind the paper on which the particles were arranged, which can be observed in the top left border of Figure 6.12. The UNET model continually masks this background as one gigantic particle, or several smaller ones. This was partially mended by leveraging on the size and area of connected components used in order to extract the dimensions of particles during the process of masking the test images. Regardless, the model kept on masking some of the background pixels as particles. The interesting results of this 'glitch' can be observed in the top right corner of Figure 6.12, where three 'ghost' particles are detected on a black background. This is an interesting example of a False Positive detection.

Third of all, the scientists devised the method presented in [Lorenzo-Navarro et al., 2021] with purpose of detecting and classifying three different classes of microplastics. The premise was that the shape of the particle is a cue for its class; namely Line has an elongated shape, Pellet is usually round, whereas Fragments show irregular polygonal shapes. This, indeed,

was part of the reasoning behind leveraging on semantic segmentation: the shape of the particle gives a cue on the origin of the particle, and the mask stores the information on the shape of the particle.
However, in case of our 1152 pixel dataset, we are working with five instead of three classes. Since some of these classes can display similar shapes, the shape of the particle drops in its relative importance in the detection and classification process.

6.10.2 Resource Limitations

This project underwent resource limitations, as do most projects in this field. In this subsection, we discuss the main resource limitations, and what could have been done if it were not for them.

Firstly, while numerous metrics have been obtained for measuring the performance of both the object detection models as well as the complete systems, there is one metric that was not taken into account when devising the complete system, and that is Mean Average Precision. While this metric was measured for individual model settings, it was not calculated for the complete systems, and that is something that could have helped in assessing the overall performance of the systems. This is a feature of this work that was affected by Time Limitations.

Secondly, as was explained in section 6.1, the Experiment 1 was prepared in a way that the 'default' settings for YOLOv5 model were chosen, and the two datasets were tested on these same settings. The best performing model was chosen among the two, and we proceeded with the remaining experiments using that dataset. Ideally, in order to make sure that the selected dataset is indeed the best performing one, the datasets could have been tested on several different model settings, and then evaluated. This was not done due to both the time as well as hardware (GPU) limitations, and most importantly due to data limitations.

Thirdly, the dataset provided was rather small, namely it comprised only nineteen 3456*4608 images. This further influenced the testing set, which was comprised of cropped up images

belonging to two original, full sized input images. For that reason, in the test set, there were only 100 particles to be detected, and only two of them belonged to Line class. For that reason, it was hard to assess reliably the performance of that class. The test set could have been expanded by one more image, but it was decided to keep it for the training and validation sets, since otherwise these sets would've been rather small.

Finally, in order to increase the reproduceability of results, a k-fold cross validation could have been done. This way the test set would change every time, and the results in terms of precision, recall and F1 would be more reliable. However, as the GPU resources were limited and uncertain (Google Colab's GPU limits vary daily), this process was not done.

Chapter 7

Conclusions

This work stems from the desire to improve the performance of the state of the art model for automatized microplastics classification and counting, in terms of Precision, Recall and F1 metric, as well as in terms of simplicity, i.e. versatility of application. Moreover, the objective was to devise a deep learning model for successfully detecting and classifying three classes of microplastics particles and two classes of byproducts of microplastics collection without the need for special equipment.

The idea behind this work was to use a Mosaic with Overlap Image Enhancement method presented in section 4.5 to solve two issues: increase the number of images in the dataset, and reduce the size of the images without losing data on smaller particles in order to reduce the computational cost of working with high resolution images. Having done the preprocessing, the YOLOv5 model was selected to perform object detection and classification, due to its impressive features explained in section 3.2.

Therefore, to our best knowledge, this is the first work on automatic plastics classification and counting apt for a five class dataset comprised of images taken with a simple camera, not relying on any special equipment. The previous model appeared to perform well in the original paper [Lorenzo-Navarro et al., 2021], but when tested on our dataset, the model's results were significantly worse in terms of Precision, Recall and F1. The state of the art model was easily confused by dark background behind the paper, which is a problem that

the system presented in this work solves. More importantly, the state of the art model, in its original paper ([Lorenzo-Navarro et al., 2021]) was tested on a dataset with only three classes of particles which are differentiated mostly by their shape, but it performed poorly on a five class dataset used in this work, where some classes display similar shapes. The premise of leveraging on shape of the particles used in their work is not as relevant in this case.

7.1 Future Research Directions

The results of the system devised in this work were shown to be extremely satisfactory. In this section several possible further research directions are discussed.

As has been described in chapter 4, the microplastics particles were arranged on a white background before taking a photo. A potential research direction is to test and revise this system on a dataset comprising images of microplastics in it's natural setting, namely, on the beach sand. This would significantly cut the time costs of classifying and counting particles, because there would be no need to first collect them, clean them off of sand and residue, and arrange them on a white piece of paper in a laboratory, which is a timely and tedious process. Thus, the next possible step might be implementing this predictor on a real-world setting, i.e. beach sand.

Another possible research direction is to simply enlarge the current dataset. As has already been mentioned in chapter 4, the dataset is comprised of only 19 images of size 3456*4608. A larger dataset is likely to contribute to increasing model's performance metrics since the training set is going to be larger and more diverse.

Bibliography

[CNN,] Convolutional neural network. https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html. Accessed: 2022-07-31.

[mor,] Convolutional neural network. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way Accessed: 2022-07-31.

[col,] Google colab. https://colab.research.google.com/?utm_source=scs-index. Accessed: 2022-07-31.

[Obj,] https://github.com/rafaelpadilla/object-detection-metrics. https://github.com/rafaelpadilla/Object-Detection-Metrics. Accessed: 2022-08-31.

[Hyp,] Hyperparameter settings low augmentation. https://github.com/ultralytics/yolov5/blob/master/data/hyps/hyp.scratch-low.yaml. Accessed: 2022-07-17.

[Ima,] Imagenet. https://www.image-net.org/. Accessed: 2022-07-31.

[lab,] Labelme. https://github.com/wkentaro/labelme. Accessed: 2022-07-31.

[Mat,] Matplotlib. ://matplotlib.org/.

[oly,] Olympus digital camera. https://www.olympus.es/site/es/c/cameras_support/downloads/e_m10_mark_ii_downloads.html. Accessed: 2022-07-17.

[ope,] Opencv. https://opencv.org/. Accessed: 2022-07-31.

[os,] Os. https://docs.python.org/3/library/os.html. Accessed: 2022-07-31.

[Pan,] Pandas. https://pandas.pydata.org/. Accessed: 2022-07-31.

[PIL,] Pillow (pil fork). https://pillow.readthedocs.io/en/stable/. Accessed: 2022-07-31.

[pyc,] Pycharm. https://www.jetbrains.com/help/pycharm/github.html. Accessed: 2022-07-31.

[PyT,] Pytorch. https://pytorch.org/. Accessed: 2022-07-31.

[PyY,] pyyaml. https://pyyaml.org/wiki/PyYAMLDocumentation. Accessed: 2022-07-31.

[son,] Sony digital camera. https://www.sony.co.uk/. Accessed: 2022-07-17.

[tf,] Tensorflow. https://www.tensorflow.org/. Accessed: 2022-07-31.

[ker,] Tensorflow. https://keras.io/. Accessed: 2022-07-31.

[bes,] Tips for best training results. https://github.com/ultralytics/yolov5/wiki/Tips-for-Best-Training-Results. Accessed: 2022-07-17.

[ult,] Train custom data · ultralytics/yolov5 wiki. https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data.

[v7,] V7 blog. https://www.v7labs.com/blog/mean-average-precision#:~:text=Mean%20Average%20Precision(mAP)%20is%20a%20metric%

20used%20to%20evaluate,values%20from%200%20to%201. Accessed: 2022-08-31.

[yol,] Yolo9000. https://jonathan-hui.medium.com/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-fo Accessed: 2022-09-30.

[Algarabel, 2022] Algarabel, A. B. (2022). Development of a damage detection system for wind turbine blades using a siamese neural network. Tratamiento de Imágenes GTI.

[Bochkovskiy et al., 2020a] Bochkovskiy, A., Wang, C., and Liao, H. M. (2020a). Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934.

[Bochkovskiy et al., 2020b] Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020b). Yolov4: Optimal speed and accuracy of object detection.

[Carbery et al., 2018] Carbery, M., O'Connor, W., and Palanisami, T. (2018). Trophic transfer of microplastics and mixed contaminants in the marine food web and implications for human health. *Environment International*, 115:400–409.

[Cerezo, 2020] Cerezo, B. Z. (2020). Clasificación de microplásticos basada en redes neuronales profundas.

[Cristianini and Ricci, 2008] Cristianini, N. and Ricci, E. (2008). *Support Vector Machines*, pages 928–932. Springer US, Boston, MA.

[De Brabandere et al., 2017] De Brabandere, B., Neven, D., and Van Gool, L. (2017). Semantic instance segmentation for autonomous driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 478–480.

[de la Fuente Stranger et al., 2019] de la Fuente Stranger, A., Meruane, V., and Meruane, C. (2019). Hydrological early warning system based on a deep learning runoff model coupled with a meteorological forecast. *Water*, 11.

[Delgado de Santos et al., 2020] Delgado de Santos, P. et al. (2020). Generación sintética de secuencias temporales a través de redes neuronales profundas. Master's thesis.

[DeVries and Taylor, 2017] DeVries, T. and Taylor, G. W. (2017). Dataset augmentation in feature space.

[Gauci et al., 2019] Gauci, A., Deidun, A., Montebello, J., Abela, J., and Galgani, F. (2019). Automating the characterisation of beach microplastics through the application of image analyses. *Ocean Coastal Management*, 182:104950.

[Girshick, 2015] Girshick, R. (2015). Fast r-cnn.

[Girshick et al., 2013] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Gu et al.,] Gu, C., Lim, J., Arbelaez, P., and Malik, J. Recognition using regions.

[Hanvey et al., 2017] Hanvey, J. S., Lewis, P. J., Lavers, J. L., Crosbie, N. D., Pozo, K., and Clarke, B. O. (2017). A review of analytical techniques for quantifying microplastics in sediments. *Anal. Methods*, 9:1369–1383.

[He et al., 2017a] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017a). Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988.

[He et al., 2017b] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017b). Mask r-cnn.

[He et al., 2014] He, K., Zhang, X., Ren, S., and Sun, J. (2014). Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[Huang et al., 2017] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269.

[Huang et al., 2020] Huang, Z., Wang, J., Fu, X., Yu, T., Guo, Y., and Wang, R. (2020). Dc-spp-yolo: Dense connection and spatial pyramid pooling based yolo for object detection. *Information Sciences*, 522:241–258.

[Jambeck et al., 2015] Jambeck, J. R., Geyer, R., Wilcox, C., Siegler, T. R., Perryman, M., Andrady, A., Narayan, R., and Law, K. L. (2015). Plastic waste inputs from land into the ocean. *Science*, 347(6223):768–771.

[Kane et al., 2020] Kane, I. A., Clare, M. A., Miramontes, E., Wogelius, R., Rothwell, J. J., Garreau, P., and Pohl, F. (2020). Seafloor microplastic hotspots controlled by deep-sea circulation. *Science*, 368(6495):1140–1145.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham. Springer International Publishing.

[Lin and Jung, 2017] Lin, Y.-P. and Jung, T.-P. (2017). Improving eeg-based emotion classification using conditional transfer learning. *Frontiers in Human Neuroscience*, 11.

[Liu et al., 2018a] Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., and Pietikäinen, M. (2018a). Deep learning for generic object detection: A survey.

[Liu et al., 2018b] Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018b). Path aggregation network for instance segmentation.

[Liu et al., 2018c] Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018c). Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534.

[Liu et al., 2020] Liu, Z., Gao, G., Sun, L., and Fang, Z. (2020). Hrdnet: High-resolution detection network for small objects.

[Long et al., 2020] Long, X., Deng, K., Wang, G., Zhang, Y., Dang, Q., Gao, Y., Shen, H., Ren, J., Han, S., Ding, E., and Wen, S. (2020). Pp-yolo: An effective and efficient implementation of object detector.

[Lorenzo-Navarro et al., 2020] Lorenzo-Navarro, J., Castrillón-Santana, M., Santesarti, E., De Marsico, M., Martínez, I., Raymond, E., Gómez, M., and Herrera, A. (2020). Smacc: A system for microplastics automatic counting and classification. *IEEE Access*, 8:25249–25261.

[Lorenzo-Navarro et al., 2021] Lorenzo-Navarro, J., Castrillón-Santana, M., Sánchez-Nielsen, E., Zarco, B., Herrera, A., Martínez, I., and Gómez, M. (2021). Deep learning approach for automatic microplastics counting and classification. *Science of The Total Environment*, 765:142728.

[Mcculloch and Pitts, 1943] Mcculloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147.

[Mukhanov et al., 2019] Mukhanov, V., Daria, L., Sakhon, Y., Bagaev, A., Veerasingam, S., and Venkatachalapathy, R. (2019). A new method for analyzing microplastic particle size distribution in marine environmental samples. *Ecologica Montenegrina*, 23:77–86.

[Norouzi et al., 2020] Norouzi, S., Fleet, D., and Norouzi, M. (2020). Exemplar vaes for exemplar based generation and data augmentation.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359.

[Parico and Ahamed, 2021] Parico, A. I. and Ahamed, T. (2021). Real time pear fruit detection and counting using yolov4 models and deep sort. *Sensors*, 21:4803.

[Pham et al., 2020] Pham, C., Pereira, J. M., Frias, J., Ríos, N., Carriço, R., Juliano, M., and Rodríguez, Y. (2020). Beaches of the azores archipelago as transitory repositories for small plastic fragments floating in the north-east atlantic. *Environmental Pollution*, page 114494.

[Prata et al., 2019] Prata, J. C., Reis, V., Matos, J. T., da Costa, J. P., Duarte, A. C., and Rocha-Santos, T. (2019). A new approach for routine quantification of microplastics using nile red and automated software (mp-vat). *Science of The Total Environment*, 690:1277–1283.

[Rawat and Wang, 2017] Rawat, W. and Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29:1–98.

[Redmon, 2016] Redmon, J. (2013–2016). Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`.

[Redmon et al., 2015] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2015). You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640.

[Redmon and Farhadi, 2017] Redmon, J. and Farhadi, A. (2017). Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525.

[Redmon and Farhadi, 2018] Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement.

[Ren et al., 2015] Ren, S., He, K., Girshick, R. B., and Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham. Springer International Publishing.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA.

[Russakovsky et al., 2014] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2014). Imagenet large scale visual recognition challenge.

[Seibold et al., 2022] Seibold, M., Hoch, A., Farshad, M., Navab, N., and Fürnstahl, P. (2022). Conditional generative data augmentation for clinical audio datasets.

[Shaha and Pawar, 2018] Shaha, M. and Pawar, M. (2018). Transfer learning for image classification. *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 656–660.

[Shijie et al., 2017] Shijie, J., Ping, W., Peiyi, J., and Siping, H. (2017). Research on data augmentation for image classification based on convolution neural networks. *2017 Chinese Automation Congress (CAC)*, pages 4165–4170.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

[Uijlings et al., 2013] Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., and Smeulders, A. W. M. (2013). Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171.

[Wang et al., 2019] Wang, C., Liao, H. M., Yeh, I., Wu, Y., Chen, P., and Hsieh, J. (2019). Cspnet: A new backbone that can enhance learning capability of CNN. *CoRR*, abs/1911.11929.

[Wegmayr et al., 2020] Wegmayr, V., Sahin, A., Sæmundsson, B., and Buhmann, J. M. (2020). Instance segmentation for the quantification of microplastic fiber images. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 2199–2206.

[Wong et al., 2020] Wong, J. K. H., Lee, K. K., Tang, K. H. D., and Yap, P.-S. (2020). Microplastics in the freshwater and terrestrial environments: Prevalence, fates, impacts and sustainable solutions. *Science of The Total Environment*, 719:137512.

[Yu et al., 2016] Yu, J., Jiang, Y., Wang, Z., Cao, Z., and Huang, T. S. (2016). Unitbox: An advanced object detection network. *CoRR*, abs/1608.01471.

[Zoph et al., 2020] Zoph, B., Cubuk, E. D., Ghiasi, G., Lin, T.-Y., Shlens, J., and Le, Q. V. (2020). Learning data augmentation strategies for object detection. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII*, page 566–583, Berlin, Heidelberg. Springer-Verlag.