Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

"Automatic identification of microorganisms in microscopic imaging"

Projeto

Telmo Alexandre Pires Marques

Mestrado em Engenharia Electrotécnica

Tomar/ Novembro/ 2023

Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Telmo Alexandre Pires Marques

"Automatic identification of microorganisms in microscopic imaging"

Projeto

Orientado por:

Prof. Pedro Daniel Frazão Correia, Instituto Politecnico de Tomar
Prof. Manuel Fernando Martins de Barros, Instituto Politecnico de Tomar

Projeto apresentada ao Instituto Politécnico de Tomar
para cumprimento dos requisitos necessários
à obtenção do grau de Mestre em Engenharia Electrotécnica

To my family, for all the love and support.

ABSTRACT

Microorganisms play a fundamental role in the world around us, affecting human life in more ways than we can imagine. The ability to visualize and correctly identify these tiny creatures under a microscope is paramount. Traditional techniques are time-consuming, costly, labor-intensive, and prone to misclassification and low accuracy, hence the importance of an automatic classification system based on convolutional neural networks. Computer models trained on the right type of dataset images have the potential to provide accurate and efficient detection and classification of microorganisms.

This report presents an image based real time microorganism classification system, using deep learning methods, namely YOLOv5. The implementation is presented and discussed in detail, explaining the whole process from the initial dataset consisting of 168 E. coli images captured using traditional microscopy and a digital camera, through dataset augmentation, image enhancement, techniques used to expand the dataset, and the various iterations of training required to obtain the final trained weights file.

A real time image classification prototype was developed, using an optical kit composed by a high amplification optical tube and high sensibility and spectral domain CMOS sensors, linked to a dedicated deep learning application, Nvidia Jetson Nano computer running the trained state-of-the-art YOLOv5 architecture.

A baseline of test results was maintained to measure progress. The initial scores, as measured by YOLO's output metrics, were very low. Over the course of many changes and adjustments to the dataset and some training parameters, we were able to increase the overall classification performance metrics. This increase in performance is significant and indicates that there is potential for further development of the model and setup.

The presented work is a solid base to improve classification algorithms and also the image acquisition and classification prototype, in portable applications without colored preparations and the traditional laboratorial setup.

Keywords: Deep learning, YOLO, convolution neural networks, classification, microbiology

RESUMO

Os microrganismos desempenham um papel fundamental no mundo que nos rodeia, afetando a vida humana de mais formas do que podemos imaginar. A capacidade de visualizar e identificar corretamente essas pequenas criaturas ao microscópio é fundamental. As técnicas tradicionais são demoradas, caras, trabalhosas e propensas a erros de classificação e baixa precisão, daí a importância de um sistema de classificação automática baseado em redes neurais convulsionais. Modelos treinados com imagens de boa qualidade têm o potencial de fornecer a deteção e classificação precisa e eficiente de microrganismos.

O presente relatório apresenta o desenvolvimento de um sistema de classificação automática de microorganismos, em tempo real, usando processamento de imagem baseada em métodos de aprendizagem profunda, nomeadamente o modelo YOLOv5. É apresentada e discutida em detalhe, a sua implementação, explicando todo o processo desde o conjunto de dados inicial composto por 168 imagens de E. coli capturadas usando microscopia tradicional e uma câmera digital, passando pelo aumento do conjunto de dados, melhoramento de imagem, técnicas usadas para expandir o conjunto de dados. São apresentadas as várias iterações de treino necessárias para obter o arquivo final referente aos melhores pesos dos parâmetros de aprendizagem.

Foi desenvolvido um protótipo de processamento de imagem, constituído por um kit ótico constituído por tubo ótico de elevadas amplificações e sensores CMOS de elevada sensibilidade e domínio espectral, ligadas a um minicomputador Nvidia Jetson Nano, concebido para aplicações em tempo real de aprendizagem profunda, dedicado ao algoritmo de classificação YOLOv5.

Os resultados iniciais funcionaram como base de referência para comparar os trabalhos desenvolvidos. As pontuações iniciais, medidas pelas métricas do YOLO terem sido muito baixas numa fase inicial, conseguiu-se aumentar o desempenho do modelo de classificação, com base nos ajustes do conjunto de dados e em alguns parâmetros de treino. Este aumento no desempenho é significativo e indica que há potencial para um maior desenvolvimento do modelo implementado.

O trabalho desenvolvido permite uma base sólida para melhoria dos algoritmos de classificação e do protótipo de aquisição para aplicações onde não seja necessária coloração das preparações e possa utilizado em aplicações portáteis sem a preparação laboratorial tradicional.

Palavras-chave: aprendizagem profunda, YOLO, redes neuronais convulsionais, classificação, microbiologia

ACKNOWLEDGMENT

To Instituito Politécnico de Tomar, where my journey begins. I would like to thank everyone who is a part of this beautiful institution, especially my former professors, who in one way or another have helped me through the many challenges I have faced throughout the years, making me a better and stronger person.

I would like to thank both Professor Pedro Daniel Frazão Correia and Professor Manuel Fernando Martins de Barros for their invaluable guidance and feedback throughout this project. Their knowledge and expertise were fundamental.

To the team behind the OMRisk project (UIDB/05567/2020/04) of the Smart Cities Research Center (Ci2), to whom I extend my appreciation, with special thanks to Professor Rui Gonçalves, Professor Dina Mateus and Professor Henrique Pinho for making this research possible and for their contributions to the development of the work.

Thank you.

Contents

1 Introduction 1
1.1 Motivation 2
1.2 Objectives 3
1.3 Contributions 3
2 Deep Neural Networks applied to image-based classification 7
2.1 Convolutional Neural Networks 7
2.2 Data Augmentation 9
2.3 Transfer learning 10
2.4 YOLO - You Only Look Once 11
2.5 R-CNN – Region based Convolutional Neural Network 13
2.5.1 Region proposal module 13
2.5.2 Feature extraction module 14
2.5.3 Classification module 15
2.5.4 RCNN Performance 15
2.5.5 RCNN Drawbacks 15
2.5.6 YOLOv8 16
2.5.7 Image preprocessing 17
2.5.8 Real image de-noising 17
2.5.9 Super resolution 18
3 Deep learning image classification applied to microorganisms 21
3.1 The early days of detection 21
3.2 The modern age of detection 22
3.3 The current state of the art 22
4 Dataset, image enhancement, and model training 27
4.1 The dataset 28
4.2 Annotation 28
4.2.1 Augmentation 30
4.3 Image enhancement 31
4.4 Image cropping 32
4.5 Model training 33
4.5.1 Google Colab Notebook 33
4.6 Training results 35

4.6.1 Metric descriptions....35
4.6.2 Overall training results....38
4.6.3 Training results pertaining to E-coli_2.v3.Rearrg. 39
5 Implementation with Nvidia Jetson Nano 45
5.1 Nvidia Jetson Nano 46
5.2 Sony IMX477 camera sensor 47
5.3 YOLOv5 and required dependencies 48
6 Conclusion 53
References 55

List of Figures

Figure 1 – Prototype setup & components. 4
Figure 2 – Functional flow diagram. 5
Figure 3 – Convolutional neural network (CNN) layout [16]. 7
Figure 4 – 2D Convolution [17]. 8
Figure 5 – Pooling layer architecture [18]. 9
Figure 6 – Flattened vector [19]. 9
Figure 7 – Data augmentation process [21]. 10
Figure 8 – Transfer learning [23]. 11
Figure 9 – YOLOv5 architecture [27]. 12
Figure 10 – YOLOv5 rankings based on COCO dataset [28]. 12
Figure 11 – Region based Convolutional Neural Network [30]. 13
Figure 12 – Example of image segmentation and region proposal [31]. 14
Figure 13 – Warping of region proposal [32]. 14
Figure 14 – Testing comparison [36]. 15
Figure 15 – YOLO comparison [46]. 16
Figure 16- YOLOv8 subversions performance information [32]. 17
Figure 17 – (a) Image with noise (b) Image without noise [35]. 18
Figure 18 – (a) Low resolution (b) High resolution [35]. 19
Figure 19 – Example of segmentation-based detection [38]. 21
Figure 20 – Milestones of deep learning object detection [37]. 22
Figure 21 – Implementation and testing procedure. 27
Figure 22 – Roboflow annotation workspace [41]. 29
Figure 23 – Labelling example. Class 1: Bacilli; Class 2: Cocci; Class 3: Cluster. 29
Figure 24 – Augmentation example. 30
Figure 25 – Yolov5s file structure. 31
Figure 26 – Before and after de-noising image enhancement. No noticeable improvements. 31
Figure 27 – Removal of the bottom 128 pixels. 32
Figure 28 – Cropping process. Division of each image into 12x 640x640 pixel images. 33
Figure 29 – Python code used to decompress dataset file. 34
Figure 30 – Configuration file data.yaml contents. 34
Figure 31 – Configurable parameters of the YOLO training process. 34
Figure 32 – IoU. A - Ground Truth Bounding Box; B - Predicted Bounding Box [50]. 36
Figure 33 – Example of IoU scores [49]. 37

Figure 34 – YOLOv5 overall training results. 38
Figure 35 – Example of removed images. (a) Blurry. (b) Captured in movement. 39
Figure 36 – Plotted training results pertaining to E-coli_2.v3.Rearrg. 40
Figure 37 – Confusion matrix results pertaining to E-coli_2.v3.Rearrg. 41
Figure 38 – Validation dataset results pertaining to E-coli_2.v3.Rearrg. 42
Figure 39 – Optical microscope with 5x lens (Left), and Nvidia Jetson Nano (Right). 45
Figure 40 – Nvidia Jetson Nano connected to camera which is mounted to the microscope. . 45
Figure 41 – Nvidia Jetson Nano hardware architecture and installed software. 46
Figure 42 – Sony IMX477 camera sensor. 47
Figure 43 – Command to view video stream from camera. 48
Figure 44 – PIP download and installation. 48
Figure 45 – Installation command for Python 3.8 49
Figure 46 - Installation command for the virtual environment 49
Figure 47 – Creation and activation of virtual environment. 49
Figure 48 – YOLOv5 install command. 49
Figure 49 – Sequence of commands to comment torch and torchvision. 50
Figure 50 – Installation of contents of requirement.txt and additional dependencies. 50
Figure 51 – Installation of the Pytorch PIP wheel. 50
Figure 52 – Installation of the Torchvision PIP wheel. 51
Figure 53 – detect.py with best weights file and camera as source. 51
Figure 54 – Real time image classification results - 1 51
Figure 55 – Real time image classification results - 2 52

List of Tables

Table 1 – Training results pertaining to E-coli_2.v3.Rearrg. 39

Abbreviations and Symbols

AI	Artificial Intelligence
CNN	Convolutional Neural network
COCO	Common Objects in Context
CPU	Central Processing Unit
DNN	Deep Neural Network
DP	Deep Learning
DPM	Deformable Part Models
eMMC	Embedded Multi-Media Card
Fast R-CNN	Fast Region Based Convolution Neural Network
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
IoU	Intersection Over Union
mAP	Mean Average Precision
PIP	Package Installer Program
R-CNN	Region Based Convolutional Neural Network
SSD	Solid State Drive
SVM	Support Vector Machine
VJ	Viola Jones
YOLO	You Only Look Once

1 Introduction

From the beginning of the digital age, computer programmers and enthusiasts alike have dreamt of coding a machine capable of performing tasks with some level of intelligence. Great developments in the field of artificial intelligence and machine learning have been achieved over the course of the past decades. Private researchers and corporations continuously strive to develop cutting edge software capable of solving the most challenging problems faced by mankind. Today, more than ever, we can witness first-hand the benefits of AI, from self-driving cars to chat bots capable of understanding Human language and responding accordingly, the possibilities are truly endless, limited only to our imaginations.

Machine learning, by definition, is a field of inquiry devoted to understanding and building methods that "learn", or in order words, methods that process data to improve performance on some set of specific tasks [1]. At their core, these algorithms are coded models that are trained using large amounts of information known as datasets. A dataset is comprised of the selected information which the user pretends to target during the machine learning process. After successfully training a model, a machine running the aforementioned model should be capable of making predictions or decisions without being explicitly programmed to do so.

These machine learning algorithms are used in a wide variety of applications, such as medicine, speech recognition and computer vision, where conventional algorithms are less efficient or uncapable of performing the required task [2].

The scientific discipline of computer vision is focused on the theory behind the artificial systems capable of extracting real world information from still images or video and processing that same information through a coded model in order to produce a prediction or decision. Computer vision can be further divided into sub-domains such as, but not limited to, image restoration, object detection and/or recognition, video tracking, learning, indexing, event detection and visual servoing [4,5].

Through the sub-domain of object detection and recognition, many advances have been possible in various scientific fields. Medicine is an example where computer learning models are trained to detect cancer cells in very early stages of development. In most of the cases, these cells are still invisible to the naked eye. The capability of correctly identifying cancer cells in such a minute state greatly increases patient survival rates [3].

Autonomous and self-driving vehicles are tangible examples of the benefits provided by the advances in computer vision technology. Heavy players in the car manufacturing industry such

as Tesla, Mercedes, and Volkswagen are investing heavily in the research and development of this technology, implementing machine vision models into road legal vehicles capable of navigating routes and obstacles in a safe and trouble-free manner. Vehicles with this technology use convolutional neural networks trained with large datasets of objects normally found on or around roadways. Examples of the application of computer visions to self-driving cars are lane assist features, self-parking, automatic breaking when obstacles are detected, and complete autonomous driving without human interaction [6].

As mentioned previously, computer vision models can be applied and trained to solve a vast array of problems over diverse areas of industry and life.

Microbiology is the science that studies life in its most minuscule form, microorganisms. Traditionally, microorganism cultures are derived from a sample swab that is placed in a microbial substrate and left to grow over time in an incubator. Aside from being a time-consuming process, once grown, the cultures are visually inspected, and colonies are identified and counted to derive conclusions and confirm assumptions [7,8].

Computer vision can greatly benefit the science of microbiology by cutting down on expensive laboratory assay equipment, and more importantly reducing the amount of time required in producing results [9].

Escherichia-coli is a rod-shaped microorganism normally found in the lower intestine of warm-blooded organisms. Food or water contamination by a dangerous strain of E-coli, when ingested by a Human, can lead to severe illness and in extreme cases kidney failure resulting in death [10,11].

1.1 Motivation

Traditional laboratory cultivation of in vitro microorganisms is a costly and time-consuming process that requires a highly trained human to correctly and precisely classify microorganisms present in the harvested sample. A longer incubation period or incorrect classification assumption may put at risk public health and safety, which can potentially lead to illness or other even more extreme circumstances.

A fully automated system capable of visually detecting and correctly classifying a microbiological sample in real time could offer a precise classification solution in a fraction of the time it would traditionally require. A system like this could be installed in many locations, such as, laboratories to expedite analyses, potable water reservoirs and main lines for early detection of potential drinking water contamination, hospitals, and other large public gathering

facilities. It is then easy to understand the importance of a rapid and precise computer model trained in E-coli detection through either still images or streaming video.

1.2 Objectives

The aim of this thesis is to develop a computer vision model based on a deep neural network, trained, and tested to an acceptable degree for the detection of E-coli bacteria.

The system must be capable of analyzing still images of microorganisms captured in real time. To achieve this, a computer processing unit must be connected to a camera and optical microscope set up with sufficient magnification power to clearly visualize the target microorganisms.

The deep neural network model must be trained using a dataset consisting of images with the same characteristics as those captured in the real-world application.

1.3 Contributions

An image dataset was created through the acquisition of 168 still photographs of e-coli bacteria. A traditional optical microscope equipped with a digital photographic camera was the adopted setup used to capture the dataset pictures. The images were annotated into three classes: cocci, bacilli, and cluster. These classes identify the different morphology of the e-coli bacteria to be classified. To increase the size of the dataset and turn the training process more efficient, data augmentation techniques such as rotations and flips of the images were used.

Image filtering techniques were applied to the dataset in an effort to increase the overall quality of the images and ultimately, training and classification. Moving forward, only the methods yielding the best results were considered.

Numerous iterations of models were trained on the dataset using the YOLO [24], in particular YoloV5 [12] algorithm developed by Ultralytics [13]. The whole process was conducted through the Google Colaboratory platform where it is possible to make use of powerful graphic processing units that can expedite the process much quicker than a conventional standalone computer [14].

A real time microbiological classification setup was implemented based on the Nvidia Jetson Nano platform. This platform is a small, powerful computer with an integrated graphics processing unit specifically designed to run deep neutral networks efficiently. The installed operating system was Linux 18.04 complete with the Nvidia Jetpack version 4.68.

The basis of the setup revolves around viewing the microorganisms in the most efficient manner possible so that the classification can be successful. Figure 1 and Figure 2, show the setup prototype and functional flow diagram respectively. Firstly, the microscope must be positioned directly over the lamella containing the sample microorganism. An external light source is required for proper optical viewing. A proper microscope support system was installed to avoid any movement or vibrations while in use. The digital camera is threaded into the microscope's top end and connected to the Nvidia Jetson Nano via a flat ribbon cable. Then, the computer is connected to a viewing monitor using a high-definition video connection.

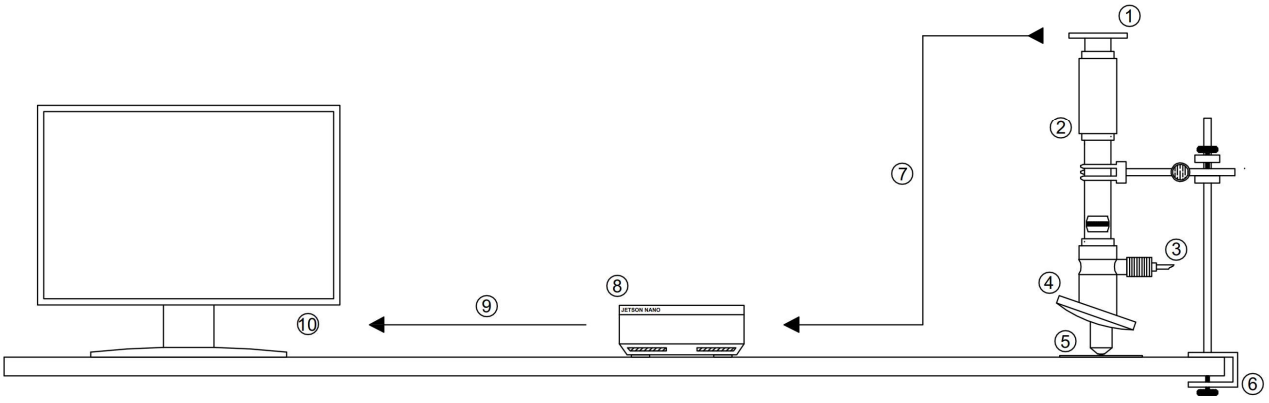


Figure 1 – Prototype setup & components.

- The following is a description of the setup components illustrated in Figure 1.

1 – Digital camera
2 – Optical lens
3 – External light source
4 – Amplification lens
5 – Sample microorganism
6 – Microscope support
7 – Camera ribbon cable connected to Jetson
8 – Nvidia Jetson Nano
9 – HDMI connection to monitor
10 – Monitor for viewing

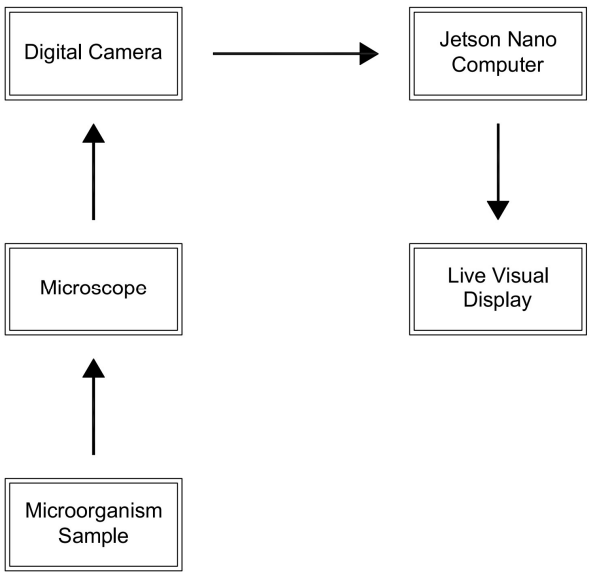


Figure 2 – Functional flow diagram.

2 Deep Neural Networks applied to image-based classification

This chapter aims to introduce and explain the key theoretical fundamentals and methods involved when applying deep neural network-based image classification techniques.

2.1 Convolutional Neural Networks

Deep models are also known as neural networks with deep structures. These models date back to the 1940s, when they were studied and developed with the intention of simulating the human brain system in solving general learning problems. Convolutional neural networks (CNNs) are in fact deep neural networks (DNNs) optimized for performance in computer image processing tasks [36].

Four basic layers make up a CNN, the convolutional layers, pooling layers, flatten layer, and the fully connected layers. By stacking the layers together, the architecture of a CNN in its most basic form is created, as described in Figure 3.

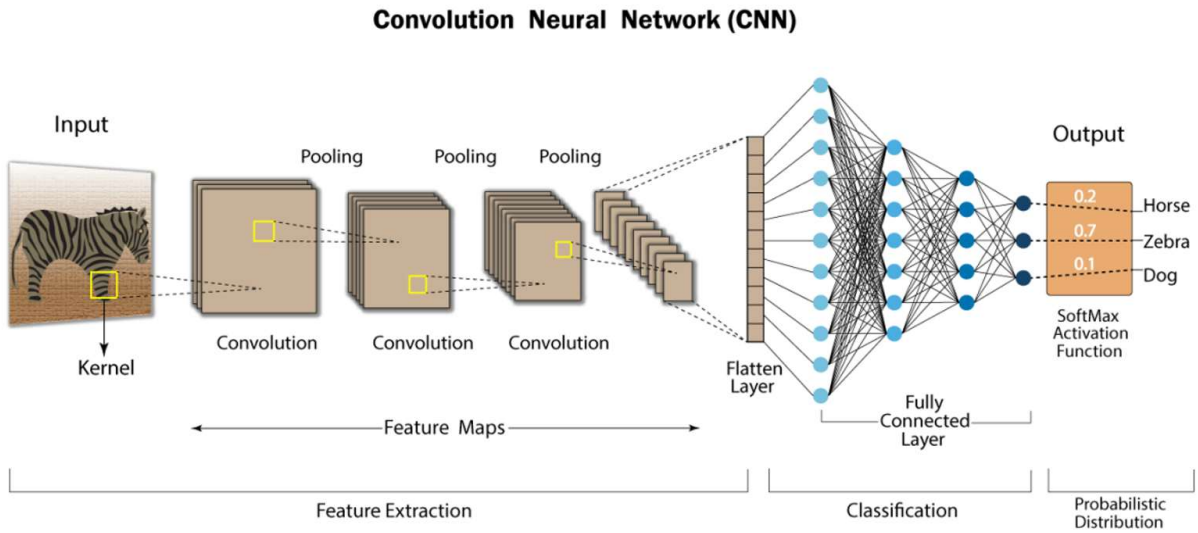


Figure 3 – Convolutional neural network (CNN) layout [16].

The convolutional layers are the first ones in the stack and are responsible for receiving the input data (still images or streaming video) and extracting the various features. To do so a mathematical operation called convolution is performed. Convolution is a special linear operation where two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other.

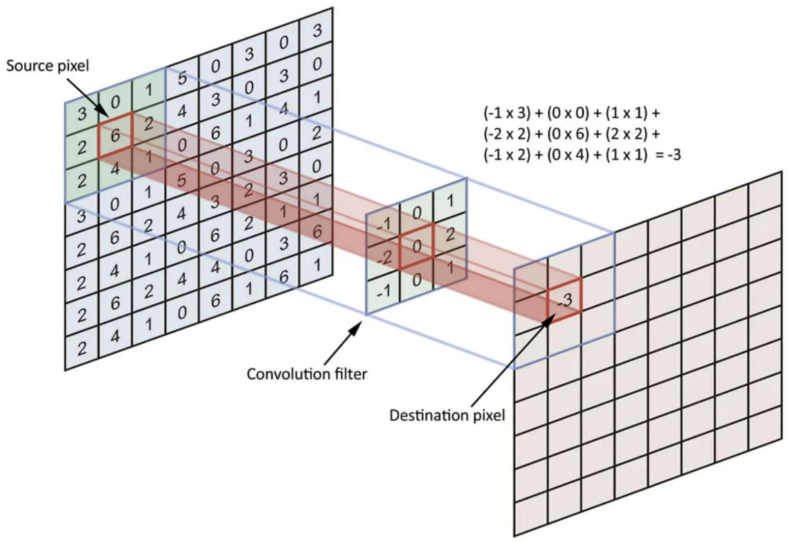


Figure 4 – 2D Convolution [17].

The convolutional layers are capable of detecting patterns in imagens using filters. One example of patterns in an image could be edges or straight lines, in this case the applied filter would be an edge detection filter. Filters that detect simple geometric patterns are located closer to the top layers of a CNN, as opposed to those that detect specific or more complex objects, these are located deeper into the network.

More specifically, the convolution is performed between the input data and a filter of a particular size MxM. By sliding the filter over the input data, the convolution dot product is taken between the filter and the parts of the input image covered by the filter's size. Figure 4 illustrates the convolution process between two matrices. This operation results in a feature map which contains information about the image such as corners, edges, straight lines, etc.

The second layer in the stack is the pooling layer. Its main purpose is to reduce the size of the convolved feature map resulting in a reduction of the computational cost. This is accomplished by decreasing the connections between the layers, independently operating on each feature map. In essence, the pooling layer summarizes the features generated by a convolution layer. Once this operation is complete, the resulting feature map is then transferred to another layer to learn several other features of the input data.

Pooling may be performed in one of three ways, maximum, minimum or average pooling. Using maximum pooling as an example, the feature map is divided into sections according to the desired pool size. The maximum value contained inside the pool size area of the feature map will be selected and copied over to the max pooling matrix. The stride parameter defines how many feature map slots may be moved for each iteration of pooling. This operation will

repeat until the complete feature map has been scanned and the new summarized pooling matrix is populated. The following Figure 5 describes the pooling procedure.

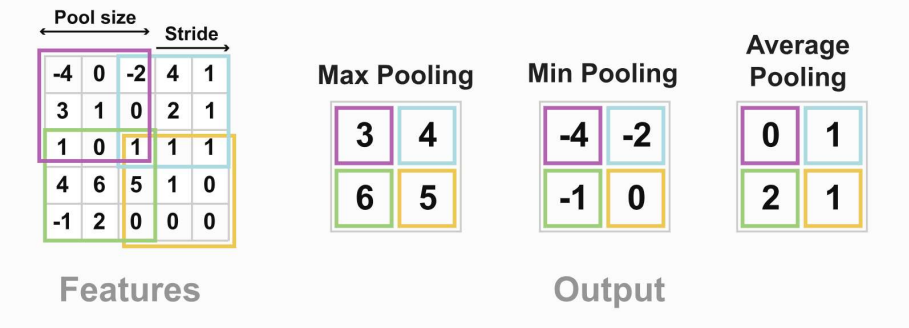


Figure 5 – Pooling layer architecture [18].

The flatten layer is the third in the stack and serves as a transition between the previous layers and the fully connected layer. Its task is to transform a multidimensional input into a one-dimensional flattened vector. The following Figure 6 illustrates the flattening process.

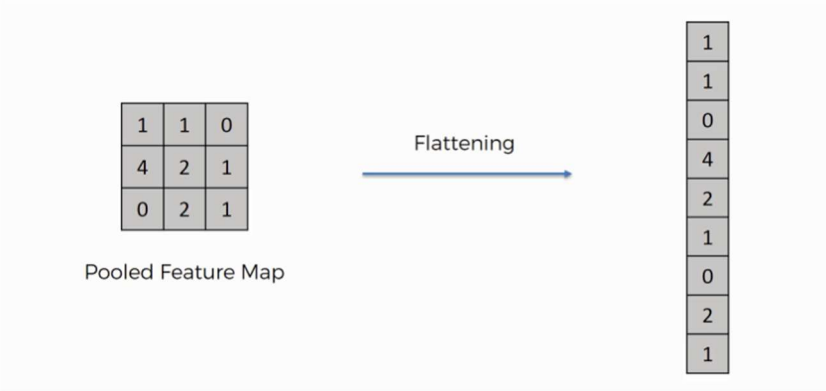


Figure 6 – Flattened vector [19].

The last stage is the fully connected layer, and it is responsible for classifying and labelling the input data. This layer connects the information extracted from the previous steps, stored in the flattened vector, to the output layer and classifies the input according to the class designations.

2.2 Data Augmentation

Data augmentation is a technique used to artificially increase the training set by creating modified copies of the original data. Some of the most common augmentation techniques are [20]:

- Geometric transformations consisting of rotating and flipping.

- Adjusting brightness, contrast, and color properties.
- Adding a matrix of random values to an image resulting in a noisy image.
- Cutting and cropping the image into different portions.
- Zooming into or out of an image.

Data augmentation is normally chosen when the original dataset is too small or when the intent is to increase the model's accuracy. This process is applied prior to the training process, this way the new dataset is trained with a greater and more varied amount of information, thus contributing to better overall results. The Figure 7 describes the basic steps required when performing data augmentation.

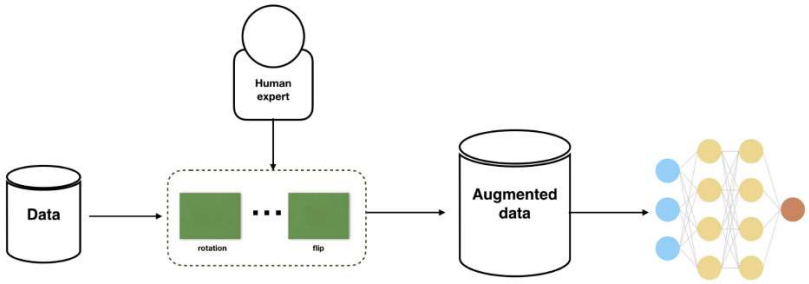


Figure 7 – Data augmentation process [21].

2.3 Transfer learning

Transfer learning works by training a model on a large dataset such as the Common Object in Context (COCO) dataset, and then using those weights as the initial starting point in a new classification problem [22]. This speeds up progress and improves performance when training a new model, so it's primarily used whenever time is a factor or the resources required for training a model aren't feasible.

Normally, only the weights of the convolutional layers are used and not the complete network. This proves its effectiveness given that many datasets share similar spatial characteristics that are better learned with big data.

The following Figure 8 illustrates how, by applying transfer learning, previously learned knowledge from a different model is easily applied to another model tasked with a different application.

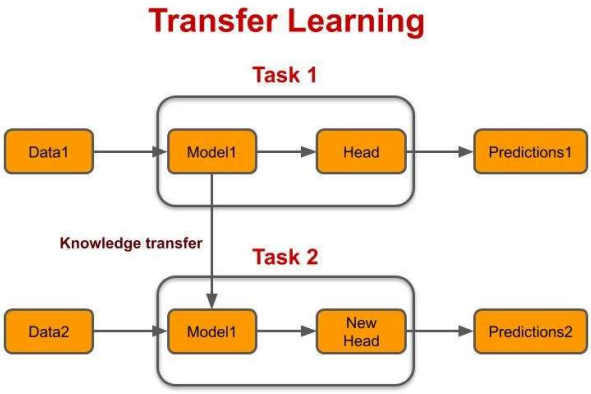


Figure 8 – Transfer learning [23].

2.4 YOLO - You Only Look Once

Single-shot object detection uses a single pass through the input image to make predictions about the presence and location of objects in the image. It processes an entire image in a single pass, making them computationally efficient and ideal for real time classification [24].

You Only Look Once, (YOLO) is an easy to use single-shot detector that uses a fully convolutional neural network to process an image. YOLO receives and partitions the input image into a NxN grid and predicts the bounding box coordinates and class confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and how accurate it thinks the prediction is. There may be multiple bounding boxes per grid cell, however only the bounding boxes with the highest probability of containing a desired class are selected. The final classification of the bounding box coordinates and class identification occurs in the final fully connected layer of the YOLO CNN framework [25].

The YOLOv5 model is the result of continued upgrades to the original framework. This version is faster, more efficient, and simpler to install and use than its predecessors.

Three main parts make up YOLOv5's architecture [26]:

Backbone – Refers to the network's main body. This is where the feature extraction process takes place.

Neck – Connects the backbone and the head. Here, concatenation and up-sampling fuse feature maps from different stages.

Head – Responsible for producing the final output. Objects are classified with bounding boxes and class designations.

The following Figure 9 illustrates the YOLOv5 architecture.

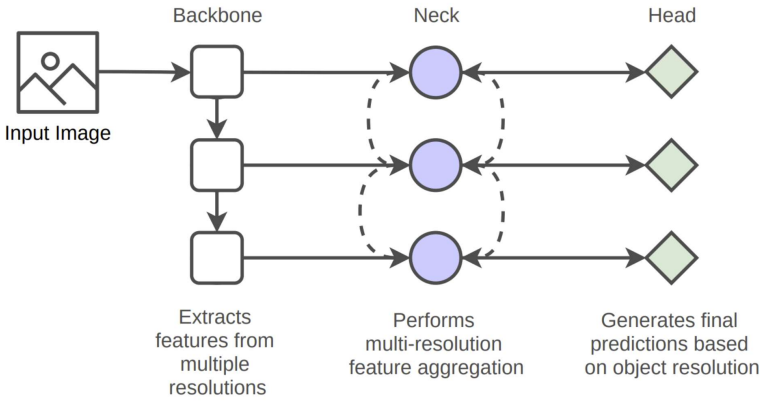


Figure 9 – YOLOv5 architecture [27].

YOLOv5 is further divided into sub-variations which include the following:

- YOLOv5s - Small
- YOLOv5m - Medium
- YOLOv5l - Large
- YOLOv5x - Extra-large

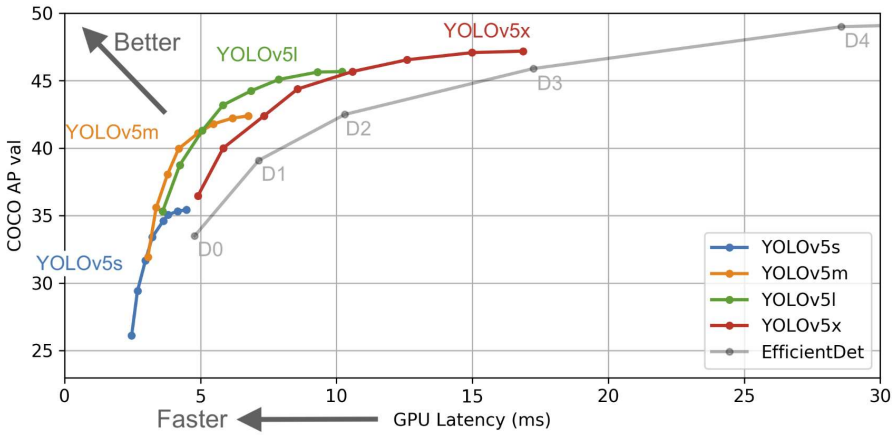


Figure 10 – YOLOv5 rankings based on COCO dataset [28].

EfficientDet is a type of object detection model that uses various backbone optimizations and tweaks to uniformly scale resolution, depth, and width of all the backbones, feature networks and box predictions at the same time [29]. This approach allows the EfficientDet family of models to achieve better accuracy and efficiency over other detections models.

Figure 10 denotes the performance comparison between the YOLOv5 versions and the EfficientDet object detection model. GPU latency measures the average inference time per image for the COCO val2017 dataset. The YOLOv5 family presents faster inference speed, however the EffcientDet model achieves higher precision values [28]. It is also worth noting that the YOLOv5s version is faster at performing inference compared to the YOLOv5x, however the latter outperforms it in average precision. These features must be considered when deciding which version is best suited for the task at hand.

2.5 R-CNN – Region based Convolutional Neural Network

Region based convolutional neural networks (RCNN) is a type of deep learning architecture used for object detection in computer vision tasks. RCNN has been recognized as one of the first classification models to combine both convolutional neural networks and the region-based approach to detect objects, and by doing so, has greatly contributed to advances in this field. The RCNN model can be divided into three main stages, the region proposal module, feature extraction module and the classification module [36], as demonstrated in Figure 11.

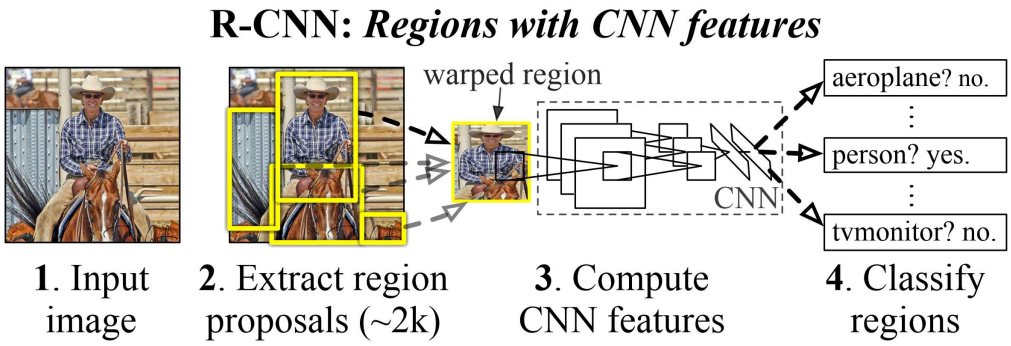


Figure 11 – Region based Convolutional Neural Network [30].

2.5.1 Region proposal module

The RCNN algorithm adopts an external source such as Selective Search which starts by dividing the input image into multiple regions that are referred to as region proposals or candidates. This step is responsible for generating a set of potential regions within the image that are likely to contain objects of interest. The number of proposed regions can be as high as 2000 for each input image. Selective Search operates by applying over segmentation techniques of an image and combining key similarity components such as color, texture, and size to create

the diverse regions of the image [30,36]. The following Figure 12 illustrates region proposal based on image segmentation techniques.

Figure 12 – Example of image segmentation and region proposal [31].

2.5.2 Feature extraction module

In this next step a convolutional neural network is applied to each region proposal to create a 4096-dimensional feature vector as the final representation [36].

An example of a convolutional neural network used with RCNN is AlexNet [47]. Comprised of 5 convolutional layers and 3 fully connected layers, AlexNet requires inputs with a fixed size of 277x277 pixels. Resizing of proposed region images must be carried out prior to applying the CNN.

The simplest way of converting the size of a proposed region input and guaranteeing compatibility with the CNN is by warping all of the pixels into a tight bounding box of the required size. This will, in many cases, distort the proposed region image, however this is expected. The following Figure 13 demonstrates image warping.

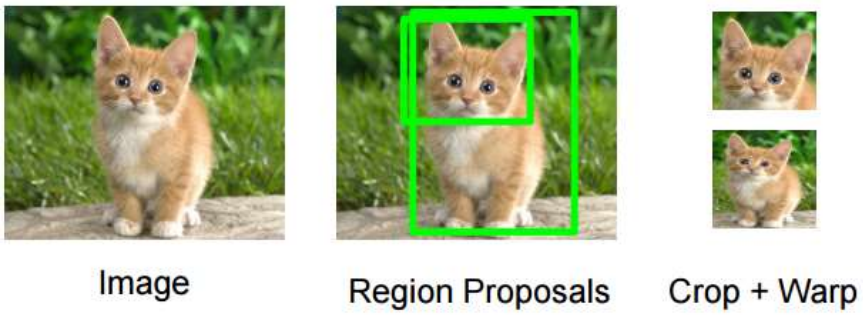


Figure 13 – Warping of region proposal [32].

Finally, the applied CNN will create a high-level feature representation for each resized region proposal [36].

2.5.3 Classification module

To classify the feature vectors created in the previous step a support vector machine (SVM) classifier is applied [48]. A SVM is a type of supervised learning algorithm used in machine learning to solve classification and regression tasks. There is one SVM for each object class, which means that for each feature vector the number of possible outputs is equal to the number of classes.

The output is a confidence score that indicates how confidante the SVM is that the feature vector represents the chosen class. Images with the highest scores will be chosen [34,36].

2.5.4 RCNN Performance

Figure 14 denotes the performance results of RCNN with selective search compared to other models such as YOLO. Clearly, RCNN ranks at the lowest score of the list, which is expected considering it is an older model. At the time of testing, YOLOv2 tested quite well at fifth best overall.

Methods	Trained on	mAP(%)	Test time(sec/img)	Rate(FPS)
SS+R-CNN [15]	07	66.0	32.84	0.03
SS+SPP-net [64]	07	63.1	2.3	0.44
SS+FRCN [16]	07+12	66.9	1.72	0.6
SDP+CRC [33]	07	68.9	0.47	2.1
SS+HyperNet* [101]	07+12	76.3	0.20	5
MR-CNN&S-CNN [110]	07+12	78.2	30	0.03
ION [95]	07+12+S	79.2	1.92	0.5
Faster R-CNN(VGG16) [18]	07+12	73.2	0.11	9.1
Faster R-CNN(ResNet101) [18]	07+12	**83.8**	2.24	0.4
YOLO [17]	07+12	63.4	**0.02**	**45**
SSD300 [71]	07+12	74.3	**0.02**	**46**
SSD512 [71]	07+12	76.8	0.05	19
R-FCN(ResNet101) [65]	07+12+coco	83.6	0.17	5.9
YOLOv2(544*544) [72]	07+12	78.6	0.03	40
DSSD321(ResNet101) [73]	07+12	78.6	0.07	13.6
DSOD300 [74]	07+12+coco	81.7	0.06	17.4
PVANET+ [116]	07+12+coco	**83.8**	0.05	21.7
PVANET+(compress) [116]	07+12+coco	82.9	0.03	31.3

Figure 14 – Testing comparison [36].

2.5.5 RCNN Drawbacks

Even though RCNN was a major breakthrough when it was introduced, today it lags what newer and more advanced models have to offer. Some of the major drawbacks of the RCNN architecture are [34]:

- A considerable amount of time is required for network training. This is due to the large number of region proposal per image, which could be up to 2000.
- Cannot be implemented in real time.
- The SVM is fixed and cannot be changed. This may lead to propagation of bad candidate regional proposals.

2.5.6 YOLOv8

YOLOv8 is the latest in the lineup of object detection models from Ultralytics. Built upon the advancements of the previous versions, YOLOv8 offers increased speed and accuracy as well as new and improved features ideal for any real time object detection task. These improvements are possible thanks to architecture upgrades that include the backbone, neck, and head, as well as a better tradeoff between accuracy and speed, and finally a larger selection of pretrained models compatible with a wider range of necessities [32].

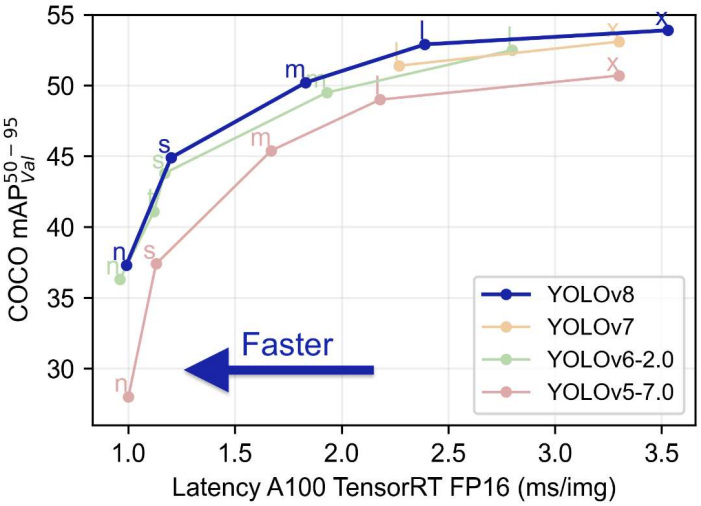


Figure 15 – YOLO comparison [46].

Figure 15 illustrates the inference time results between the previous versions of YOLO and the latest. It is clearly understood that YOLOv8 offers superior inference performance over its predecessors, especially compared to YOLOv5, which is understandable considering this version is the oldest on the list.

YOLOv8 offers the following sub-versions:

- YOLOv8n – Nano
- YOLOv8s – Small
- YOLOv8m – Medium
- YOLOv8l – Large
- YOLOv8x – Extra large

Figure 16 establishes the benchmark data for the various versions of YOLOv8. It is worth noting that the inference time increases from the smaller models to the largest, (nano to extra-large) while precision values are inversely proportional, decreasing from largest to smallest (extra-large to nano).

Detection (COCO)

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 16- YOLOv8 subversions performance information [32].

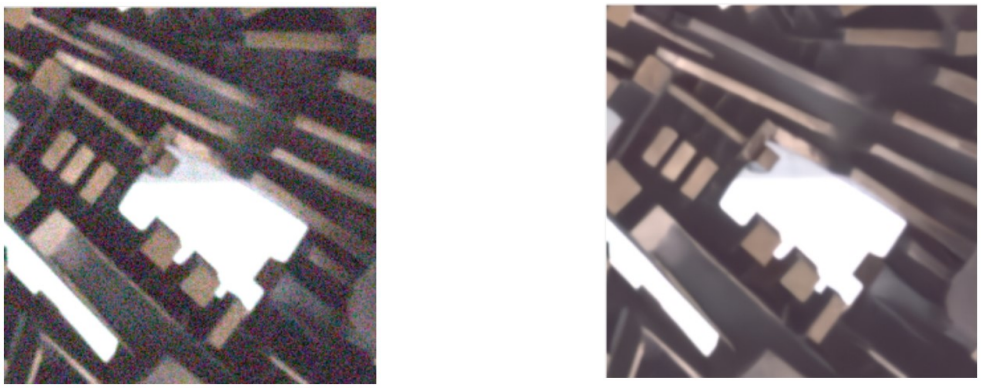
2.5.7 Image preprocessing

Computer vision tasks require images of excellent quality in order to increase the possibility of obtaining the highest precision rates possible during object detection. Through the use of CNN, images may be enhanced in many ways, for example, increasing resolution, de-noising, dual pixel defocus blurring, amongst others [35].

2.5.8 Real image de-noising

Image denoising is defined as the process of applying techniques with the ultimate goal of removing noise from images. This noise may occur due to any number of reasons or conditions which makes it very difficult to identify, and thus, difficult to prevent.

Traditionally image de-noising was achieved by manipulating transform coefficients by averaging neighboring pixels in order to obtain a uniform and overall better image [35].



(a) (b)

Figure 17 – (a) Image with noise (b) Image without noise [35].

Today, deep learning techniques such as CNNs are used to enhance images and remove noise contamination almost completely. Figure 17 demonstrates the visual differences between an image containing noise (a), and another that has been submitted to noise reduction techniques, (b).

2.5.9 Super resolution

Super resolution is defined as the task of constructing a high-resolution image from one of lower resolution.

Prior to CNNs, techniques such as the sampling theory, edge-guided interpolation, natural image priors, and others, were proposed and applied to accomplish this task [35].

Today, deep learning techniques are used as they provide improved results compared to the traditional algorithms. Early deep learning super resolution methods take a low-resolution image as input and learn directly from it generating its higher resolution copy. More recently, super resolution models learn the high frequency image detail and employ this onto the exiting low resolution input image, thus creating the upgraded output image [35].

(a) (b)

Figure 18 – (a) Low resolution (b) High resolution [35].

Figure 18 demonstrates the visual differences between a low-resolution image (a), and another image of a higher resolution, (b).

3 Deep learning image classification applied to microorganisms

This chapter covers the current state of deep learning image classification techniques applied to the field of microbiology and some of its contributions.

3.1 The early days of detection

When microorganism detection was introduced, the overall goal was to determine whether or not objects existed in the sample. At this point, traditional detection methods were based on segmentation, where shapes and forms were targeted and not necessarily biological structures [38].

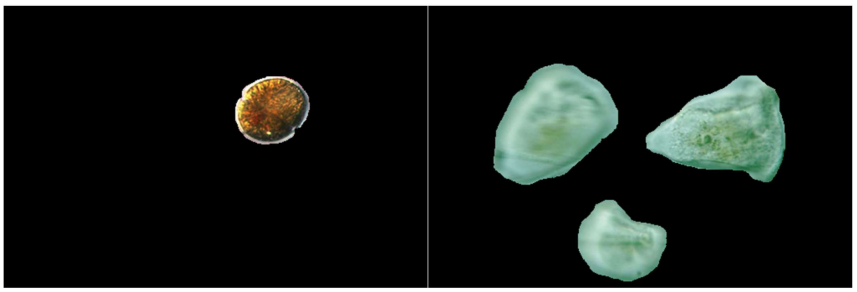


Figure 19 – Example of segmentation-based detection [38].

Figure 19 exemplifies the end product of segmentation-based detection where lines, edges and other shapes are identified to resemble the desired object.

The early 2000's era was mainly influenced by three traditional machine learning detection methods, the Viola-Jones (VJ) detector, the Object Detection with Deformable Part Models (DPM) detector, and finally the Histogram of Oriented Gradients (HOG) detector [38].
With AlexNet in the mid-2000s came the introduction of CNN, which led to an explosion of different types of image detection models. These were divided into two groups, the one-stage detectors, and the two-stage detectors [37]. The following Figure 20 illustrates the milestones associated with object detection using deep learning techniques.

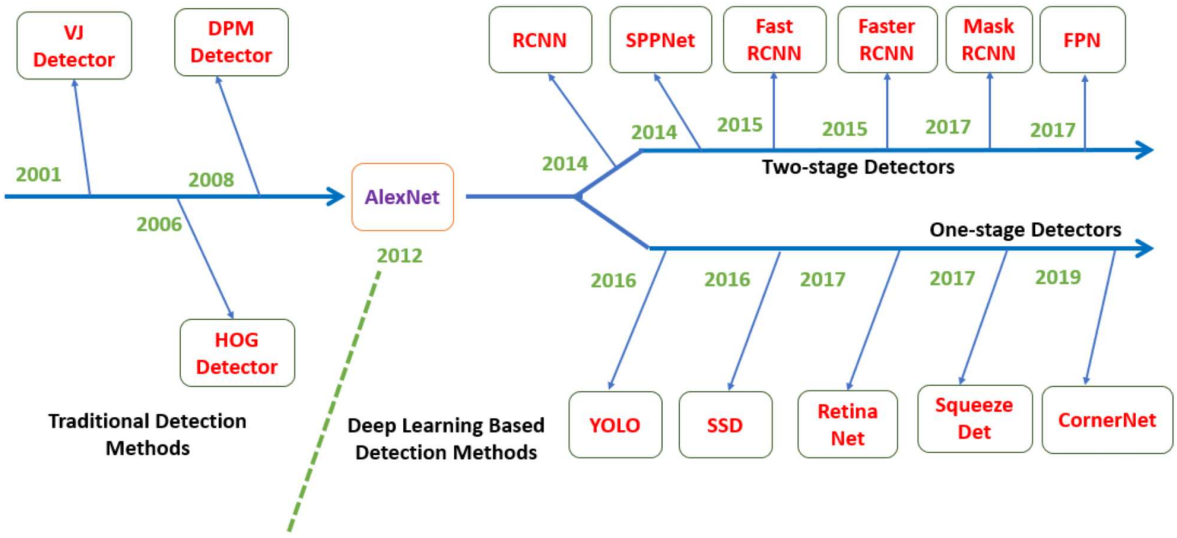


Figure 20 – Milestones of deep learning object detection [37].

These CNN-based models provide faster and better results than the machine learning models prior to AlexNet, making their application to microbiological imaging tasks remarkable.

3.2 The modern age of detection

Current CNN-based image classification methods applied to microorganisms provide high quality information such as class designation and precise locations of objects through the use of bounding boxes surrounding each identified object [38].

The morphology and aspect of microorganisms can vary greatly depending on the species. This can pose a problem when applying image classification methods as it becomes extremely difficult to train a model capable of detecting a large number of characteristics that distinguish the different types and species of microorganisms. The solution is to segregate the classification tasks into specific types of microorganisms, ideally, all with very similar features.

3.3 The current state of the art

Research conducted [39] suggests the great potential of open-source deep learning models applied to the analysis of microbial bioimages. These open source and user-friendly approaches give researchers the freedom to test different models and networks allowing them to choose the one that provides the best results and benefits for the particular dataset.

Networks such as StarDist and SplineDist have provided good results at detecting small rod-shaped and coccoid bacteria, namely E-coli and Staphylococcus aureus, in bright field and fluorescence images, [39] while U-Net and pix2pix excelled when applied to elongated cells

[39]. It is worth noting that StarDist and SplineDist are both deep learning models designed specifically for nucleus and cell segmentation, respectively.

Investigative work has also been conducted on datasets containing a variety of microorganisms against various deep learning CNN networks. An example of this is the EMDS-7 [9] dataset which contains images of microorganisms with very different morphological characteristics and features. This adds a certain level of complexity as there are many variables that need to be trained in order to obtain acceptable results. Additionally, special consideration was taken into account when dividing up the test, train, and validations splits. It was necessary to equally divide the 42 classes amongst the splits, otherwise this would affect the overall results [9]. The tests were carried out against 5 different deep learning models, Faster RCNN, YOLOv3, YOLOv4, SSD, and RetinaNet [9] where Faster RCNN presented the best results.

Another study used a dataset of 437 images of algae to train and test various deep learning models to detect between 30 classes of different species [40]. The images were collected using a microscope, then sorted and a total of 1164 labels were completed. The end goal was to develop a field-testing device capable of detecting the target algae in natural outdoor conditions in real time. For this, YOLOv3 and YOLOv4 along with the associated Tiny models were considered for training. The Tiny models are normally implemented on smaller devices with less computational power; thus this option is ideal for a small transportable device. All four models were used for training and the weight files were used for inference [40]. The results showed that both YOLOv3 Tiny and YOLOv4 Tiny have a higher accuracy score and lower inference time than the other versions. The training times are also shorter for the aforementioned versions [40]. These results are very encouraging.

The research paper "An Improved YOLOV5 Based on Triplet Attention and Prediction Head Optimization for Marine Organism Detection on Underwater Mobile Platforms" by Yan Li, Xinying Bai, and Chunlei Xia, published in the Journal of Marine Science and Engineering [44], updated and implemented a YOLOv5-based model for automatic detection of marine organisms in their natural habitat using a mobile device. Some initial challenges were identified, such as the possibility of lower image quality, blurriness, and complexity of the image background due to the nature of the underwater environment. These challenges would cause performance issues, so to overcome them, the original YOLOv5 model was subjected to some modifications, such as the addition of an attention mechanism and multi-scale detection strategy, fine-tuned to detect four types of marine organisms in their natural environment, as well as a triplet attention mechanism to increase the feature extraction capabilities, and a prediction head which was reconfigured to increase the efficiency of small-scale capture. The

dataset images were captured by an underwater robot off Zhangzidao Island in China. The sea urchin, sea cucumber, starfish, and scallop were the four randomly selected organisms for this project. A total of 3100 images were divided into test, training, and validation splits and used for model training. The model training sessions were performed on a Linux-based server with Nvidia Titan GPU cards. The final trained weight file was implemented on a Jetson Nano Developer Kit, where the system was tested in real time for detection and classification of the four classes. The overall metrics obtained show a significant increase in performance over the course of the test iterations, leading the researchers to believe that this platform has the potential for further development into an autonomous underwater vehicle used to monitor underwater organisms and habitats.

Another research paper entitled "Accelerating the Detection of Bacteria in Food Using Artificial Intelligence and Optical Imaging" aims to develop a method based on a CNN model, trained, and used for real-time detection of E-coli contamination on food. The detection method will rely on the YOLOv4 algorithm and bright field microscopy.

First, the microbial samples were prepared in the laboratory where they were cultured in soy agar overnight. The samples were then transferred to agar plates and digital imaging was performed using phase contrast microscopy. The images were captured using a digital camera coupled to the microscope. Images of the microcolonies were taken every 30 minutes for a period of 5 hours.

Identification and classification was performed using a standard computer equipped with an appropriate GPU for efficient performance. The dataset contained 315 images of each bacterial strain, collected over 3 independent experimental periods. The images were labeled according to the specific species and bounding boxes were applied. Augmentation techniques such as flipping, and rotations were applied to the dataset to improve accuracy. The images were also resized from their original size to 608 x 608 pixels. Finally, the model was trained on the dataset and the metrics were analyzed to determine performance.

The detector was first tested on E. coli samples in the lab to determine if the system was working correctly. Once the microorganisms were correctly located and classified, the next step was to detect E. coli on food samples. Romaine lettuce was chosen to host the E. coli for analysis. The leaves were inoculated with sample E-coli and allowed to dry. Once dry, the leaves and E. coli mixture were homogenized in a blender and a sample was transferred to a sample dish for microscopic analysis. A total of 100 images were acquired and analyzed using the trained YOLOv4 model. The results were very encouraging as 11 out of 12 samples were

correctly identified as being E-coli positive, a strong indication that the trained model is performing extremely well [15].

The research paper "Rapid on-site monitoring of Legionella pneumophila in cooling tower water using a portable microfluidic system" aims to develop a portable system capable of real-time detection of Legionella in cooling tower water. This bacterium poses a real threat to humans as it is a waterborne pathogen that can easily affect a large number of people very quickly. The portable setup consisted of three main components, the syringe application pump, the detection system, which consisted of an optical system with a diode laser, filter block and charge-coupled device camera, and finally a laptop computer for analysis. The entire setup is packed in a carrying case for easy transport.

The analysis is based on a microfluidic chip for "on-chip" staining and counting of the microorganisms of interest. The sample is injected together with the dye using two separate syringe pumps. As the sample and dye make their way through the chip's channels, they mix and react. The fluorescently labeled bacterial cells are excited by the bright blue beam from the laser diode. The stained cells were videotaped by the charge-coupled device camera for 10 to 15 minutes for each sample. An epifluorescence microscope was used to detect the signals from the fluorescent antibody. The cells present in the movie were processed and counted using analysis software capable of enhancing positive signals and discriminating background fluorescence. The cell count and flow volume were derived from each sample and calculated as cells/mL. Although this project does not involve the use of machine learning techniques, it holds value as a portable, real-time detection and classification technique [52].

4 Dataset, image enhancement, and model training

This chapter describes in detail the steps taken to prepare the raw images into a working dataset, applied image enhancement techniques and finally, model training.

The initial phase of the project can be broken down into three main parts, data acquisition and testing, image enhancement as a possible method to increase test results, and finally cropping and development of a larger dataset based on the original.

Figure 21 shows the implementation and testing procedure used for this project. First, a data set consisting of 168 images of E. coli was created. These images were captured in the laboratory using a microscope equipped with a digital camera. The images were then annotated to identify the objects of interest by labeling them with the appropriate class label. Augmentation techniques, such as rotations and flips, were applied to the dataset images to create a more diverse image set. The images were then trained using this dataset and the resulting metrics were analyzed. The first phase of training yielded very low results, so image enhancement techniques were applied in an effort to improve image quality and increase training performance. Again, the results obtained showed little or no improvement, so we returned to the first step. The images were then reduced to 640x640 pixels to fit the resolution requirements of YOLO. This increased the number of images and therefor the size of the dataset. The previous steps were repeated, annotation was applied to the new dataset, and augmentation was performed. The model was trained on the new images and the results were much better. A significant increase in performance was achieved. Finally, the trained weights file was extracted and kept for future use.

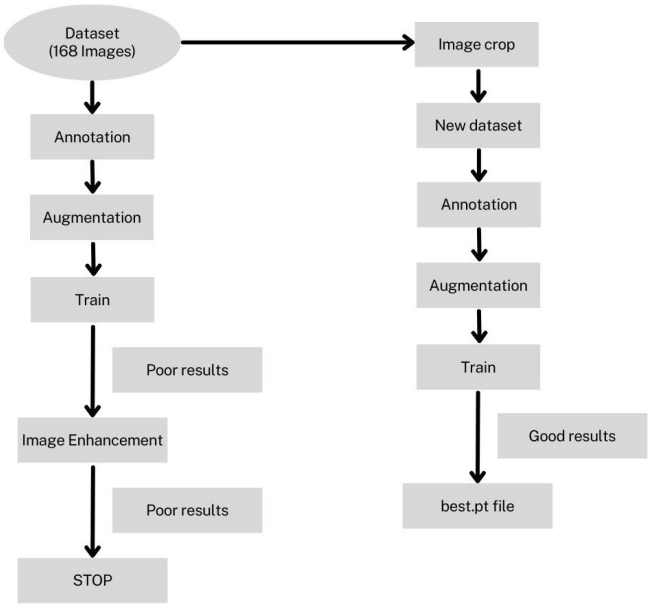


Figure 21 – Implementation and testing procedure.

4.1 The dataset

The original dataset was created through still photography of E-coli samples contained in a laboratory. A total of 168 images were captured for this project using a traditional microscope equipped with a digital camera connected to a desktop computer.

4.2 Annotation

The image annotation involves several steps, starting from selecting the right data sources to establishing clear annotation guidelines and performing the actual annotation. In order to define the classes required to train the model, it was decided that the three unique morphological features of E-coli would serve as the class designations.

These are as follows:

- Class 1 – Bacilli
  - Elongated spherical form.
- Class 2 – Cocci
  - Spherical form.
- Class 3 – Cluster
  - Communities of E-coli microorganisms.

The annotation process normally requires manual labelling of classes for each picture of the dataset. To do so, an online software called Roboflow was used. This software provides a toolset for quick and easy data annotation and training. All 168 images were uploaded to the Roboflow [41] platform, and each image was annotated using the bounding box annotation technique. Figure 22 pertains to the Roboflow annotation workspace.

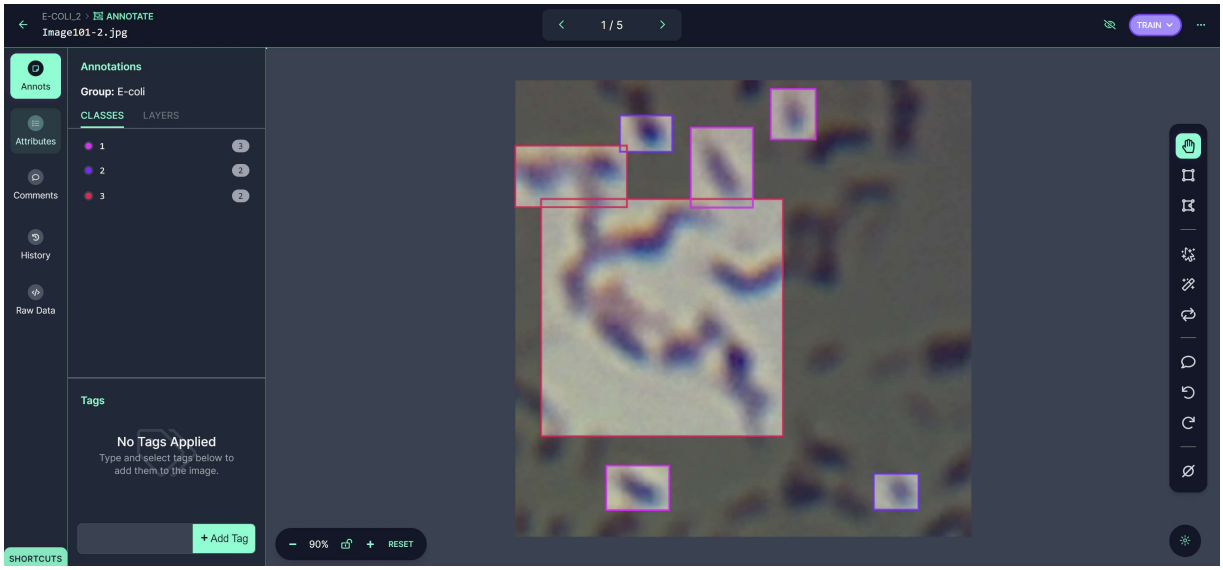


Figure 22 – Roboflow annotation workspace [41].

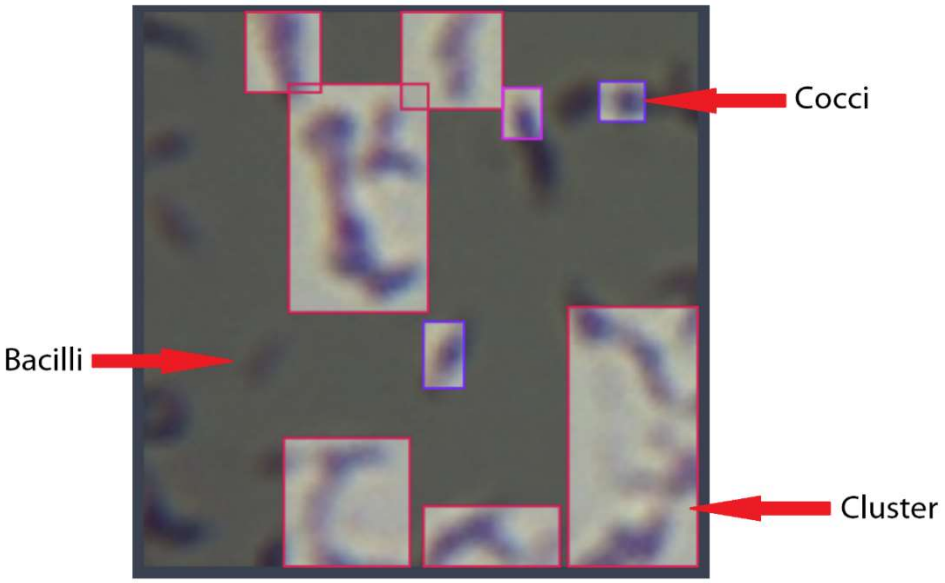


Figure 23 – Labelling example. Class 1: Bacilli; Class 2: Cocci; Class 3: Cluster.

Bounding box annotation provides the coordinates, dimensions and labelling of the objects, enabling machines to understand and interpret visual data. The previous Figure 23 illustrates the labelling process for one image. The object of choice is identified with an appropriately sized bounding box and given a class designation. Once all the objects of interest are labelled, the image is saved, and the process repeats itself until all the images are annotated.

Following, a dataset split was performed dividing the images into three groups, the training set, validation set and test set. The groups were divided in a 70%, 20%, 10% split, respectively. This percentile division was chosen as it is the default offered by Roboflow, however the exact

split can vary depending on many factors including project specifics or the amount of data available. On larger datasets it is common to have a smaller percentage for testing and validation. Conversely, for a smaller dataset it is preferable to use a larger percentage for testing and validation, by doing so, this will ensure a reliable measure of the model's performance.

4.2.1 Augmentation

Next, augmented data is added to create a more robust model by providing more varied data for the model to learn from during training.

In this case, the following augmentation steps were chosen:

- Flip
- 90º Rotate
- Rotation: -15º and +15º
- Bounding box: Flip horizontal
- Bounding box: 90º Rotate
- Bounding box: Rotation -15º and +15º

By doing so, the dataset size was increased from 168 images to 210. The following Figure 24 illustrates some of the augmented images contained in the dataset.

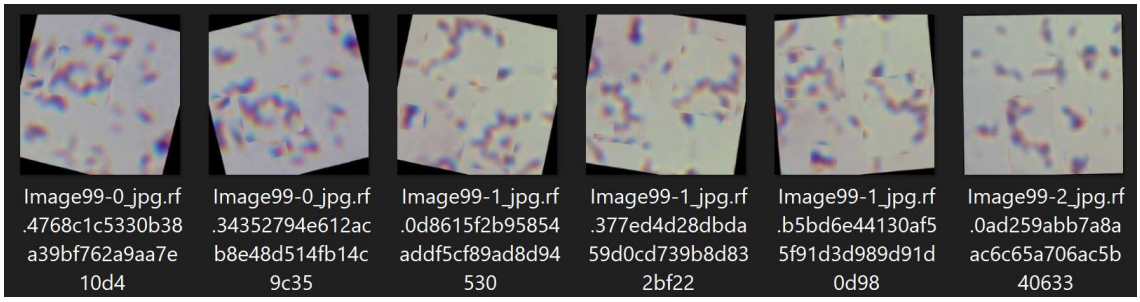


Figure 24 – Augmentation example.

It is important to note that the test and validation sets are used to evaluate the model's performance on unseen data and only the training set receives the augmented images. This is because we want to test and validate the model on real-world, unaltered data to achieve a true measure of the model's performance.

Finally, the new dataset is exported in "YOLO v5 PyTorch" format and downloaded. The following Figure 25 illustrates the obtained file structure.

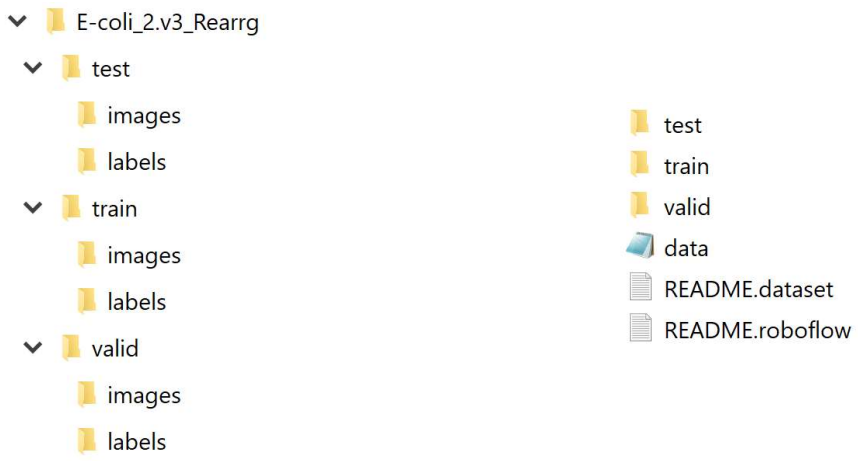
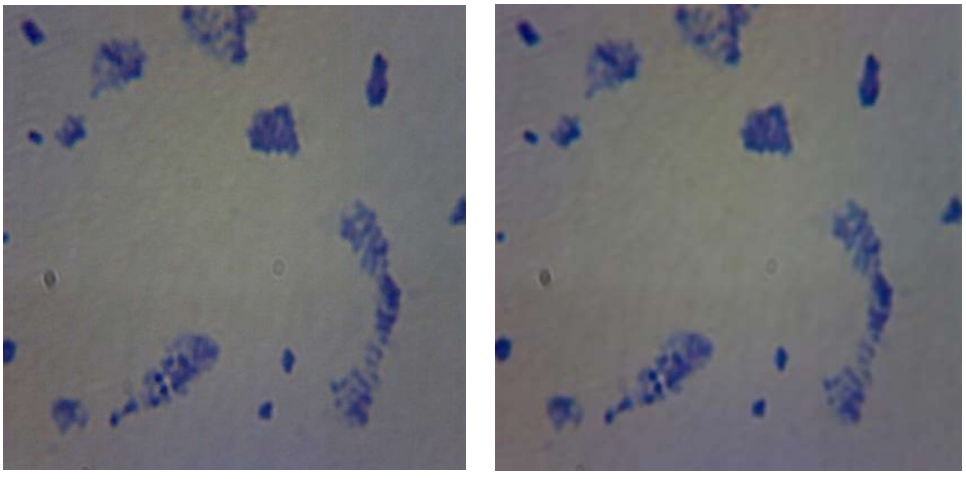


Figure 25 – Yolov5s file structure.

The dataset is now organized into three folders were each one contains a folder with images and another with the labels created during the annotation process. The data configuration file is also included as this will also be required for the training process.

4.3 Image enhancement

During the early stages of training the model, results were very poor. Initially it was thought that the quality of the captured images was to blame. Unfortunately, using a different microscope with higher image quality capabilities was not possible. The decision was made to tentatively increase the quality using image enhancement software. For this, a software called MIRNetv2_demo.ipynb [42], available on GitHub, was used. The images were submitted through a de-noising process, the results are shown on Figure 26.



Original De-noising

Figure 26 – Before and after de-noising image enhancement. No noticeable improvements.

The model was newly trained using the enhanced dataset, however the results continued to be very poor. The image enhancement technique was abandoned as it did not resolve our issue.

4.4 Image cropping

While trying to understand the reason behind the low-test results, it was thought that our dataset may not contain enough images to efficiently train the model. Training a model with a reduced number of images can ultimately produce negative effects as the model will not be able to correctly classify images that are very different from those it was trained on. Obtaining more images through the original method of digital microscope photography was also not an option as this would require a third party and a considerable amount of time. It was also noticed that the dataset images were sized at 2560 x 2048 pixels which is much larger than the recommended size for the YOLOv5s algorithm, which should be 640 x 640 pixels. It was then decided that the images could be cropped into sizes that would suit the algorithm's requirements and by doing so, the dataset would grow in size, effectively solving two problems. ImageMagick [43] was the software chosen for the cropping process. This software speeds up image processing using batch commands capable of processing a great number of images at once. Running on a Linux terminal, the dataset images were cropped down to the required size. Firstly, a bottom strip was removed from each image, creating images with dimensions of 2560 x 1920 pixels. The images were then cropped into squares of 640 x 640 pixels originating 12 squares per image, as shown on the following Figure 27 and Figure 28.

Figure 27 – Removal of the bottom 128 pixels.

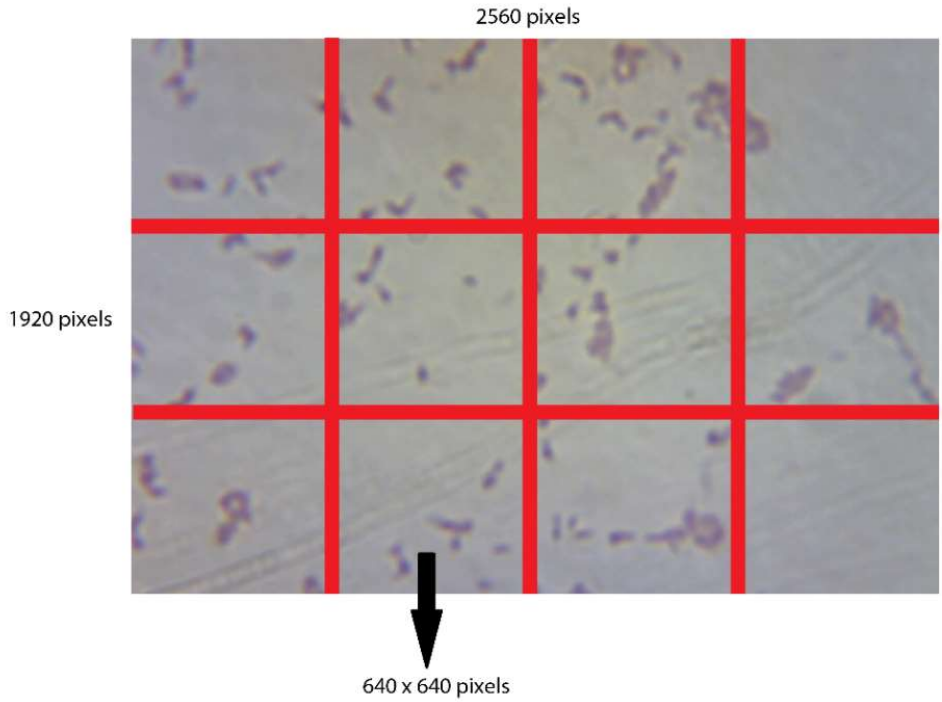


Figure 28 – Cropping process. Division of each image into 12x 640x640 pixel images.

The training that followed yielded much better results, this approach was preferred moving forward.

4.5 Model training

Model training is a process that, depending on the size of the dataset, can be very demanding and require a considerable amount of time and processing power.

Ultralynics maintains a up to date GitHub internet page with all their open-source algorithms, including the YOLO series. For users who may not have sufficient computing power, or require additional, the YOLOv5 repository page offers the use of a Google Colaboratory notebook. Here, the basic code required to run YOLOv5 is available and free for anyone to modify to their project's specifications. Users are encouraged to write and execute Python code through the browser while being provided access to high end computing resources including GPU's. This project benefited greatly from this resource as training times were significantly shortened allowing for more testing iterations.

4.5.1 Google Colab Notebook

The first step was to create a Google Drive folder to where we could save our work to. This was a requirement set by Colab. Next, the YOLOv5 repository was cloned direct from GitHub

and all dependencies were imported and installed. The dependencies normally vary between different versions of YOLOv5.

Our dataset was exported from Roboflow in a compressed file format, it was necessary to modify our code to include the installation of a decompression software and a subsequent decompression step for the target file, demonstrated by Figure 29.

```
# Install RAR
!pip install unrar

Collecting unrar
  Downloading unrar-0.4-py3-none-any.whl (25 kB)
Installing collected packages: unrar
Successfully installed unrar-0.4

# UnRAR the file
!unrar x "/content/gdrive/MyDrive/E-coli_2.v3_Rearrg.rar"
```

Figure 29 – Python code used to decompress dataset file.

Next, the configuration file, *data.yaml* is included. This file contains information regarding the path to the training, validation and testing images and label sets as well as the number of classes and designations of each, as shown by Figure 30.

```
train: /content/yolov5/E-coli_2.v3_Rearrg/train/images
val: /content/yolov5/E-coli_2.v3_Rearrg/valid/images
test: /content/yolov5/E-coli_2.v3_Rearrg/test/images

nc: 3
names: ['1 - Bastonete', '2 - Coco', '3 - Aglomerado']
```

Figure 30 – Configuration file data.yaml contents.

The model training process was initiated by modifying some of the configuration parameters, demonstrated by Figure 31.

```
%%time
%cd /content/yolov5/
!python train.py --img 640 --batch 16 --epochs 1000 --data /content/yolov5/E-coli_2.v3_Rearrg/data.yaml

--cfg ./models/custom_yolov5s.yaml --weights '' --name yolov5s_results
```

Figure 31 – Configurable parameters of the YOLO training process.

- *img* – Pertains to the size of the images used, in this case, 640 x 640.
- *batch* - Number of times a sample is processed before the model is updated.
- *epochs* – Number of the times the dataset is processed through the model.
- *Data* – Indicates the path to the data configuration file.
- *cfg* – Indicates the path to the YOLOv5 configuration file.
- *weights* – No weights used; this option was left blank.
- *name* – Name given to the files containing the results.

Even though the number of epochs is set high at 1000, this amount was never reached as the algorithm is configured to detect when training results show no improvement for at least 100 epochs. This step is beneficial as training beyond this number of epochs will not yield better model results.

After training completion, the trained weights are saved to a file named *best.pt*, this will be the file used for the remainder of the project.

4.6 Training results

4.6.1 Metric descriptions

To measure the efficiency of a trained model, the following standard metrics are typically used.

- **True Positives (TP)** - Correctly predicted positive classes.
- **True Negative (TN)** - Correctly predicted negative classes.
- **False Positive (FP)** - Incorrectly predicted positive class. The true class was negative.
- **False Negatives (FN)** - Incorrectly predicted negative class. The true class was positive.

- **Precision**

This metric refers to the calculated ratio obtained by dividing the number of true positives by the sum of true positives and false positives according to Equation 4.1 [14]. This metric indicates how accurate a model is at correctly reporting positive samples.

$$Precision = \frac{TP}{TP + FP} \qquad (4.1)$$

- **Recall**

This metric refers to the calculated ratio obtained by dividing the number of true positives by the sum of true positives and false negatives according to Equation 4.2 [14]. Higher recall values indicate a higher number of positive samples detected.

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

- **Intersection over Union (IoU)**

The IoU is calculated by dividing the area of the intersection between the ground truth bounding box and the predicted bounding box by the total area of their union, according to Equation 4.3 [14]. A higher IoU value indicates a better prediction.

- Ground Truth Bounding Box - $BBox_{GT}$
- Predicted Bounding Box - $BBox_{Xp}$

$$IoU = \frac{area(BBox_{GT} \cap BBox_{Xp})}{area(BBox_{GT} \cup BBox_{Xp})} \tag{4.3}$$

Figure 32 illustrates the mathematical operation where the area of intersection between the ground truth bounding box (A) and the predicted bounding box (B) is divided by the total area of union (A and B).

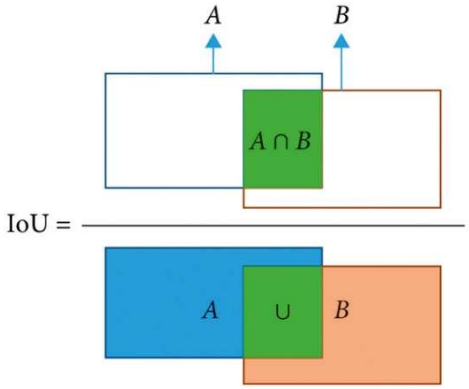


Figure 32 – IoU. A - Ground Truth Bounding Box; B - Predicted Bounding Box [50].

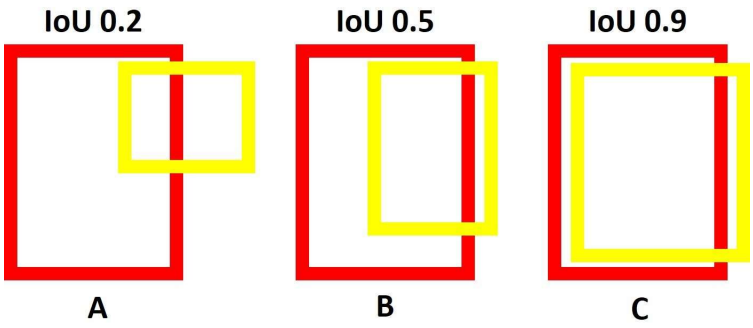


Figure 33 – Example of IoU scores [49].

Figure 33 shows how the IoU score increases as the ground truth bounding box and predicted bounding box coincide with each other.

- **Mean Average Precision (mAP)**

This metric is used to measure model accuracy for intersections over union (IoU) between 0.5 and 0.95 at intervals of 0.05. The higher the mAP@.5:.95 score the better the model is at detecting the objects overlap with the ground truth bounding box.

The following metrics apply to both the training and validation datasets.

- **train/box_loss**
- **val/box_loss**

Box Loss simultaneously evaluates how well an algorithm identifies the center of an object and how well the proposed bounding box fits around the same object.

- **train/obj_loss**
- **val/obj_loss**

Objectness probabilistically measures the presence of an object in a given region. A high objectness score indicates a high probability that the object is contained within the window.

- **train/cls_loss**
- **val/cls_loss**

Classification loss is a measure of how well a model assigns the correct class label to an object.

4.6.2 Overall training results

Figure 34 shows the overall results obtained over many training iterations. After each training session, the results were analyzed, and adjustments were made to the dataset until the best scores were achieved.

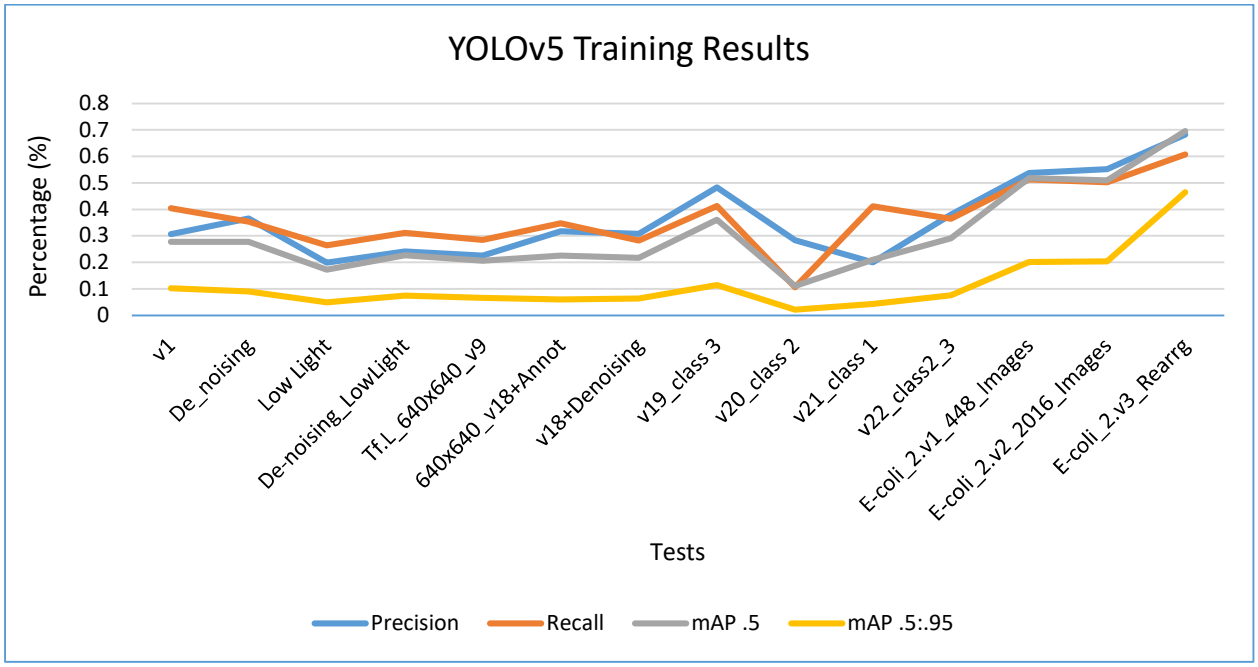


Figure 34 – YOLOv5 overall training results.

Analyzing the above Figure 34, the first dataset, "v1", used to test the model rendered very low metric values. During this test, training was performed without any modification to the dataset or the YOLO parameters, as specified in the Ultralynics manual. This allowed a baseline to be established and the future evolution to be determined.

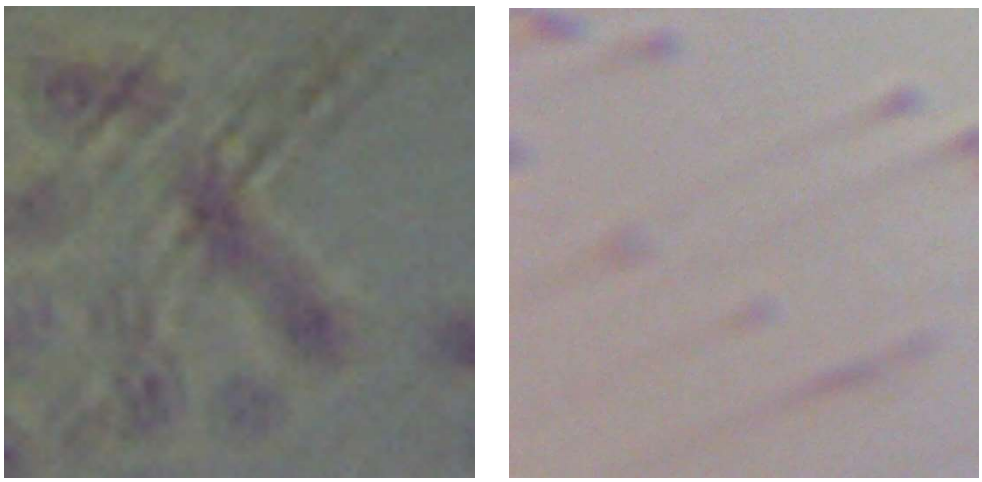
In the next series of test iterations, image enhancement techniques such as denoising and low light enhancement were applied to the dataset. Concerns grew that the quality of the images might be to blame for the poorer results. After applying the odds and running several test sessions, the results were worse in some cases.

As a test, the model was trained with only the third class, clusters. This class is the easiest to identify due to its random and almost unique shape. The results were much better now, leading us to believe that the problem was not the quality of the images, but rather a parameter.

After realizing that the images in the dataset were incorrectly sized, the dataset was reworked from the ground up. The images were cropped to the 640x640 pixel size required by YOLOv5s.

A significant amount of time was required to manually annotate a new and much larger dataset. The results obtained were now much higher than ever before.

In an effort to further improve the metrics, some outlier images were identified and removed. These images consisted of images that were captured while moving or were too blurry. Figure 35 is an example of two images that were removed from the working dataset.



(a) (b)

Figure 35 – Example of removed images. (a) Blurry. (b) Captured in movement.

After removing these images and retraining the model, the metric scores increased significantly. The mAP.5:.95 score increased to more than twice its previous value, a good indication that the model is performing better.

4.6.3 Training results pertaining to E-coli_2.v3.Rearrg.

The following Table 1 and Figure 36 show the results obtained for the E-coli_2.v3.Rearrg model.

Table 1 – Training results pertaining to E-coli_2.v3.Rearrg.

	Precision	**Recall**	**mAP.5**	**mAP.5:.95**
All	0.683	0.608	0.696	0.465
Bacilli	0.597	0.532	0.611	0.393
Cocci	0.699	0.617	0.699	0.436
Cluster	0.754	0.674	0.779	0.567

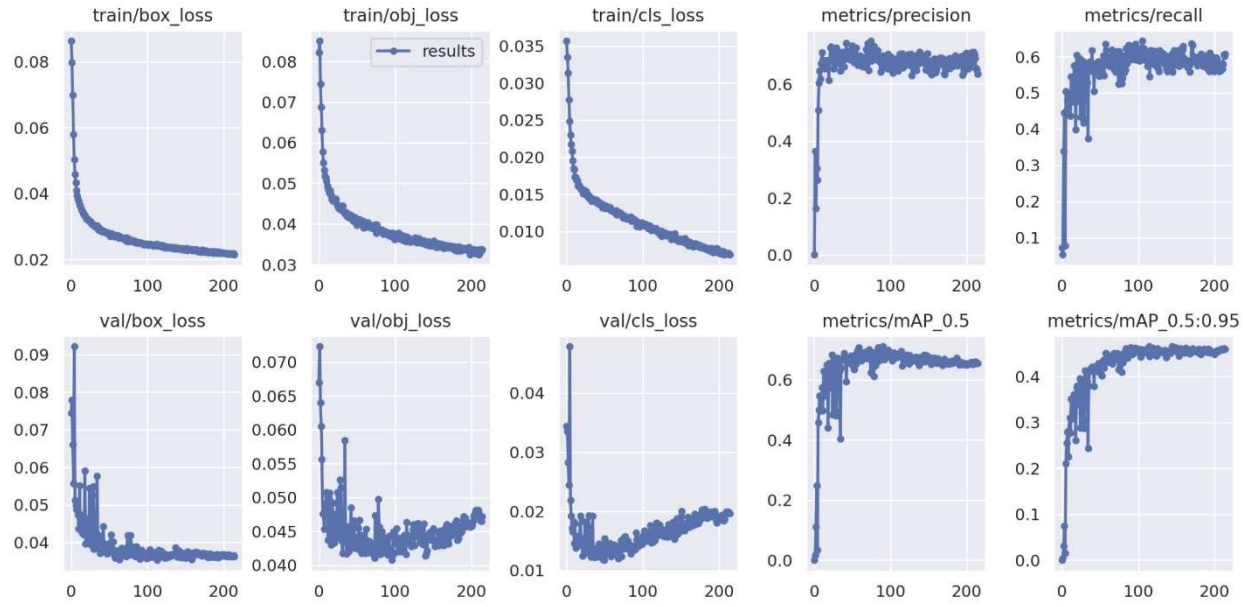


Figure 36 – Plotted training results pertaining to E-coli_2.v3.Rearrg.

According to the training and validation loss metrics in Figure 36, the model evolved very well, showing a rapid decrease toward zero as training progressed. This indicates that the model converged very quickly and produced good results in a short time. Early Stop was used to automatically stop training when no further training progress was extracted. This can be seen in the box loss plot as the trend becomes nearly horizontal at about 100 epochs and training is automatically stopped at about 200 epochs. The validation loss metrics show some isolated spikes that may be due to irregular or outlier images that don't fit the overall nature of the dataset. A possible mitigation could be to increase the size of the validation dataset, thus including more variation and homogenizing the dataset. The precision, recall and mAP values also average 60%, which is a significant increase in performance compared to the baseline test results. Again, there are isolated outlier spikes for the precision, recall, and mAP metrics, which may be due to the same reason as previously explained.

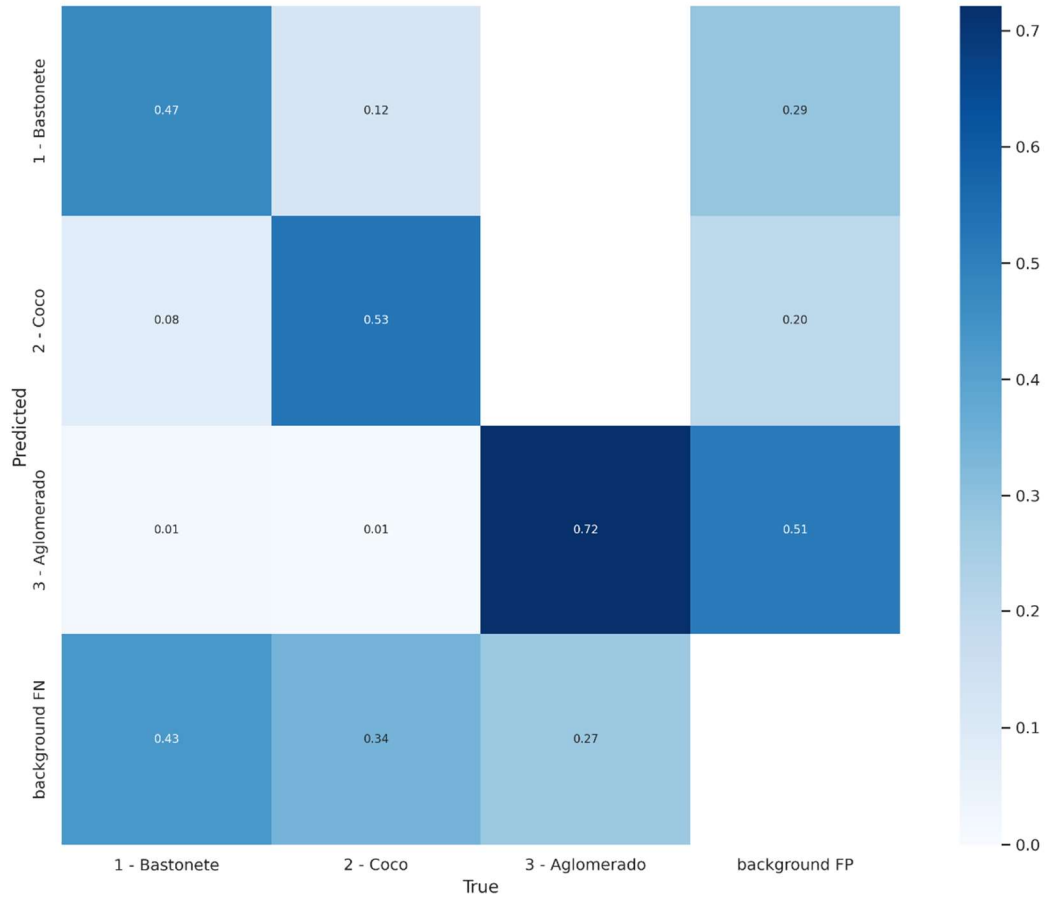


Figure 37 – Confusion matrix results pertaining to E-coli_2.v3.Rearrg.

Figure 37 shows the confusion matrix used to evaluate the model's classification performance. The rows in a confusion matrix correspond to the model's predictions, and the columns correspond to the known truth. The samples that are correctly classified by the model are represented by the cells arranged diagonally from left to right in the matrix. The off-diagonal cells show the number of samples incorrectly classified by the model for each class. For object detection and classification tasks, a column and row of background classes are added. This represents the regions of an image that do not contain objects. The values contained in each cell are normalized and range from 0 to 1.

From the confusion matrix, we can see that the model performs very well in detecting and classifying objects of the third class - clusters, ("Aglomerados"), with a score of 0.72. The extremely low scores contained in the adjacent cells to the left indicate that the model very rarely, practically never, confuses the other two classes with the third class. This is a good indication that the model is very good at detecting clusters.

For the second class - Cocci, ("Cocos"), the model performs well with a score of 0.53, indicating that the classifications are correct more than half the time. The score of the adjacent cell on the left is very low, and similar to the third class, this indicates that the model does not confuse the second class with the first very often.

Finally, the model shows some difficulty in classifying the first class - Bacillus, ("Bastonete") as the cell score is the lowest at 0.47. In addition, the adjacent cell has the highest score of the group, further indicating that the model has difficulty distinguishing and classifying this class correctly. A possible explanation for the lower performance of this class may be that in the laboratory sample solution from which the images were photographed, many microorganisms are at different spatial depths and orientations, making the shapes appear to be what they are not. For example, if a Bacillus microorganism is tilted downward and being a round cylinder in shape, it may appear to be a Coccus and the model may recognize it as such. A possible mitigation for this would be to include more and better images of Bacillus in the dataset to reinforce the model's training.

Figure 38 – Validation dataset results pertaining to E-coli_2.v3.Rearrg.

Figure 38 shows the object classification results for the validation images. The majority of the scores are quite high, averaging over 0.6. This is another indication that the overall performance of the trained model has improved.

5 Implementation with Nvidia Jetson Nano

This chapter objectively identifies the steps taken in order to successfully set up the hardware and software requirements for the real time object classification system.

The initial implementation was based on the Nvidia Jetson Nano, the camera sensor, a traditional optical microscope with a 5x magnification lens, and a computer monitor for visualization, as shown in Figure 39 and Figure 40.

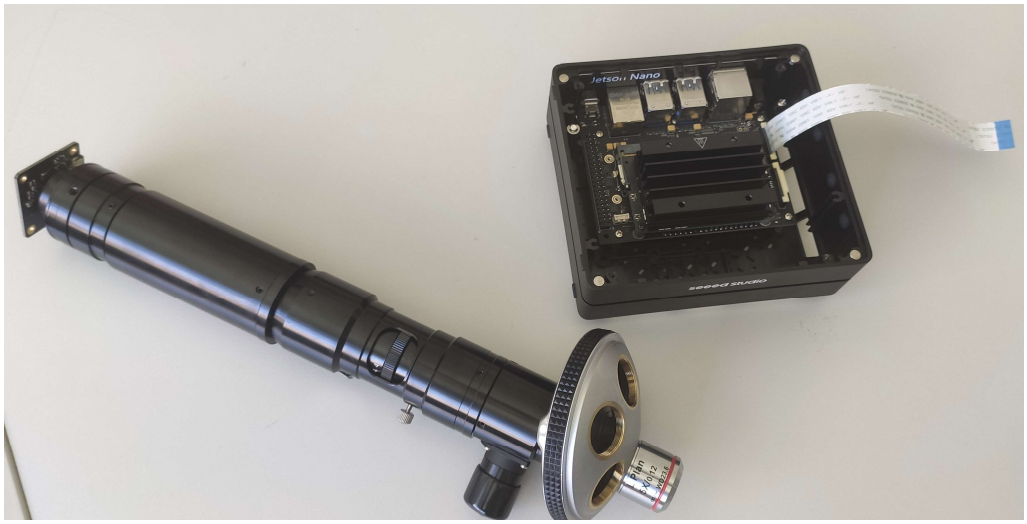


Figure 39 – Optical microscope with 5x lens (Left), and Nvidia Jetson Nano (Right).

Figure 40 – Nvidia Jetson Nano connected to camera which is mounted to the microscope.

5.1 Nvidia Jetson Nano

The Nvidia Jetson Nano developer kit is part of a family of small computers developed for deep learning and neural network applications. Aimed for developer communities and learners alike, its powerful graphics processing unit allows it to perform demanding tasks such as image classification and object detection at ease.

Our systems characteristics are as follows [45]:

- GPU - 128-core Maxwell
- CPU - Quad-core ARM A57 @ 1.43 GHz
- Memory - 4 GB 64-bit LPDDR4 25.6 GB/s
- Storage – 16 GB eMMC
- CSI camera connect – 2x MIPI CSI Camera

The following Figure 41 illustrates the Nvidia Jetson Nano hardware specifications as well as the installed software.

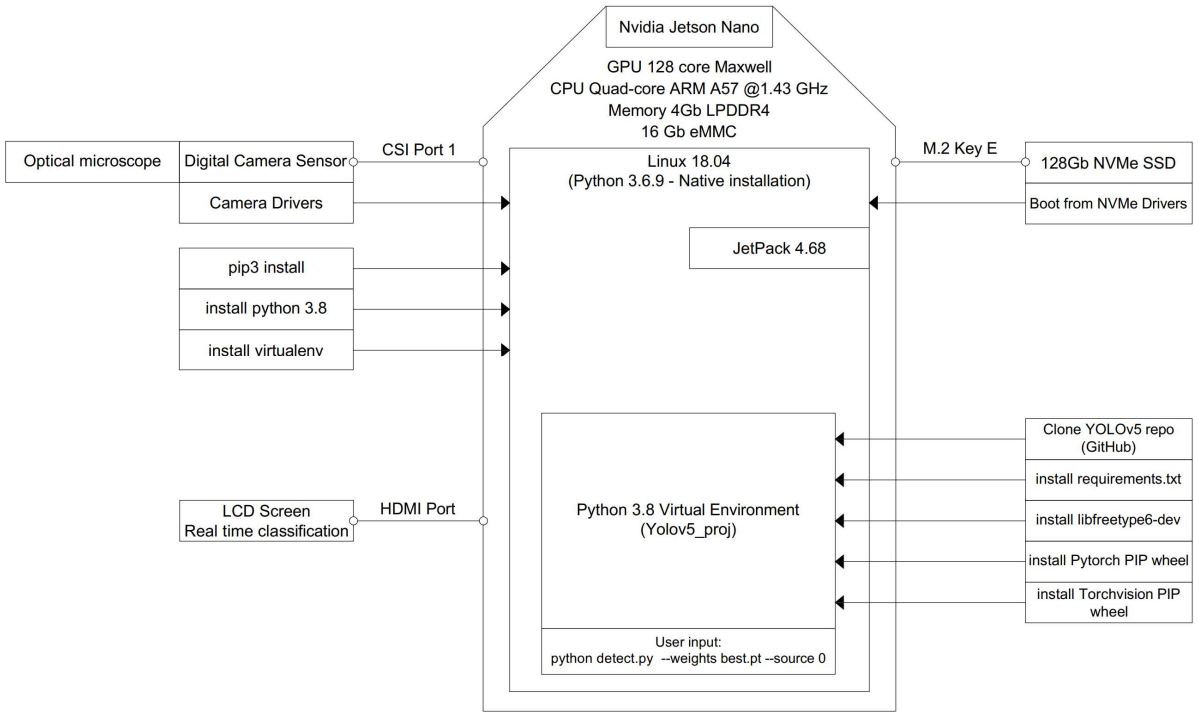


Figure 41 – Nvidia Jetson Nano hardware architecture and installed software.

Out of the box, Linux version 18.04 was the installed operating system. As this was a brand-new unit, it was necessary to finalize the operating system's initialization process inserting the desired system language, keyboard type, username, and geographical location.

Jetpack 4.68 was installed; this framework provides a suite of applications focused on deep learning applications. This version of Jetpack is specific to this particular model of Jetson Nano and upgrading to a newer version would cause compatibility issues.

The first step taken was to update the operating system by executing the appropriate command in a terminal window. After all of the updates are downloaded and the installation process has finalized, a system restart is required.

It was noticed that after concluding the initialization of the Jetson Nano, only 1,5 Gb of free hard drive space was available. This would not be enough disk space to continue with our project so an additional solid-state drive (SSD) of 128Gb was procured and installed. Running the operating system directly from the newly installed SSD proved to be difficult as the automatic OS installer would only detect the 16Gb eMMC device and not the SSD. This issue seemed to be common amongst many people. The solution was a script that permitted the OS boot to happen on the eMMC as normal and then transfer the file system onto the SSD where the remainder of the session would happen. In other words, the eMMC was only required for the booting process and nothing else.

5.2 Sony IMX477 camera sensor

The camera sensor chosen for this project was the Sony IMX477. This sensor has a resolution of 4056 (H) x 3040 (V) at 12.3MP and is capable of preforming at a maximum framerate of 240 frames per second. Figure 42 illustrates the camera sensor.

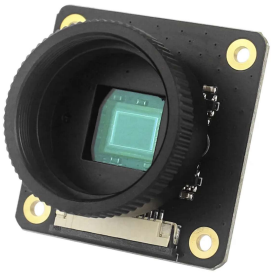


Figure 42 – Sony IMX477 camera sensor.

In order to install the camera sensor to the Jetson Nano, the cameras ribbon cable was connected to the Jetson's native MIPI CSI camera port. The camera must be connected prior to booting

the OS, otherwise it will not be detected. Next, the necessary drivers were downloaded from ArduCam's GitHub repository and installed.

Executing the command from the below Figure 43, a window will open where it is possible to view the video stream captured by the camera and thus, confirming the camera is functioning appropriately.

```
gst-launch-1.0 nvarguscamerasrc sensor_id=0 ! \
'video/x-raw(memory:NVMM),width=1920, height=1080, framerate=30/1' ! \
nvvidconv flip-method=0 ! 'video/x-raw,width=960, height=540' ! \
nvvidconv ! nvegltransform ! nveglglessink -e
```

Figure 43 – Command to view video stream from camera.

5.3 YOLOv5 and required dependencies

The following outlines the key steps taken during the installation of YOLOv5 and all of the required dependencies and libraries.

- PIP – Preferred Installer Program

The preferred installer program (PIP) is a command line utility that installs, reinstalls, or uninstalls packages in one simple command. This package handler was not installed natively on our version of Linux, to do so, the following command illustrated by Figure 44 was executing in a terminal window.

```
sudo apt install -y python3-pip
pip3 install --upgrade pip
```

Figure 44 – PIP download and installation.

This was a fundamental step, without this program no other installations would be possible.

- Python & Virtual environment

Python is a high-level, general-purpose programming language which came pre-installed on our system as part of Jetpack. The version of Python required by YOLOv5 is Python 3.8, unfortunately the pre-installed version was 3.6.9. Uninstalling this older version and substituting it with the more recent one created compatibility issues and rendered the system unusable. A system restoration was required in order to correct the problem.

The solution to this problem was to download and install Python 3.8 on the OS and only execute it within a virtual environment. The following two commands were used to carry out the process, demonstrated by Figure 45 and Figure 46.

```
sudo apt install python3.8
```

Figure 45 – Installation command for Python 3.8

```
sudo pip3 install virtualenv
```

Figure 46 - Installation command for the virtual environment

At this point both Python version 3.6.9 and 3.8 are installed as well as the virtual environment application.

The desired virtual environment must be run from within a dedicated project folder. To do so, a simple terminal command is executed creating this folder. Changing directories into the newly created folder, we now execute the below command in Figure 47 which creates and activates the virtual environment.

```
python3.8 -m venv Yolov5_proj
source Yolov5_proj/bin/activate
```

Figure 47 – Creation and activation of virtual environment.

- Installing YOLOv5

YOLOv5 is installed directly from Ultralynics GitHub repository using the below command in Figure 48.

```
git clone https://github.com/ultralytics/yolov5
```

Figure 48 – YOLOv5 install command.

After successfully completing the installation, the requirements text file located inside the YOLOv5 folder must be edited to omit torch and torchvision from the list by adding a "#" before them. Both of the omitted will be installed at a later time as we must install versions of these software's which are compatible with the specific ARM architecture of the host system.

The code executed to achieve the aforementioned editing of the requirements text file is illustrated by Figure 49.

```
vi requirements.txt
i (edit mode)
# torch
# torchvision
esq (button)
:wq (to exit)
```

Figure 49 – Sequence of commands to comment torch and torchvision.

To finalize, the requirements text file and an additional dependency file are executed, as per the Figure 50.

```
pip install -r requirements.txt
sudo apt install -y libfreetype6-dev
```

Figure 50 – Installation of contents of requirement.txt and additional dependencies.

The Pytorch PIP wheel was installed by following the indications on the Nvidia Jetson's developer internet page [51]. As previously mentioned, the specific version of Pytorch, in this case v1.10.0, was chosen according to the systems processor type and the installed version of Jetpack. The command lines were copied directly from the developer's page and executed, as illustrated below in Figure 51.

```
wget https://nvidia.box.com/shared/static/fjtbno0vpo676a25cgvuqc1wty0fkkg6.whl
                                        -O torch-1.10.0-cp36-cp36m-linux_aarch64.whl

sudo apt-get install python3-pip libopenblas-base libopenmpi-dev libomp-dev
sudo apt-get install -y libopenblas-base libopenmpi-dev
pip3 install Cython
pip3 install numpy torch-1.8.0-cp36-cp36m-linux_aarch64.whl
```

Figure 51 – Installation of the Pytorch PIP wheel.

Similarly, the Torchvision PIP wheel was also installed by copying the provided code from the Nvidia developer page referenced to version 1.12.0, as per Figure 52.

```
sudo apt-get install libjpeg-dev zlib1g-dev libpython3-dev libopenblas-dev libavcodec-dev
                                                                    libavformat-dev libswscale-dev

git clone --branch <version> https://github.com/pytorch/vision torchvision
cd torchvision
export BUILD_VERSION=0.x.0
python3 setup.py install --user
cd ../
pip install 'pillow<7'
```

Figure 52 – Installation of the Torchvision PIP wheel.

- Running YOLOv5

A successful installation process renders the final step, executing YOLOv5 for real time image classification. Here, we will finally be able to visualize a sample a test the classification capabilities of our setup.

The first step, and probably the most important, is to paste the best.pt file created earlier during the model training into the YOLOv5 weights folder. By doing so, we are ensuring that the classification weights are ours and not the default.

Executing the below command in the terminal window will launch the detect function of YOLOv5 with the best.py file as the weights and the locally connected camera as the input source, as described in Figure 53.

```
python detect.py --weights best.pt --source 0
```

Figure 53 – detect.py with best weights file and camera as source.

The below Figure 54 and Figure 55 demonstrate the real time image classification results achieved with our setup.

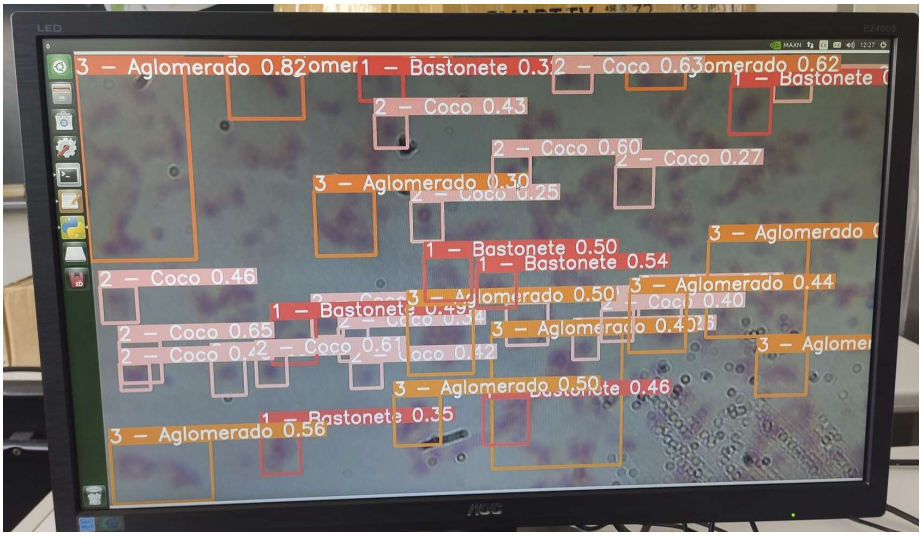


Figure 54 – Real time image classification results - 1

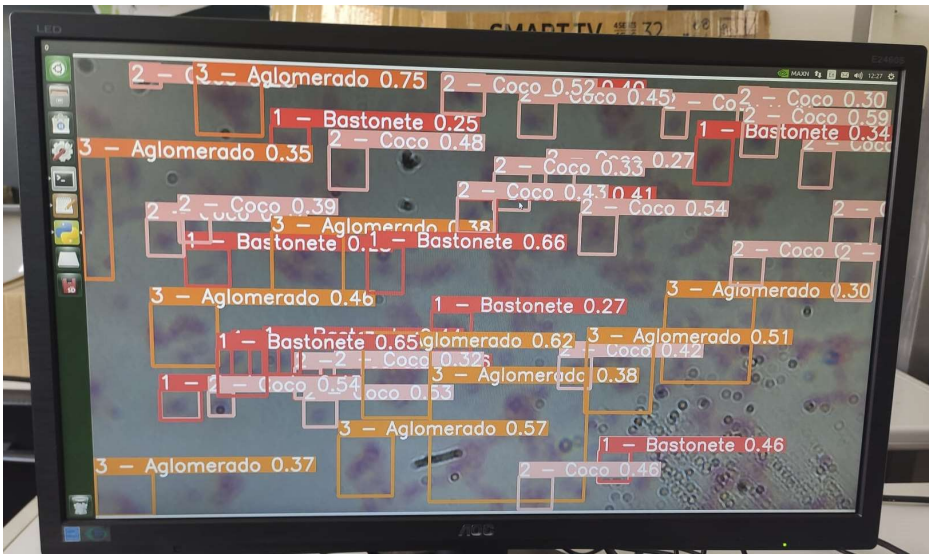


Figure 55 – Real time image classification results - 2

The initial results were very encouraging as the system was capable of detecting microorganisms present on the sample lamella very well. A large number of bounding boxes with varying degrees of prediction score are visible on the screen. This can be fine-tuned by increasing this threshold value prior to executing the detection command. In this case, the value was left at default, which is considerably low. If increased to, for example, 50%, only the objects with a prediction score equal or greater than that value would be classified with a bounding box.

6 Conclusion

The original goal of this project was to develop a computer vision model based on DNN that could detect E. coli bacteria in real time. This was a long process because many steps, such as acquiring the images and building them into a working dataset, were very laborious and time consuming. Working through the setup of the Jetson Nano also involved a long learning curve, which also translates into a significant time investment.

The first phase of testing resulted in an overall precision of 30% and a recall of 40%. These values are extremely low, so further tests were conducted using different techniques. It became clear that the problem was the small number of images, which were also too large for YOLOv5s. Finally, the images were reworked, the dataset was rebuilt and increased in size by a factor of three, and the model was trained again. This produced better results, with a precision of 70% and a recall of 60%. Overall, the model's ability to perform the desired detection and classification tasks increased by a factor of almost 2 times, making this a success.

The second part of the project focused on taking the weight file from the trained model and implementing it on the Jetson Nano platform. As mentioned above, this involved a considerable amount of effort and learning, as the system was reconfigured from scratch and many libraries and other dependencies were required to run the YOLOv5s correctly. After executing YOLOv5s on the Jetson Nano using the trained weights file, the system was able to recognize sample images of microorganisms and classify the images in real time. This was a great success, as the objects in the bounding boxes were correctly classified, in some cases with a high degree of accuracy.

The project benefited from all of the steps taken throughout the process. However, there is room for further development and improvement. One of the key points for improvement is the data set. Considering the increase in metric scores after the dataset was increased in size, it is understood that adding even more images to the existing dataset would further contribute to an increase in training metrics and model efficiency.

The next step to further improve the setup would be to study the YOLOv5 architecture and identify structures or parameters that could be modified to improve detection accuracy and increase inference speed by reducing computational cost.

References

[1] Shorten, C., & M. Khoshgoftaar, T. (2019). A survey on Image Data Augmentation for Deep Learning (pp. 1–20)

[2] Pacal, I., Karaman, A., Karaboga, D., Akay, B., Basturk, A., Nalbantoglu, U., & Coskun, S. (2022). An efficient real-time colonic polyp detection with YOLO algorithms trained by using negative samples and large datasets (pp. 1–11)

[3] Wallner, C., Alam, M., Drysch, M., Wagner, J. M., Sogorski, A., Dadras, M., Glinski, M. von , Reinkemeier , F., Becerikli, M., Heute, C., Nicolas , V., Lehnhardt, M., & Behr, B. (2021). A Highly Reliable Convolutional Neural Network Based Soft Tissue Sarcoma Metastasis Detection from Chest X-ray Images: A Retrospective Cohort Study. Cancers MDPI, 1–2.

[4] Zamir, S. W., Arora, A., Khan, S., Hayat, M., Khan, F. S., Yang, M.-H., & Shao, L. (2022). Learning Enriched Features for Fast Image Restoration and Enhancement. 2–6.

[5] Qiu, Z., Zhao, Z., Chen, S., Zeng , J., Huang, Y., & Xiang, B. (2022). Application of an Improved YOLOv5 Algorithm in Real-Time Detection of Foreign Objects by Ground Penetrating Radar. Remote Sensing MDPI, 10–12.

[6] Kanagaraj, N., Hicks, D., Goyal, A., Tiwari, S., & Singh, G. (2021). Deep learning using computer vision in self driving cars for lane and traffic sign detection. 1–6.

[7] von Chamier, L., Laine, R. F., Jukkala, J., Spahn, C., Krentzel, D., Nehme, E., Lerche, M., Hernández-Pérez, S., Mattila, P. K., Karinou, E., Holden, S., Solak, A. C., Krull, A., Buchholz, T.-O., Jones, M. L., Royer, L. A., Leterrier, C., Shechtman, Y., Jug, F., & Heilemann, M. (2021). Democratising deep learning for microscopy with ZeroCostDL4Mic. Nature Communications, 12(1). https://doi.org/10.1038/s41467-021-22518-0

[8] Chen, W., Liu, P.-Y., Lai, C.-C., & Lin, Y.-H. (2022). Identification of environmental microorganism using optimally fine-tuned convolutional neural network. Environmental Research, 206, 112610–112610. https://doi.org/10.1016/j.envres.2021.112610

[9] Yang, H., Li, C., Zhao, X., Cai, B., Zhang, J., Ma, P., Zhao, P., Chen, A., Jiang, T., Sun, H., Teng, Y., Qi, S., Jiang, T., & Marcin Grzegorzek. (2023). EMDS-7:

Environmental microorganism image dataset seventh version for multiple object detection evaluation. 14. https://doi.org/10.3389/fmicb.2023.1084312

[10] Sun, L., Xu, Y., Rao, Z., Chen, J., Liu, Z., & Lu, N. (2022). YOLO Algorithm for Long-Term Tracking and Detection of Escherichia Coli at Different Depths of Microchannels Based on Microsphere Positioning Assistance. 1–5.

[11] Park, J., Baek, J., Kim, J., You, K., & Kim, K. (2022). Deep Learning-Based Algal Detection Model Development Considering Field Application. Water, 14(8), 1275. https://doi.org/10.3390/w14081275

[12] Jubayer, F., Soeb, J. A., Mojumder, A. N., Paul, M. K., Barua, P., Kayshar, S., Akter, S. S., Rahman, M., & Islam, A. (2021). Detection of mold on the food surface using YOLOv5. Current Research in Food Science, 4, 724–728. https://doi.org/10.1016/j.crfs.2021.10.003

[13] Ultralytics | Revolutionizing the World of Vision AI. (n.d.). Www.ultralytics.com. Retrieved November 15, 2023, from https://www.ultralytics.com

[14] Maria Rocha, M. (2022). Object Detection in GPR Data Using Classical Vision and YOLOv5 (pp. 1–15).

[15] Ma, L., Yi, J., Nicharee Wisuthiphaet, Earles, M., & Nitin Nitin. (2022). Accelerating the Detection of Bacteria in Food Using Artificial Intelligence and Optical Imaging. Applied and Environmental Microbiology, 89(1). https://doi.org/10.1128/aem.01828-22

[16] E, S. K. (2020, August 21). Convolutional Neural Network | Deep Learning. Developers Breach. https://developersbreach.com/convolution-neural-network-deep-learning/

[17] Stewart, M. (2019, February 26). Simple Introduction to Convolutional Neural Networks. Medium; Towards Data Science. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

[18] Pooling (CNN) — EpyNN 1.0 documentation. (n.d.). Epynn.net. https://epynn.net/Pooling.html

[19] SuperDataScience. (n.d.). Www.superdatascience.com. https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

[20] Guo, Z., Wang, C., Yang, G., Huang, Z., & Li, G. (2022). MSFT-YOLO: Improved YOLOv5 Based on Transformer for Detecting Defects of Steel Surface. Sensors, 22(9), 3467. https://doi.org/10.3390/s22093467

[21] Li, a href=’http://yixuanli net/’>Sharon Y. (2020, April 24). Automating Data Augmentation: Practice, Theory and New Direction. SAIL Blog. https://ai.stanford.edu/blog/data-augmentation/

[22] Jaikumar, P., Vandaele, R., & Ojha, V. (2021). Transfer Learning for Instance Segmentation of Waste Bottles using Mask R-CNN Algorithm (pp. 2–5).

[23] Bhavsar, P. (2020, September 13). Transfer Learning In NLP. Medium. https://medium.com/modern-nlp/transfer-learning-in-nlp-f5035cc3f62f

[24] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (n.d.). You Only Look Once: Unified, Real-Time Object Detection (pp. 1–10.

[25] Qiu, Z., Zhao, Z., Chen, S., Zeng, J., Huang, Y., & Xiang, B. (2022). Application of an Improved YOLOv5 Algorithm in Real-Time Detection of Foreign Objects by Ground Penetrating Radar. Remote Sensing, 14(8), 1895. https://doi.org/10.3390/rs14081895

[26] Ultralytics. (n.d.). Architecture Summary. Docs.ultralytics.com. https://docs.ultralytics.com/yolov5/tutorials/architecture_description/

[27] Kateb, F. A., Monowar, M. M., Hamid, Md. A., Ohi, A. Q., & Mridha, M. F. (2021). FruitDet: Attentive Feature Aggregation for Real-Time Fruit Detection in Orchards. Agronomy, 11(12), 2440. https://doi.org/10.3390/agronomy11122440

[28] Lee, Y., An, J., & Joe, I. (2022). Deep-Learning-Based Object Filtering According to Altitude for Improvement of Obstacle Recognition during Autonomous Flight. Remote Sensing, 14(6), 1378. https://doi.org/10.3390/rs14061378

[29] Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and Efficient Object Detection (pp. 5–7).

[30] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (n.d.). Rich feature hierarchies for accurate object detection and semantic segmentation (pp. 1–6).

[31] Rosebrock, A. (2020, June 29). OpenCV Selective Search for Object Detection. PyImageSearch. https://pyimagesearch.com/2020/06/29/opencv-selective-search-for-object-detection/

[32] From R-CNN to Faster R-CNN – The Evolution of Object Detection Technology. (n.d.). Alibaba Cloud Community. Retrieved November 15, 2023, from https://www.alibabacloud.com/blog/from-r-cnn-to-faster-r-cnn-the-evolution-of-object-detection-technology_593829

[33] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (n.d.). Region-based Convolutional Networks for Accurate Object Detection and Segmentation (pp. 4–7).

[34] Mahendrakar, T., Ekblad, A., Fischer, N., White, R., Wilde, M., Kish, B., & Silver, I. (2022, March 1). Performance Study of YOLOv5 and Faster R-CNN for Autonomous Navigation around Non-Cooperative Targets. IEEE Xplore. https://doi.org/10.1109/AERO53065.2022.9843537

[35] Zamir, S. W., Arora, A., Khan, S. H., Munawar, H., Khan, F. S., Yang, M.-H., & Shao, L. (2022). Learning Enriched Features for Fast Image Restoration and Enhancement. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1–1. https://doi.org/10.1109/tpami.2022.3167175

[36] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, Xindong Wu, Object Detection with Deep Learning: A Review, 2019

[37] Murthy, C. B., Hashmi, M. F., Bokde, N. D., & Geem, Z. W. (2020). Investigations of Object Detection in Images/Videos Using Various Deep Learning Techniques and Embedded Platforms—A Comprehensive Review. Applied Sciences, 10(9), 3280. https://doi.org/10.3390/app10093280

[38] Ma, P., Li, C., Rahaman, M. M., Yao, Y., Zhang, J., Zou, S., Zhao, X., & Grzegorzek, M. (2022). A state-of-the-art survey of object detection techniques in microorganism image analysis: from classical methods to deep learning approaches. Artificial Intelligence Review. https://doi.org/10.1007/s10462-022-10209-1

[39] Spahn, C., Laine, R. F., Pereira, P. M., Estibaliz Gómez-de-Mariscal, Lucas von Chamier, Conduit, M., Pinho, M. G., Holden, S., Guillaume Jacquemet, Heilemann, M., & Henriques, R. (2021). DeepBacs: Bacterial image analysis using open-source deep learning approaches. BioRxiv (Cold Spring Harbor Laboratory). https://doi.org/10.1101/2021.11.03.467152

[40] Park, J., Baek, J., Kim, J., You, K., & Kim, K. (2022). Deep Learning-Based Algal Detection Model Development Considering Field Application. Water, 14(8), 1275. https://doi.org/10.3390/w14081275

[41] Sign in to Roboflow. (n.d.). App.roboflow.com. https://app.roboflow.com/

[42] Zamir, S. W. (2023, November 15). Learning Enriched Features for Fast Image Restoration and Enhancement (TPAMI 2022). GitHub. https://github.com/swz30/MIRNetv2

[43] LLC, I. S. (n.d.). ImageMagick. ImageMagick. https://imagemagick.org/

[44] Li, Y., Bai, X., & Xia, C. (2022). An Improved YOLOV5 Based on Triplet Attention and Prediction Head Optimization for Marine Organism Detection on Underwater

Mobile Platforms. Journal of Marine Science and Engineering, 10(9), 1230. https://doi.org/10.3390/jmse10091230

[45] NVIDIA Jetson Nano Developer Kit-B01. (2023, September 11). Www.seeedstudio.com. https://www.seeedstudio.com/NVIDIA-Jetson-Nano-Development-Kit-B01-p-4437.html

[46] Ultralytics. (n.d.). YOLOv8. Docs.ultralytics.com. https://docs.ultralytics.com/models/yolov8/

[47] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM, 60(6), 84–90. https://doi.org/10.1145/3065386

[48] Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., & Lopez, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. Neurocomputing, 408(1), 189–215. https://doi.org/10.1016/j.neucom.2019.10.118

[49] Gad, A. F. (2020, October 16). Mean Average Precision (mAP) Explained. Paperspace Blog. https://blog.paperspace.com/mean-average-precision/

[50] Li, Y., Wei, H., Han, Z., Huang, J., & Wang, W. (2020). Deep Learning-Based Safety Helmet Detection in Engineering Management Based on Convolutional Neural Networks. Advances in Civil Engineering, 2020, 1–10. https://doi.org/10.1155/2020/9703560

[51] PyTorch for Jetson. (2019, March 27). NVIDIA Developer Forums. https://forums.developer.nvidia.com/t/pytorch-for-jetson/72048

[52] Yamaguchi, N., Tokunaga, Y., Goto, S., Fujii, Y., Banno, F., & Edagawa, A. (2017). Rapid on-site monitoring of Legionella pneumophila in cooling tower water using a portable microfluidic system. Scientific Reports, 7(1). https://doi.org/10.1038/s41598-017-03293-9