Internship Project

SMART DRONE NAVIGATION FOR OBJECT RECOGNITION

REPORT

By:

MESSAOUDI Oussama

Supervied by:

Mr. HMAMOUCHE Youssef

Pr. SEGHROUCHNI EL FALAH Amal

Academic supervision:

Mr. HOSNI Mohamed

Mr. MASROUR Tawfik

15 February - 15 August 2022

Acknowledgements

Having to work on this end of studies project, in this specific topic, in this promising domain, and under the guidance and supervision of such great people, is a blessing, and an opportunity, that is not given to everyone.

First and foremost, I would like to express my gratitude and appreciation to my supervisor, ***Mr. HMAMOUCH Youssef****, whose guidance, immense knowledge, motivation, and support were invaluable throughout this project. I'd also like to thank* ***Pr. EL FALLAH SEGHROUCHNI Amal*** *for the support, resources, and the opportunity to be part of such a great vision of hers.*

I am extremely grateful to ***Mr. HOSNI Mohamed*** *and* ***Mr. KHADIRI Hassan*** *for their academic guidance, valuable advice and dedicated effort helping me work on and present this project in the best possible conditions.*

I cannot express enough gratitude enough thanks to ***Mr. MASROUR Tawfik****, without whom, this whole framework and direction wouldn't be possible, his efforts, knowledge, motivation, and guidance had and still have a great impact on my work and way of thinking.*

I would like to thank ***my family****, their encouragement, support, and spiritual touch had the deepest impact on my path in life, my way of thinking and in particular, the work I have done in this project.*

Finally, the completion and the shape of this work wouldn't be possible without the efforts and guidance of all teachers and staff of ENSAM school, whom I owe a debt of gratitude, as well as for jury members who had me honored by accepting my work.

Abstract

The work presented in this report is carried out within **Ai movement** -The International Center of Artificial Intelligence, of Mohammed VI Polytechnic University. It is part of the end-of-studies internship project.

It concerns the study and development of two main aspects of autonomous supervision of industrial sites using drones, namely navigation and object recognition. This autonomy becomes necessary when considering large and complex sites, such as OCP sites, and the use of artificial intelligence, in particular, supervised machine learning (ML) and reinforcement learning (RL) would increase the efficiency of the monitoring and supervision process.

Computer vision and machine learning play an important role in the field of object recognition and being able to develop and deploy robust object detection models will certainly have a significant positive impact on the accuracy and cost of monitoring.

This work aims at studying the state of the art of artificial intelligence techniques used in the field of object recognition on one hand, in particular computer vision and deep learning (DL), and to develop and customize these techniques to be compatible with our context. On the other hand, we address the problem of autonomous navigation with drones for surveillance purposes using coverage path planning modelling. We adopt the same methodology as in the first part by studying, implementing, and testing different relevant mathematical and reinforcement learning models.

Keywords: Computer Vision, Deep Learning, Object Detection, Navigation, Reinforcement Learning.

Resumé

Le travail présenté dans ce rapport est réalisé au sein du **Ai movement** - The International Center of Artificial Intelligence, de l'Université Polytechnique Mohammed VI. Il s'inscrit dans le cadre du projet de stage de fin d'études.

Il concerne l'étude et le développement de deux aspects principaux de la supervision autonome des sites industriels à l'aide de drones, à savoir la navigation et la reconnaissance d'objets. Cette autonomie devient nécessaire lorsqu'on considère des sites vastes et complexes, tels que les sites OCP, et l'utilisation de l'intelligence artificielle, en particulier l'apprentissage automatique (ML) supervisé et l'apprentissage par renforcement (RL), permettrait d'augmenter l'efficacité du processus de surveillance et de supervision.

La vision par ordinateur et l'apprentissage automatique joue un rôle important dans le domaine de la reconnaissance d'objets et le fait de pouvoir développer et déployer des modèles de détection d'objets robustes aura certainement un impact positif significatif sur la précision et le coût de la surveillance.

Ce travail vise à étudier l'état de l'art des techniques d'intelligence artificielle utilisées dans le domaine de la reconnaissance d'objets d'une part, en particulier la vision par ordinateur et l'apprentissage profond (DL), et à développer et personnaliser ces techniques pour qu'elles soient compatibles avec notre contexte. D'autre part, nous abordons le problème de la navigation autonome avec des drones à des fins de surveillance en utilisant la modélisation de la planification des chemins de couverture. Nous adoptons la même méthodologie que dans la première partie en étudiant, implémentant et testant différents modèles mathématiques et d'apprentissage par renforcement pertinents.

Mots-clés : Vision par ordinateur, apprentissage profond, détection d'objets, navigation, apprentissage par renforcement.

الملخص

يتم تنفيذ العمل المعروض في هذا التقرير لدى **Ai movement** - المركز الدولي للذكاء الاصطناعي التابع لجامعة محمد السادس للفنون التطبيقية. إنه جزء من مشروع تدريب نهاية الدراسة.

يتعلق هذا المشروع بدراسة وتطوير جانبين رئيسيين للإشراف الأوتوماتيكي على المواقع الصناعية باستخدام الطائرات بدون طيار ، وهما الملاحة والتعرف على الأشياء. تصبح هذه الأوتوماتيكية ضروريَّة عند اعتبار المواقع الكبيرة والمعقدة واستخدام الذكاء الاصطناعي ، على وجه الخصوص ، التعلم الآلي الخاضع للإشراف والتعلم المعزز من شأنه ، أن يزيد من كفاءة عملية المراقبة والإشراف.

تلعب رؤية الكمبيوتر والتعلم الآلي دورًا مهمًا في مجال التعرف على الأشياء ، كما أن القدرة على تطوير نماذج قوية للكشف عن الأشياء سيكون لها بالتأكيد تأثير إيجابي كبير على دقة وتكلفة المراقبة.

يهدف هذا العمل إلى دراسة أحدث تقنيات الذكاء الاصطناعي المستخدمة في مجال التعرف على الأشياء من جهة ، ولا ، سيما الرؤية الحاسوبية والتعلم العميق ، وتطوير وتخصيص هذه التقنيات لتكون متوافقة مع سياقنا .من ناحية أخرى نعالج مشكلة الملاحة الأوتوماتيكية المستقلة باستخدام الطائرات بدون طيار لأغراض المراقبة باستخدام نمذجة تخطيط مسار التغطية .نعتمد نفس المنهجية كما في الجزء الأول من خلال دراسة وتنفيذ واختبار نماذج التعلم الرياضية والتعزيزية المختلفة ذات الصلة.

الكلمات الرئيسية: رؤية الكمبيوتر ، التعلم العميق ، اكتشاف الأشياء ، الملاحة ، التعلم المعزز.

List of figures

List of tables

List of abbreviations

AI: Artificial Intellignce……15

CPP: Coverage Path Planning……...77

DL: Deep Learning…….. ……..……02

FN : False Negative……...……...…….47

FP: False Positive……..……47

IoU: Intersection over Union……47

mAP: Mean Average Precision……..30

ML: Machine Learning ……02

OCP : Office Chérifien des Phosphates……...……..12

P: Polynomial time complexity…….....81

RL: Reinforcement Learning…….……02

NP: Non-deterministic Polynomial Time complexity……...81

TFOD: TensorFlow Object Detection……...…….…...62

TN : True Negative ……...……...……47

TP: True Positive……...……...……47

UAV: Unmanned Aerial Vehicle……..….24

YAML: YAML Ain't Markup Language……..….62

YOLO: You Only Look Once…….…...20

Content

General introduction

The different parts of OCP's site El Jorf Lasfar are large, sometimes dangerous, and difficult to supervise by humans, in a continuous and efficient way, even when using fixed cameras, and especially when conditions are inconceivable (high temperature, low light, ...). An autonomous surveillance will allow for better site management, increase the efficiency, and minimize the costs.

The use of drones and artificial intelligence to enhance industrial efficiency is spreading through many large organizations and companies across the world. This project is framed in this context, and we aim by it to study, develop and integrate the technology needed for the deployment of a system responding to the challenges of surveillance from both an object.

This project focuses on two main aspects of using drones for supervision, which are object recognition and drone navigation. That's why our project is divided into two parts: The first one concerns object detection and classification using image data, and the second is about the autonomous navigation of the drone in an open space, under some defined constraints to handle.

The first chapter of the first part discussed computer vision in general, its definition and use cases, its main components and the methodology used in it with the integration of deep convolutional neural networks.

The second chapter of the first part details the state of the art of both image classification and object detection. It introduces major achievements and breakthroughs in the field, and it shows significant characteristics of widely used architectures. It also contains results of our conducted implementations regarding every introduced architecture.

The third chapter is where we introduce our work and results, it has two main parts, the first is about the construction and customization of the datasets to meet our needs, and the second is about the training pipelines adopted to construct the proposed models. We also perform a comparison between the results of every model.

The first chapter of the second part consists of a presentation of the online coverage path planning problem, the demonstration that it's an NP-hard problem as well as an introduction to various mathematical results in the field and its relevance to out project.

We discuss in the second chapter a methodology to solve the online coverage path planning with energy constraints with a logarithmic complexity as well as our new proposed method that makes use of reinforcement learning, and we explain aspects taken into account while designing it.

Chapter 1: Organization and general context

Organization

AI movement

Ai movement, the Moroccan International Center for Artificial Intelligence is a center of excellence in Artificial Intelligence that aims to foster the emergence of Moroccan expertise in Artificial Intelligence and Data Sciences. It is both:

An articulating and consolidating tool of various actions related to the field of AI, with the ambition of making Morocco a regional AI hub impacting its ecosystem, on strategic, educational, and industrial levels.

A lever to anticipate and accompany the evolutions and transformations related to Artificial Intelligence and Data Sciences, the aim of which is to provide innovative, operational, resilient, and ethical solutions to the problems of society, environment, market, economy and technology.

*Figure 1: **Ai movement** logo*

Mission

As a hub for Artificial Intelligence transformation, **Ai movement** is based on 7 pillars that structure and define its missions:

- Foster the emergence of an attractive ecosystem
- Attract international expertise in AI and Data Sciences to work in collaboration with national researchers.
- Create international synergies and bring out national talent.
- Create and strengthen international quality partnerships.
- Develop a field-oriented approach

*Figure 2: **Ai movement** mission elements*

- Rise society's awareness and understanding to accept transformations related to the dematerialization of services and AI.
- Leading change in a way that is adapted to different strata of society.
- Study the various fields and identify the needs and expectations of different societal strata.

*Figure 3: **Ai movement** research axes*

Ai movement focuses on a variety of axes of research, that are related to artificial intelligence, and would have a positive impact on both society and industry. Figure 3 illustrates those axes, as well as the corresponding projects and research concepts.

General context of the project

The different parts of OCP's site El Jorf Lasfar are large, sometimes dangerous, and difficult to supervise by humans, in a continuous and efficient way, even when using fixed cameras, and especially when conditions are inconceivable (high temperature, low light, ...). An autonomous surveillance will allow for better site management, increase the efficiency, and minimize the costs.

Figure 4: OCP site

The use of drones and artificial intelligence to enhance industrial efficiency is spreading through many large organizations and companies across the world. This project is framed in this context, and we aim by it to study, develop and integrate this technology in our context, for the many advantages of its deployment, using state of the art techniques.

This project focuses on two main aspects of using drones for supervision, which are object recognition and drone navigation. That's why our project is mainly divided into two parts: The first one concerns object detection and classification using image data, and the second is about the autonomous navigation of the drone in an open space, under some defined constraints to handle.

To find the best solutions to the problem constraints, it is necessary to ask the right questions first. An ultimate goal of deploying drones for the purpose of surveillance in complex

environments requires having well defined guidelines while seeking solutions to that challenge, in the context of this project, we seek to answer the next research questions:

- What is the best technical approach to handle both the problem of object recognition and drone navigation?
- What types of constraints should be taken into consideration while designing the model of detection?
- What frameworks and tools to use?
- What factors have the most impact on the performance of the model?
- How to correctly collect and construct datasets necessary for the development of models?
- How to evaluate and handle the hardware specifications?

We give answers to all these questions and many more in the context of this project.

Part 1: Computer Vision

As stated earlier, OCP sites are large and difficult to supervise using traditional and manual methods, thus increasing the efficiency of the surveillance process needs more sophisticated methods and techniques, which might yield better performance metrics to the different components to the overall surveillance system, and we focus in this first part of the project on the object detection component.

Our main goal in this first section is to discover, study and customize state of the art object detection models in order to be able to detect and classify objects that might exist in OCP sites and ports, for an ultimate objective of surveillance and supervision.

We aim to develop models with very good performance metrics. The input of the models will be a video streaming or a consecutive set of images of any size, and the output of them will be the same type of the input with the same size with the addition of bounding boxes containing the detected objects.

Deep learning models are the state-of-the-art object detection models that yield the best accuracy compared to other traditional approaches, as well as their ability to be trained on specific datasets containing targeted objects, this makes them the right call for us in this project.

Following the deep learning model development approach, this section is divided into two main axes:

- Building and customizing datasets that are suitable for training the models: Customized VisDrone and Ships dataset (built from scratch).
- Studying, training and evaluating state of the art object detection models and choosing the best fit among them for implementation.

In the first axis, we aim to customize existing dataset called VisDrone to be compatible with our task and context, and we also not only customize but also collect data from scratch in order to integrate objects that no other dataset on the internet and a need to be detected on the targeted sites, like industrial ships for example.

Regarding the second axis, we study, re-implement using our own code, and train, using the available set of hardware configuration, state of the art models like YOLOv5, FasterRCNN, SSD, EffecientDet and CenterNet, with different backbone architectures and hyperparameters, and we evaluate their results based on their accuracy metrics values and processing speed.

Chapter 1: Computer vision generalities

Chapter 1: Computer vision

Before getting into the process of image classification or object detection, we first need to understand the basics of image processing and computer vision, to be able to know exactly what we are manipulating with our models later. This section introduces the foundations of computer vision and its building blocks.

I.1.1 Definition

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos, and other visual inputs — and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand [1].

Figure 5: The output of Faster R-CNN on an image taken by a drone

Computer vision uses a combination of image sensors, data and algorithms to get insights and create valuable information. As an example, figure 5 presents a set of bounding boxes that have been added to the initial image by the model, marking the existing objects in the image. Figure 5 is the output of our trained model FasterRCNN trained on the original VisDrone dataset, that we will discuss in more detail later.

So, the main task of computer vision is the extraction of high dimensional information from the real-world data in order to help in the process of decision making, which is in our case the actions that might result from the supervision of industrial sites. Computer vision aims to make

sense of the world that is presented to it as visual images and transform them into clear descriptions ready to aid and elicit appropriate action.

Sub-domains of computer vision include scene reconstruction, object detection, event detection, video tracking, object recognition, 3D pose estimation, learning, indexing, motion estimation, 3D scene modeling, and image restoration [2].

Many great leaps were taken lately by computer vision thanks to general advances of artificial intelligence and innovations in deep learning and neural networks, as humans were relatively surpassed on the technical aspect in some tasks related to detecting labeling objects [3].

I.1.2 Use cases

By 2022, it is expected that the computer vision and related hardware market will reach $48.6 billion [1]. Many industries use computer vision for the many powerful and compelling aspects that it has, and its proven added value to many industrial sectors, ranging from energy and utilities to manufacturing and automotive.

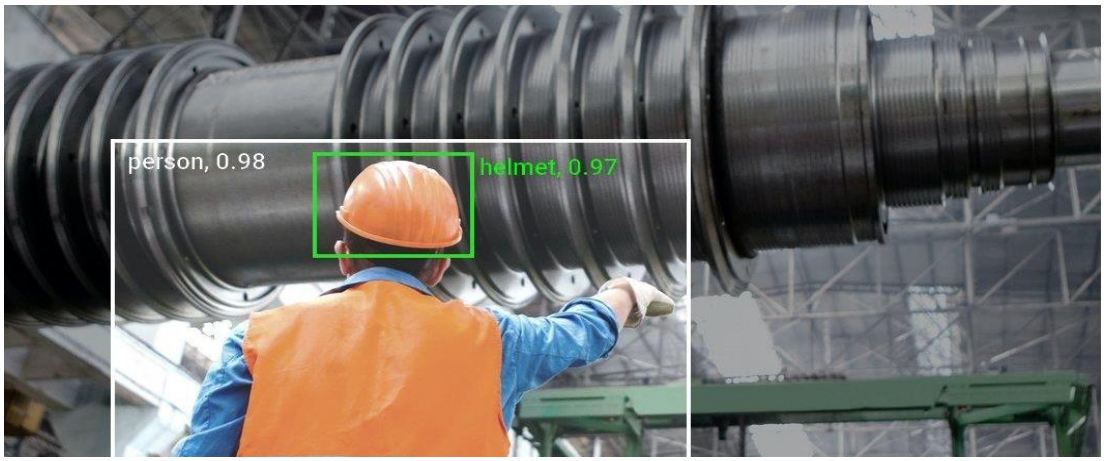


Figure 6: The use of computer vision in the industry

And as the figure 6 shows, computer vision can have a variety of applications in manufacturing processes, ranging from machine vision systems to computers or robots that can comprehend the world around them. Many examples can be stated in this context:

- Automatic quality inspection in manufacturing
- Industrial controlling processes

- Optimizing supply chains
- Equipment monitoring and predictive maintenance
- Safety of workforce and digital lean manufacturing

Those manufacturing applications can be added to a longer list of many other ones, where computer vision has proven its great utility and efficiency. Furthermore, computer vision has also a great utility in other totally different fields, such as medicine, where medical image processing aids to diagnose a patient based on information extracted from image data. An example of this is the detection of tumors, arteriosclerosis, measurements of organ dimensions, blood flow, providing information about the structure of the brain, etc.

Military sector also benefits from the power of computer vision, from object detection to missile guidance as well as targeting specific entities or providing a set of information about a combat scene which can be used to support strategic decisions.

Computer vision has another great utility in the development of autonomous vehicles, which include submersibles, land-based vehicles (small robots with wheels, cars or trucks), aerial vehicles and unmanned aerial vehicles (UAV). Systems based on computer vision support a driver or a pilot in various situations, for example in navigation, or in producing a map of its environment or for detecting obstacles. The level of autonomy given by these systems range from fully autonomous (unmanned) vehicles to vehicles.

I.1.3 Main Components

The building block of all the computer vision process is the image, it's the raw material that we seek to extract meaningful information from, as the previous definition of computer vision states. The image itself has another foundational building block which the pixel, and this is what makes the task of extraction very challenging is that an image is represented on the low level as a two-dimensional matrix with numerical values between 0 and 255, and three channels that represent the basic colors: red, green and blue.

As you can see in the figures below, a truck is easily detected in the image by us humans, but the same picture is represented as a two-dimensional matrix with three channels on the

Figure 5: Image of a truck

151	121	1	93	165	204	14	214	28	235
62	67	17	234	27	1	221	37	189	141
20	168	155	113	178	228	25	130	139	221
236	136	158	230	10	5	165	17	30	155
174	148	93	70	95	106	151	10	160	214
103	126	58	16	138	136	98	202	42	233
235	103	52	37	94	104	173	86	223	113
212	15	179	139	48	232	194	46	174	37
119	81	241	172	95	170	29	210	22	194
129	19	33	253	229	5	152	233	52	44
88	200	194	185	140	200	223	190	164	102
113	16	220	215	143	104	247	29	97	203
9	210	102	246	75	9	158	104	184	129
124	52	76	148	249	107	65	216	187	181
6	251	52	208	46	65	185	38	77	240
150	194	28	206	148	197	208	28	74	93
33	183	248	153	168	205	146	100	254	218
130	53	128	212	61	226	201	110	140	183
165	246	22	102	151	213	40	138	8	93
152	251	101	230	23	162	70	238	75	24
187	105	152	83	167	98	125	180	136	121
139	197	55	209	28	124	208	208	104	40
123	19	144	223	62	253	202	108	47	242
220	144	31	16	136	123	227	62	183	163

Figure 6: Representation of an image on the computer

computer.

So, the main challenge then is to be able to turn the quantity of structured numbers into meaningful amount of information, that would describe the contents if the images in a similar way that a human could do, minimizing what's called *the semantic gap*, which is the difference between how a human perceives the contents of an image versus how an image can be presented in a way a computer can understand the process [11].

We might for example describe the figure 7 as follows:

- Spatial: The sky is at the top of the image and the land/ocean are at the bottom.
- Color: The sky is dark blue, the ocean water is a different shade of blue, while other objects have other colors.
- Texture: The sky has a relatively uniform pattern, while the land is very diverse.

Figure 7: The semantic gap in a port

The design of the algorithm should take into consideration this semantic gap, as well as many other challenges, for example the factors of variation that should be handled as well.

Figure 8: Viewpoint variations

Figure 8 displays a visualization of a number of these factors of variation: viewpoint variation, scale variation, deformation, occlusion variation, illumination variation, background clutter, intra-class variation. So how do we go about treating all these constraints and seek significant results? The answer is in the following methodology section.

I.1.4 Methodology

The answer to encoding all the information contained in an image, as mentioned in the section before, in a way that a computer can understand is applying feature extraction, in order to quantify the contents of a given image. Feature extraction is the process of taking an input image, applying an algorithm, and obtaining a feature vector that quantifies the image.

There are two ways to tackle this problem, the first one is by applying hand-engineered features such as HOG, LBPs, or other "traditional" approaches to image quantifying. The other way, which is the one we are going to follow, is to apply deep learning to automatically learn a set of features that can be used to quantify and ultimately label the contents of the image itself.

I.1.4.1 Convolutional neural networks

Deep learning and convolutional neural networks have demonstrated significant robustness and classification power under a variety of challenges, this what makes it the best choice for us in handling our given problem.

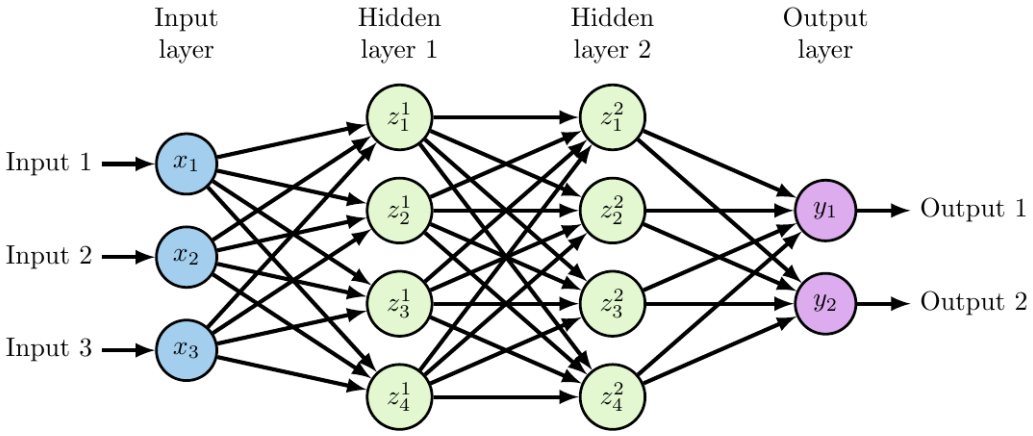


Figure 9: A neural network

Deep learning is defined as a class of machine learning algorithms that use multiple layers to progressively extract higher-level features from the raw input. The figure 9 shows an example of a neural network.

Many concepts are included in the design of every neural network that help increase its performance, such as: the convolutional layers, the fully connected layers, the activation function, the pooling layers, the batch normalization, the dropout, learning rate, decay, regularization, etc.

Convolutional neural networks are a specialized type of artificial neural networks that use a mathematical operation called convolution in place of general matrix multiplication in at least one of their layers. They are specifically designed to process pixel data and are used in image recognition and processing.

Figure 10 illustrates two steps of the convolution operation between two matrices. In general, a convolutional layer performs an operation between in input matrix, and another feature detector, or kernel (with size 3*3, 5*5 or 9*9). The orange matrix is the input matrix, and the blue one is the kernel.

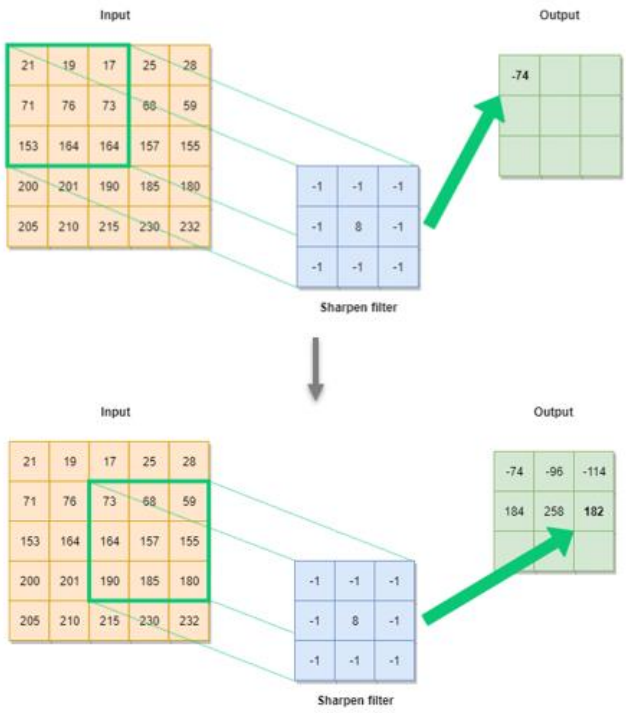


Figure 10: Convolution operation

The expression of the convolution formula used in the convolutional neural networks is:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n K(i-m, j-n) I(m,n)$$

Where:

- I: The input matrix.
- K: The kernel matrix.
- S: The output matrix.

The displacement of the kernel depends on the stride parameter, which defines the exact amount of movement over the image. For example, if the stride is 1, the filter will move one pixel, or unit at a time.

I.1.4.2 The learning pipelines

Constructing a deep learning model follows a well-defined process, containing many steps:

- Gathering the dataset:

  In this step, we work on gathering our initial dataset, which is the first component of building a deep learning network. We need the images themselves as well as the labels associated with the image. Labels must be represented as a finite set of categories, such as: categories = dog, cat, panda.

  Class imbalance should be considered when building the dataset, as the problem of overfitting into these heavily represented categories will make our model naturally biased, thus the importance of approximate uniformity across all categories regarding the number of images.

- Splitting the dataset:

  After having the initial dataset ready, splitting it into two parts is the next step: A training set and a testing set.

  A training set is used by the model to learn what each category looks like and make predictions on the input data and then adjust itself to maximize its performance and ability to get predictions right most of the time.

  After the model has been trained, evaluation should be performed on a testing set, and guarding a certain independence between this test set and the training set is an extremely important thing to consider, as this overlap avoidance won't give the model an unfair advantage by seeing testing examples before.

  Common split sizes for training and testing sets include: 66.6% - 33.3%, 75% - 25%, 90% - 10%.

- Training the network

  Given our collected dataset that consists of images and labels, now we can train our network. The goal here for the network is to learn how to classify or correctly detect the categories in the labeled data.

Deep learning neural networks learn a mapping function from input to output, and when the model makes a mistake while training, it learns from the mistake and improves itself, and this is achieved by updating the weights of the network in response to the errors the model makes on the training dataset.

Unlike other machine learning algorithms, the parameters of a neural network must be found by solving a non-convex optimization problem with many good solutions and many misleadingly good solutions.

The stochastic gradient descent algorithm is used to solve the optimization problem where model parameters are updated each iteration using the backpropagation algorithm.

- Evaluation

Model evaluation is the process of using different evaluation metrics to understand a machine learning model's performance, as well as its strengths and weaknesses. In the context of object detection, we mainly use the accuracy of predictions and processing speed as performance metrics. The mAP (mean average precision) is the metric used to evaluate an object detector's accuracy, which we will discuss in the next section.

Conclusion

This chapter introduces computer vision and its building blocks, which will facilitate getting into other advanced topics related to this concept in the context of our project.

Next chapter discusses the literature of image classification and object detection knowing its importance in understanding the procedures of training and testing, and every other concept in this context.

Chapter 2: Literature review

Chapter 2: Literature review

CNN are often compared to the way the brain achieves vision processing in living organisms. This is where the inspiration came at first, to imitate biosystems in nature, and many contributions after that have helped computer vision to rise and reach great achievements, as we talk now about great models in both detection and classification, that achieve state of the art performance metrics.

In this section, we go through the main contributions in both image classification and object detection models and architectures and dig in the techniques that allows them to achieve their significant results, and at the same time study them in a way that allows us to have a clear vision about their way of working and their parameters and hyperparameters.

We shall notice that during this project, we followed the approach of studying and testing at the same time, so that for example all the image classification architectures that we will be mentioning in the next section are architectures that we studied and wrote from scratch and tested them on state on the art datasets of image classification.

Before we start analyzing different models and architectures, we should first clarify the difference between classification and detection. Figure 11 shows the differences between the four tasks.

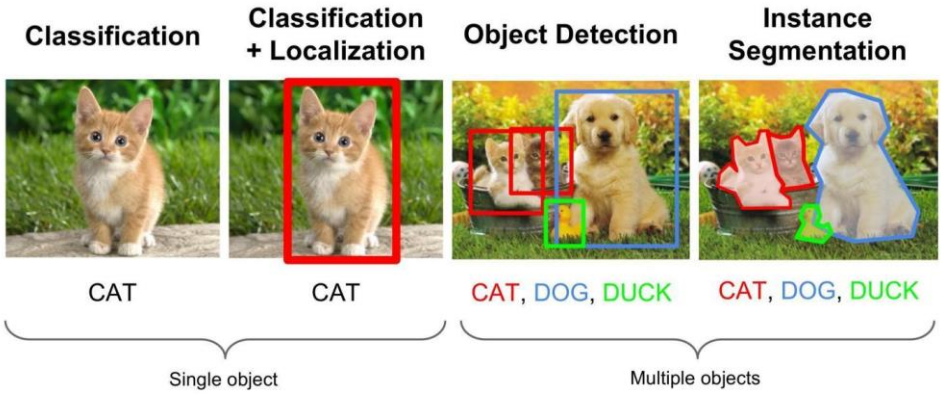


Figure 11: The difference between classification, localization, and detection

Classification of an image is associating it with a label from the set of classes in the dataset, while localization refers to the producing a bounding box that shows the limits of the region containing the object of the image, object detection is a localization and classification of many objects in the same image, and the segmentation refers to the process of detecting objects in

the image with an output of segments that would approximate the exact area of the object in the image.

We start by studying classification architecture as they are the backbones that object detectors are built upon. And after that we study the most popular models of object detection and their different approaches

I.2.1 Image Classification

Image classification is the task of associating one or more labels to a given image. In this section, we discuss the ImageNet dataset challenge, which is the challenge that shed the light on many of the state-of-the-art image classifiers, after turning the attention from traditional computer vision approaches as from 2012 and the remarkable results of AlexNet.

1.1 ImageNet

The ImageNet project is a large visual database designed for use in visual object recognition software research. It contains more than 14 million images that have been hand-annotated by the project, for more than 200,000 categories. Software programs started to compete to correctly classify and detect objects and scenes in an annual software contest as of 2010, called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The challenge uses a “trimmed” list of one thousand non-overlapping classes [5][6].

As you can see in the figure 12, the best performing classifiers appeared in the ImageNet contest, and after two years of using only the hand engineered feature extractors, deep learning took all attention from 2012 on, after the breakthrough of AlexNet, that achieved a top-5 error rate of 16.4% more than 10% lower than that of the runner up. We discuss in this section LeNet which is a deep learning classifier that was introduced in 1998, as well as the next architectures: AlexNet, VGGNet, GoogleNet and ResNet.

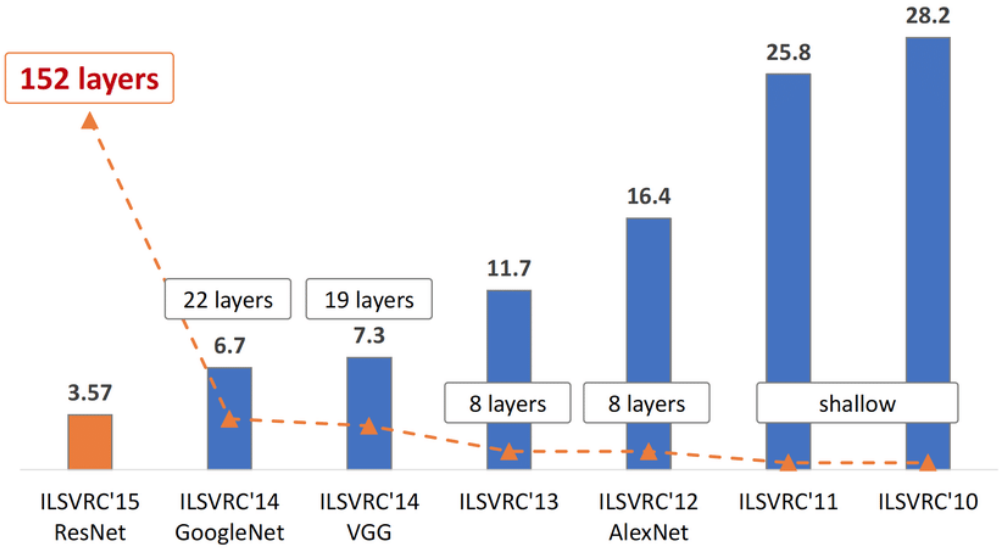


Figure 12: The ImageNet contest results

1.2 LeNet

LeNet was first introduced by LeCun and his team in their 1998 paper, Gradient-Based Learning Applied to Document Recognition. The figure 13 shows LeNet architecture as presented in the paper [7].

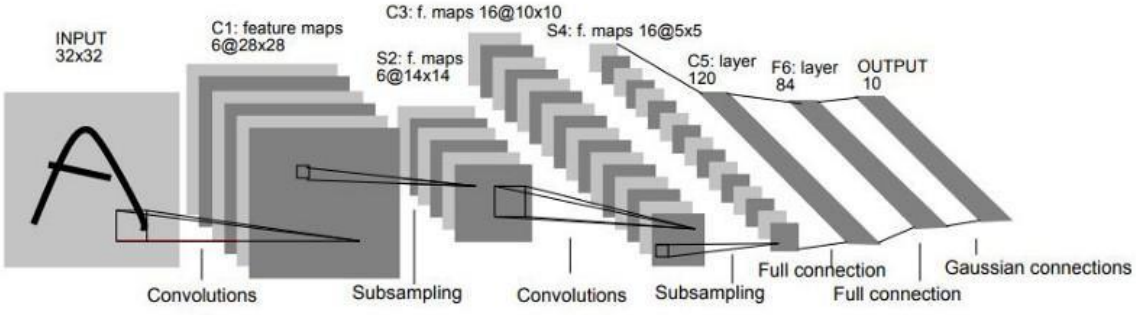


Figure 13: LeNet architecture

LeNet is an excellent real-world network, it is small and easy to understand yet large enough to provide interesting results. The Lenet architecture consists of two series of CONV => TANH => POOL layer sets followed by a fully connected layer and softmax output.

Implementation: We trained the LeNet architecture on the MNIST dataset, which has a goal of correctly classifying handwritten digits 0 - 9, and with 60,000 images in the training set and 10,000 images in the test set.

Result: This experiment yields an accuracy of 98%.

1.3 AlexNet

AlexNet is a deep convolutional neural network that achieved a top-5 error rate in the ImageNet Large Scale Visual Recognition Challenge in 2012. The original paper's result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of graphics processing units during training.

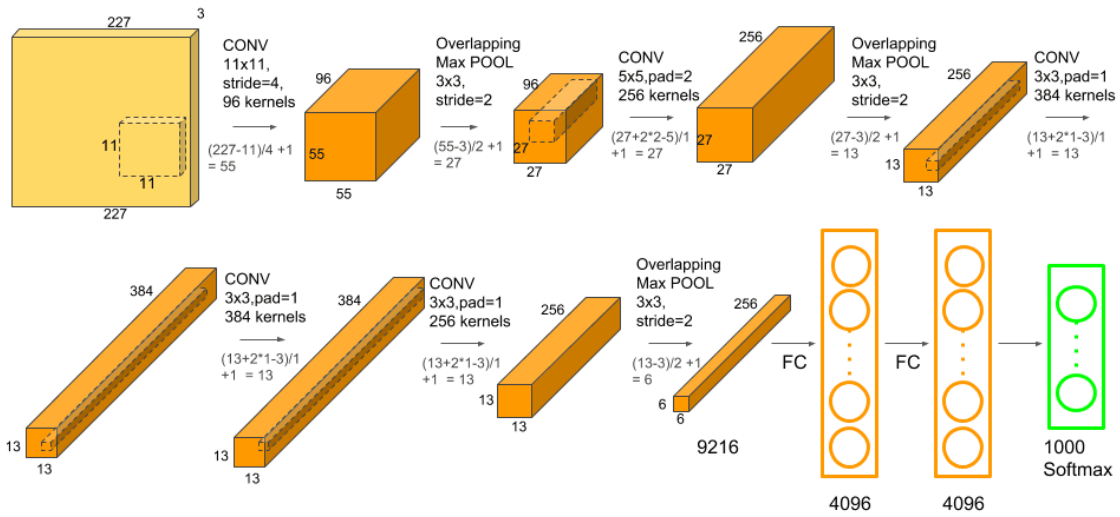


Figure 14: AlexNet architecture

As you can see in the figure 14, AlexNet contains eight layers; the first five were convolutional layers, some of them followed by max-pooling layers, and two fully connected layers, each with 4096 nodes. The final layer in the network is the softmax classifier. AlexNet uses the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid [8].

Implementation: We trained the AlexNet architecture on the Dogs vs. Cats dataset from Kaggle that has a total of 25,000 images with varying image resolutions. The goal of this dataset is to help train models to classify between dogs and cats.

Result: This experiment yields an accuracy of 92.97%.

1.4 VGGNet

In the previous sections we discussed LeNet and AlexNet, which played a significant role in the development of deep neural networks in the computer vision literature. VGGNet, (sometimes referred to as simply VGG), was first introduced in a 2014 paper entitled Very Deep Learning Convolutional Neural Networks for Large-Scale Image Recognition. The primary contribution behind this paper was to demonstrate that an architecture with very small (3x3) filters can be trained to increasingly higher depths (16-19 layers) and obtain state of the art classification on the challenging ImageNet classification contest.

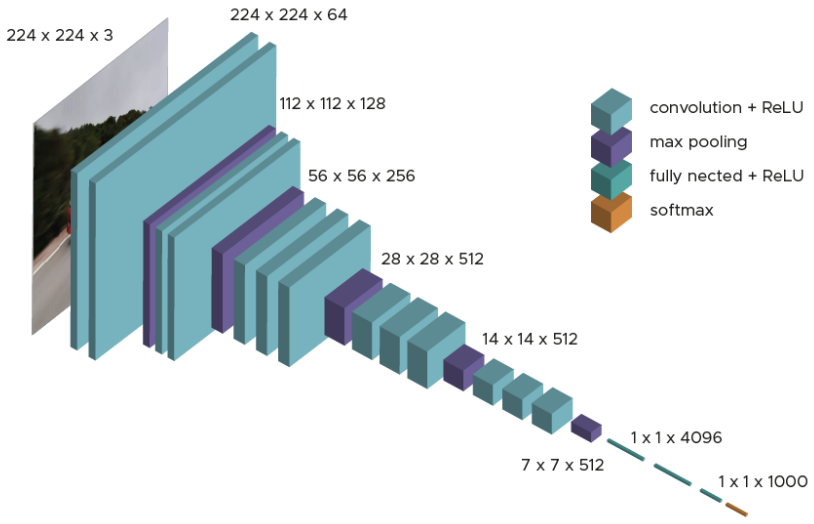


Figure 15: VGGNet architecture

The VGG family of convolutional neural networks can be characterized by two components:

- All convolutional layers in the network use only 3 x 3 filters.
- Stacking multiple convolution => ReLU layer sets before applying a pooling operation.

The use of small kernels is arguably what helps VGGNet generalize to classification problems. VGG16 significantly outperforms the previous generation of models in the ILSVRC-2012 and ILSVRC-2013 competitions, but on the other hand, VGGNet is painfully slow to train, and the network weights themselves are quite large. Due to its depth and number of fully connected nodes, VGG16 is over 533MB.

Implementation: We trained the a variant of the VGG architecture called MiniVGGNet on the CIFAR-10 dataset. CIFAR-10 contains 60,000 images of size 32 x 32 x 3, consisting of 10

classes, including: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks

Result: This experiment yields an accuracy of 83% with the use of batch normalization.

1.5 GoogleNet and inception module

Up to this point we have seen only sequential neural networks where the output of one network feeds directly into the next, but the GoogleNet architecture proposed by the 2014 paper Going Deeper with convolutions follows another approach. This network makes usage of a network in network or micro-architecture when constructing the overall macro-architecture, and this is what makes GoogleNet particular, in addition to the fact that the model architecture is very tiny compared to AlexNet and VGGNet, just about 28MB for the weights themselves [9].

Micro-architectures are small building blocks that are used inside the rest of the architecture, where the output of one layer can be split into a number of various paths and can be rejoined later. The contribution to the deep learning community of this paper, in this context is the inception module, which is a building block that fits into a convolutional neural network enabling it to learn features from convolutional layers with multiple filter sizes, turning the module into a multi-level feature extractor.

The idea behind the inception module, that concatenates the learned features from different kernel sizes, is that it can be hard to decide the size of the filter you need to learn a given convolutional layer. Should they be 5 x 5 or 3 x 3? What about learning local features using 1 x 1 filters? Instead, the proposed concatenation method is to let the model decide itself.

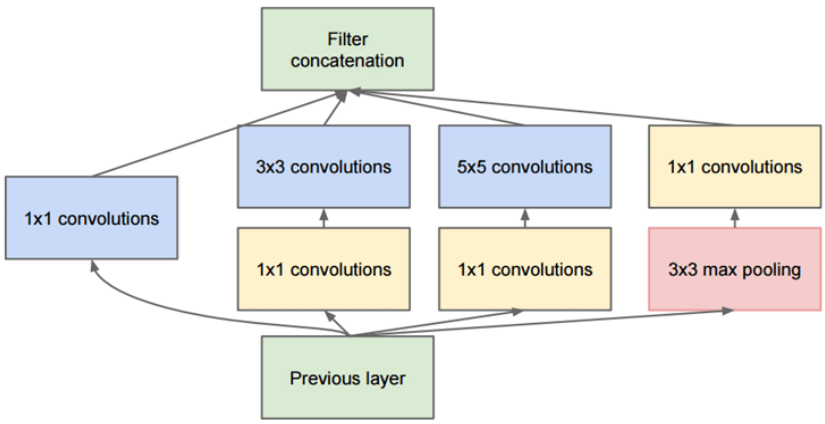


Figure 16: The inception module

GoogLeNet is a 22-layer deep convolutional neural network based on the Inception module developed by researchers at Google. The network was designed with computational efficiency and practicality in mind, so that inference can be run on individual devices including even those with limited computational resources, especially with low-memory footprint.

Figure 17 shows the GoogLeNet architecture alongside its embedded Inception module blocks.

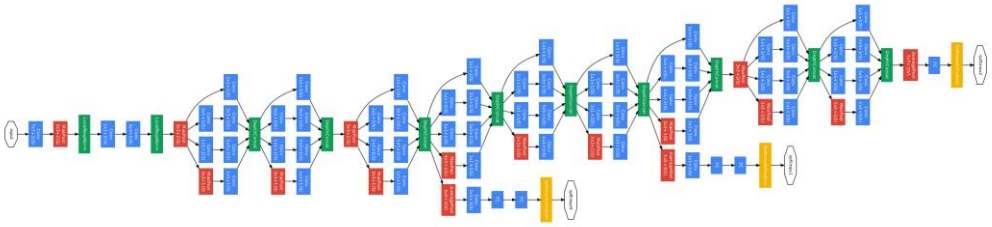


Figure 17: The GoogLeNet classification network

Implementation: We trained a variant of GoogleNet architecture called MiniGoogLeNet that is based on a Miniception module which is a simplified version of Inception, since the original one was designed so that it could be trained on the ImageNet dataset. But for smaller datasets, like in our case we trained it on the CIFAR-10 dataset that has smaller image spatial dimensions, only fewer network parameters are required.

Result: This experiment yields an accuracy of 90.81%.

1.6 ResNet and the residual module

The inception module has inspired many other micro-architectures after it, like the residual module which is the building block of the ResNet architecture. ResNet took first place in all three ImageNet Large Scale Visual Recognition Contest 2015 (classification, detection, and localization).

Just like GoogLeNet with the Inception module, ResNet uses the residual module to train convolutional neural networks to depths previously thought impossible. For example, in 2014, VGG16 and VGG19 architectures were considered very deep. However, with ResNet, networks based on it can have more than 1000 layers [10].

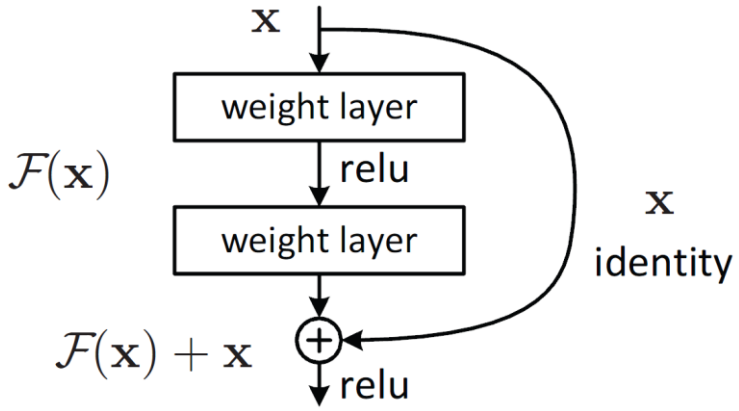


Figure 18: The residual module

These depths are only made possible by using "smarter" weight initialization algorithms along with the concept of identity mapping, shown in the figure 18, which is the process of taking the original input to the module and adding it to the output of a series of operations.

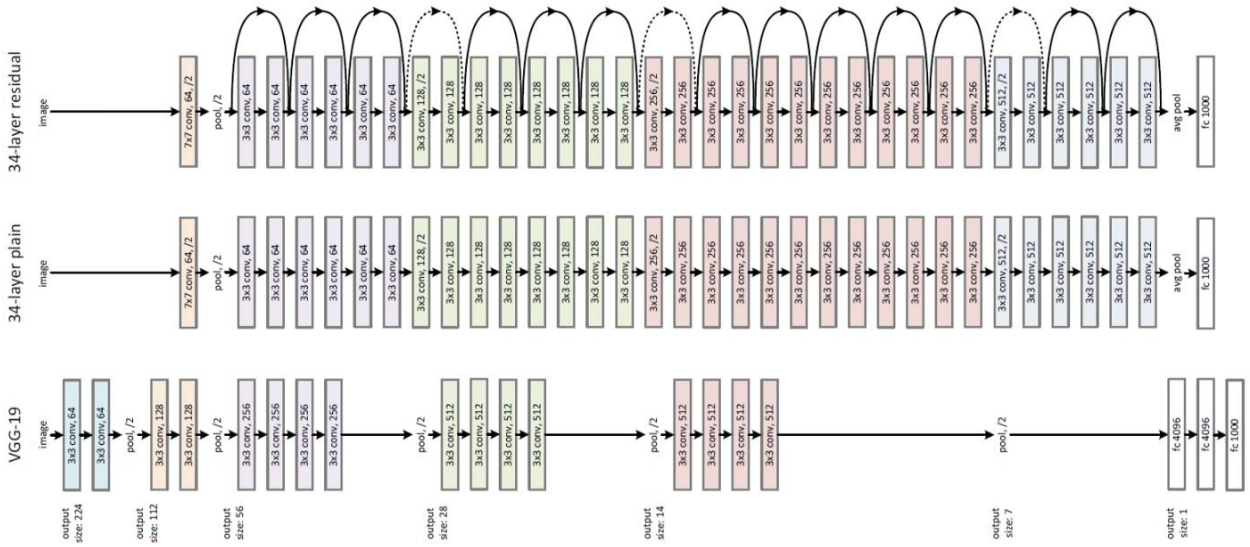


Figure 19: The ResNet classification network

Implementation: After writing the ResNet from scratch, using function that return the residual module, we trained on CIFAR-10 dataset, defined in the GoogLeNet section [11].

Result: This experiment yields an accuracy of 89.06%.

1.7 Implementations summary

Model	Dataset	Result (Accuracy)
LeNet	MNIST	98%
AlexNet	Kaggle Dogs vs. Cats	92.79%
MiniVGGNet	CIFAR-10	83%
MiniGoogLeNet	CIFAR-10	90.81%
ResNet	CIFAR-10	89.06%

I.2.2 Object Detection

2.1 Definition

Object detection is the field of computer vision that deals with the localization and classification of multiple objects contained in an image or video, and this is done simply by drawing bounding boxes around detected objects which allows us to locate them in each scene.

As we explained earlier, object detection is slightly more advanced than image classification, because it does not give us a unique tag associated with an image, but the classification and the localization of every single object in that image.

Just like in image classification, almost all object detectors were based on classical machine learning techniques, like the histogram of oriented gradients, scale-invariant feature transforms, etc. But deep learning-based techniques vastly outperform these traditional techniques.

Deep learning-based approaches use neural network architectures like YOLO (You Only Look Once), CenterNet, SSD (Single Shot Multibox Detector), Region proposals (R-CNN, Fast-RCNN, Faster-RCNN) for feature detection of the object, and then identification into labels.

2.2 Types of object detectors

Object detection is generally categorized into two types:

- Single-stage object detectors.
- Two-stage object detectors.

Two-stage detectors divide the object detection task into two stages: extracting the regions of interest that would normally contain the objects, then classify and regress them. Examples of object detection architectures that are two stage oriented, as shown in the figure 20, include: R-CNN, Fast-RCNN, Faster-RCNN, Mask-RCNN and others.

Popular two-step algorithms like Fast-RCNN and Faster-RCNN typically use a Region Proposal Network that proposes regions of interest that might contain objects. The output from the Region Proposal Network is then fed to a classifier that classifies the regions into classes.

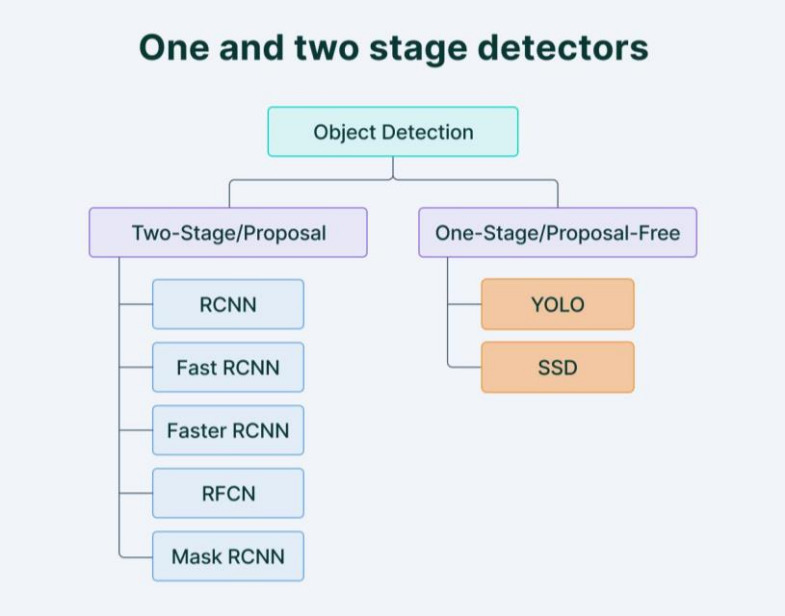


Figure 20: Types of detection models

While this gives accurate results in object detection with a high mean Average Precision (mAP), it results in multiple iterations taking place in the same image, thus slowing down the detection speed of the algorithm and preventing real-time detection

A single-stage detector removes the RoI extraction process and directly classifies and regresses the candidate anchor boxes. Examples of models that use this approach, as shown in the figure 20, include YOLO family with all its versions up to YOLOv5, SSD, CenterNet, etc.

Let's take a look at how both YOLO and Faster-RCNN work.

2.3 YOLO

Compared to the approach taken by object detection algorithms before YOLO, which repurpose classifiers to perform detection, YOLO proposes the use of an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once.

Following a fundamentally different approach to traditional object detection, YOLO achieves state-of-the-art results beating other real-time object detection algorithms by a large margin. Methods that use Region Proposal Networks thus end up performing multiple iterations for the same image, while YOLO gets away with a single iteration.

The problem is framed as a regression problem to spatially separated bounding boxes, predict bounding boxes and class probabilities directly from images in one evaluation, so that the whole detection pipeline is a single network that can be optimized end-to-end.

As you see in the image YOLO is very simple, a single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes [12].

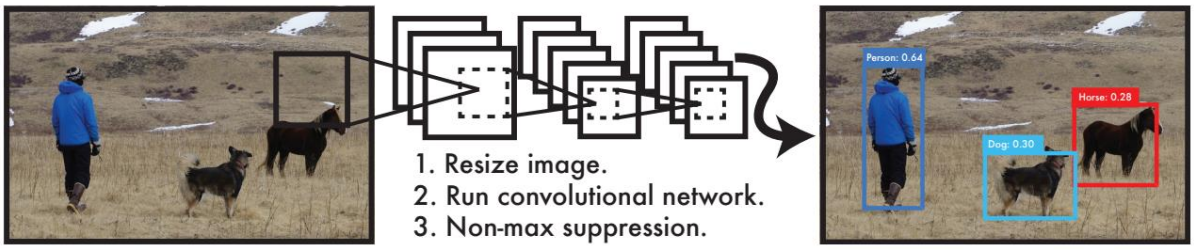


Figure 21: YOLO detection steps

Another advantage of YOLO is that it reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time, so it implicitly encodes contextual information about classes as well as their appearance. Faster R-CNN mistakes back-ground patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision. The network is extremely fast, 45 frames per second.

The system divides the input image into an S x S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each bounding box consists of 5 predictions: x, y, w, h, and confidence.

The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates. The network is inspired by the GoogLeNet model for image classification. It has 24 convolutional layers followed by 2 fully connected layers. The final output of the network is a 7 x 7 x 30 tensor of predictions.

We use the first 20 convolutional layers from the architecture followed by an average-pooling layer and a fully connected layer. Trained on ImageNet 1000-class competition dataset. They trained this network for approximately a week and achieved a single crop top-5 accuracy of 88% on the ImageNet 2012. They then convert the model to perform detection. As the article

shows, they add four convolutional layers and two fully connected layers with randomly initialized weights.

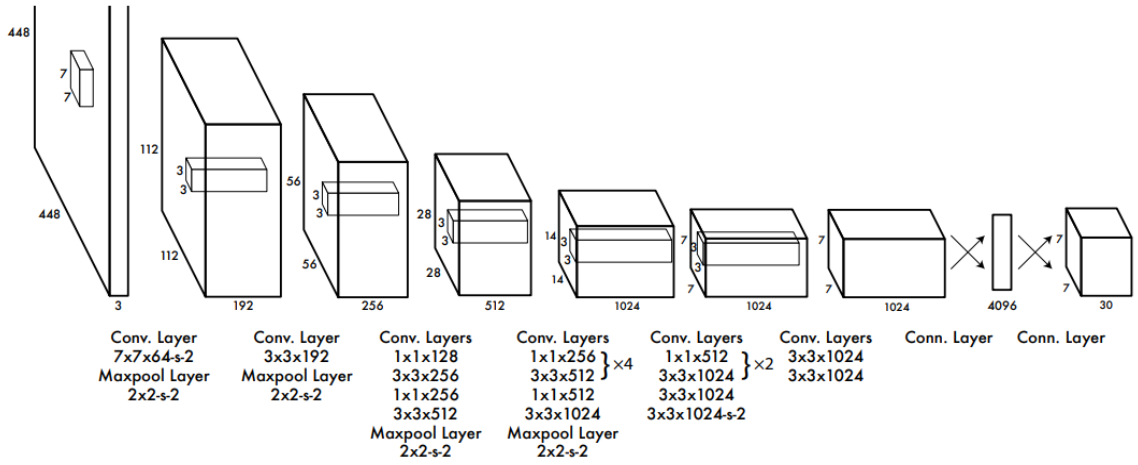


Figure 22: YOLO architecture

Detection often requires fine-grained visual information, so they increase the input resolution of the network from 224 x 224 to 448 x 448.

They normalize the bounding box width and height and the bounding box x and y coordinates by the image width and height so that they fall between 0 and 1. They use a linear activation function for the final layer and all other layers use the following leaky rectified linear activation as shown in the on the right.

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

They use the sum-squared errors because it is easy to optimize, however it does not align with the goal of maximizing average precision. It weights localization error equally with classification error which is not ideal. Also, in every image many grid cells do not contain any objects. This pushes to overpower the gradient from cells that do contain objects. This can lead to model instability, causing training to diverge early on.

To remedy this, they increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don't contain objects. We use two parameters, coord and noobjto accomplish this. We set coord=5 and noobj = 0.5.

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

Figure 23: The loss function of YOLO

Regarding the limitations of Yolo, it imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that the model can predict. And since the model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations.

2.4 Faster R-CNN

Faster R-CNN is an object detection model that improves on Fast R-CNN by utilizing a region proposal network (RPN) with the generated feature maps from the convolutional layer, to estimate a region-based object classification (ROI pooling).

Region proposals are candidates that might have objects within them. The number of these regions is usually in the several thousands, e.g., 2,000 or more. Examples of some algorithms that generate region proposals are Selective Search and EdgeBoxes [13].

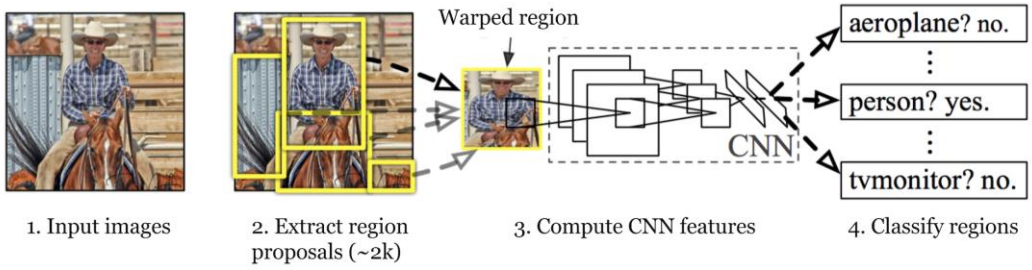


Figure 24: Faster R-CNN steps

In the R-CNN family of papers, the evolution between versions was usually in terms of computational efficiency (integrating the different training stages), reduction in test time, and improvement in performance (mAP) [14]. These networks usually consist of:

- A region proposal algorithm to generate "bounding boxes" or locations of possible objects in the image.
- A feature generation stage to obtain features of these objects, usually using a CNN
- A classification layer to predict which class this object belongs to.
- A regression layer to make the coordinates of the object bounding box more precise.

Below is an architectural diagram of Faster R-CNN:

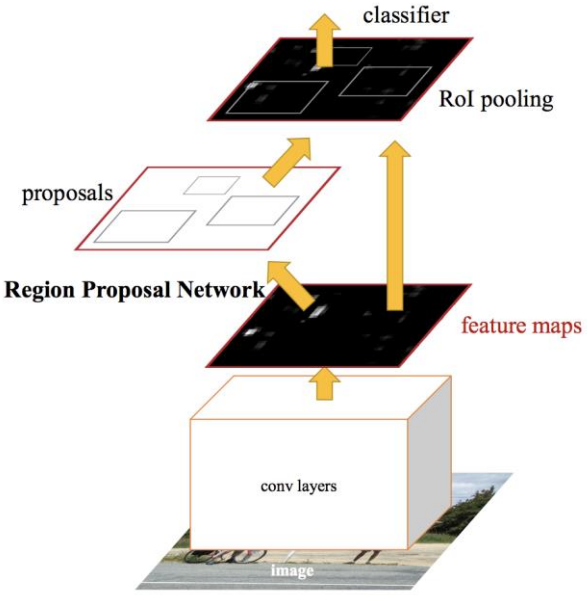


Figure 25: Faster R-CNN architecture

Approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections. These complex pipelines are slow and hard to optimize because each component must be trained separately.

2.5 Accuracy Metrics - mAP

Mean Average Precision(mAP) is a metric used to evaluate object detection models such as Fast R-CNN, YOLO, Mask R-CNN, etc. The mean of average precision (AP) values are calculated over recall values from 0 to 1 [15].

The mAP formula is based on the following sub metrics:

- Intersection over Union (IoU),
- Confusion Matrix,
- Recall,
- Precision

Intersection over Union (IoU)

Intersection over Union is a popular metric to measure localization accuracy and calculate localization errors in object detection models.

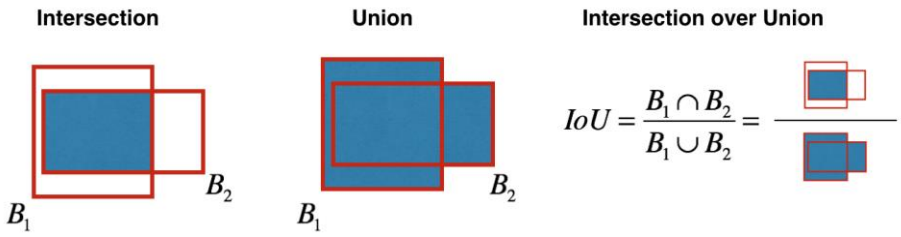


Figure 26: The intersection over union formula

The mAP is calculated by finding Average Precision (AP) for each class and then average over a number of classes.

Here is a summary of the steps to calculate the AP:

- Generate the prediction scores using the model.
- Convert the prediction scores to class labels.
- Calculate the confusion matrix—TP, FP, TN, FN.
- Calculate the precision and recall metrics.
- Calculate the area under the precision-recall curve.
- Measure the average precision.

Chapter 3: Object Detection Work and results

Chapter 3: Object Detection Work and results

As the deep learning model development pipeline suggests, the dataset collection has a crucial role in the development of our computer vision models, that's why our methodology was at first concentrated on the construction of the dataset that contains data that is relevant to our problem. And then the second step was to study and train the object detectors by training them on the constructed datasets.

So, the main axes of our work are:

- Building and customizing datasets that are suitable for training the models: Customized VisDrone and Ships dataset (built from scratch). - Dataset section.
- Studying, training, and evaluating state of the art object detection models and choosing the best fit among them for implementation. - Models section.

In addition to the relevance of the object classes included in the dataset and their significance to the context of the problem, the collection and construction of the dataset should take into consideration the next technical constraints, as it demonstrates similar aspects between drone captured images:

- Viewpoint variations: compared to surveillance cameras with fixed viewpoints, drone-equipped cameras monitor the objects in arbitrary viewpoints.
- Scale variations: drone-equipped cameras monitor the objects at different altitudes, resulting in large variations of scales of objects.
- Motion blur: videos are generally recorded by the drone equipped cameras in the moving process, bringing in considerable motion blurs of the recorded videos

I.3.1 Datasets

The context of our problem, which the surveillance of OCP industrial sites, imposes many constraints on the dataset construction process, the first constraint is that the images included in the dataset must be captured from of view of drones, as the surveillance of site will be conducted using cameras installed on drones. In addition to this constraint, the classes contained in the dataset should be classes that we aim to target in the surveillance process, which means that objects we detect should potentially be part of the environment of the sites. This means that we wouldn't be training on objects that are not likely to exist in one of OCP

sites, and on the other hand, if no dataset contains an object that crucial to the surveillance process, we should somehow integrate it in the dataset, while still respecting the first constraint, meaning that the added images of the new class should have a point of view of a drone or at least closed to it.

We chose to work on a dataset called VisDrone, for the reason that it perfectly satisfies the first constraint. As for the second the constraint, many of the classes of this dataset are to be targeted by our model, tough we have to customize it for there will be no anomaly regarding the other classes, which we will tackle by modifying all annotation files, either by removing unnecessary classes from them, or concatenate existing ones, or any other necessary operation for that matter.

I.3.1.1 Modified VisDrone

In this section, we will define the VisDrone dataset, and the preprocessing steps of its customization to our context, which is basically the class categories restructuring and the annotations processing.

Dataset Definition

The VisDrone2019 dataset is collected by the AISKYEYE team at the Lab of Machine Learning and Data Mining, Tianjin University, China. It's a large-scale benchmark with carefully annotated ground-truth for various important computer vision tasks to make vision meet drones [17].

Original VisDrone dataset has: 6471 in training, 548 images in validation and 1610 in testing. We moved all validation files to training. So that 7020 in training and 1610 in testing, about 18.6% for testing. The next two images are from the VisDrone dataset.

Figure 27: Image from Visdrone dataset

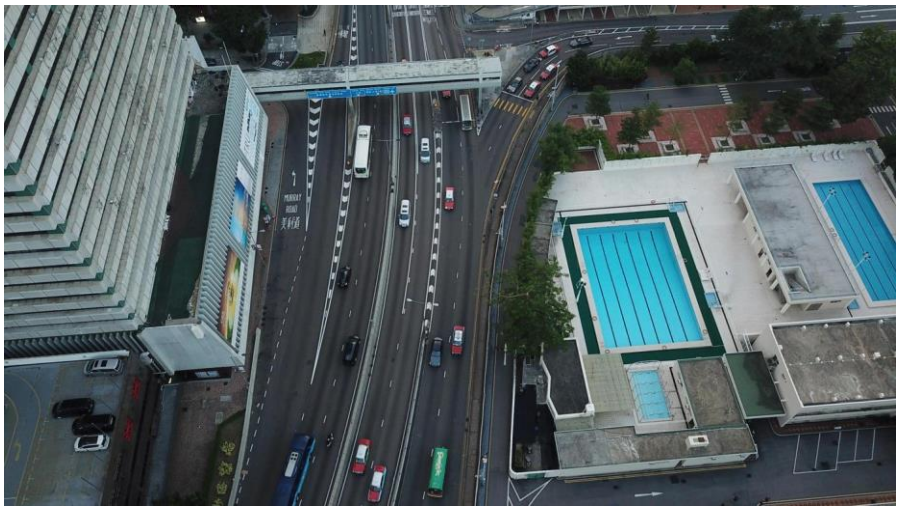


Figure 28: Image from VisDrone dataset

Class categories processing

The dataset contains 12 object categories represented in the annotation files as their indices from 0 to 11: ignored regions (0), pedestrian (1), people (2), bicycle (3), car (4), van (5), truck (6), tricycle (7), awning-tricycle (8), bus (9), motor (10), others (11) [18].

The classes that seem to be interesting for us are person, bicycle, car, van, truck, motor. So, our task is to write a Python script to make the transformation from the old version of annotations related to every image to the new framework containing our new set of classes, as shown in the figure 29.

It's worth mentioning that in the original format of VisDrone annotations, there is a distinction between the pedestrians and the people, considering that pedestrians are people walking on the

streets, while people are the ones who could be riding a bicycle for example and not walking, we decided to concatenate them all in one class with one index.

Regarding the rest of the original classes that are not important to us, they are just ignored by the script.

The figure 29 also shows how the indices are either ignored, or concatenated into one index, or just shifted down. The first case corresponds to when a class is not interesting to us, the second one corresponds to the case of concatenating persons and pedestrians into one class named person, or the index is just shifted when the case is none of the mentioned ones.

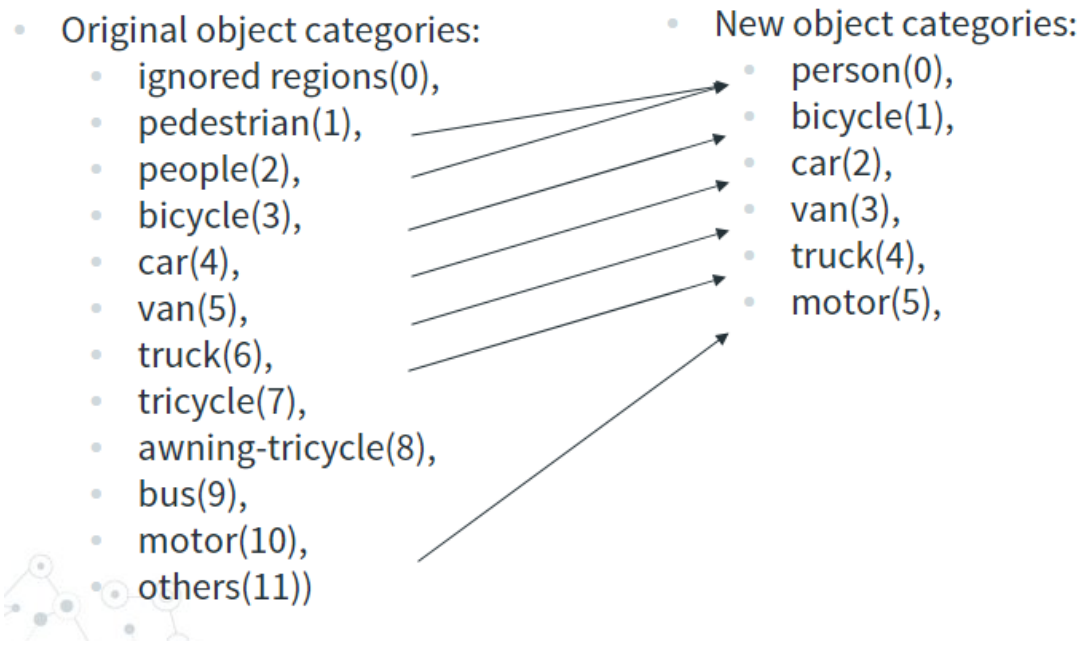


Figure 29: Annotation transformation of the VisDrone

The extraction of the indices in order to perform the processing rely on the annotation file structure discussed in the next section.

Annotations processing

Datasets generally consist of images and annotations of those images, containing information about objects and their bounding boxes in each image. Format and structure of the annotation files differ from dataset to dataset and depend also on the model we aim to train.

VisDrone has a very specific format and structure of the annotation files [9], it is a .txt file consisting of the structure represented in the figure 30:

```
<bbox_left>,<bbox_top>,<bbox_width>,<bbox_height>,<score>,<object_category>,<truncation>,<occlusion>
```

Figure 30: VisDrone annotation format

The following table 1 gives a description to every single element of this structure.

Table 1: VisDrone annotation format elements

Name	Description
<bbox_left>	The x coordinate of the top-left corner of the predicted bounding box
<bbox_top>	The y coordinate of the top-left corner of the predicted object bounding box
<bbox_width>	The width in pixels of the predicted object bounding box
<bbox_height>	The height in pixels of the predicted object bounding box
<score>	The score in the DETECTION file indicates the confidence of the predicted bounding box enclosing an object instance. The score in GROUNDTRUTH file is set to 1 or 0. 1 indicates the bounding box is considered in evaluation, while 0 indicates the bounding box will be ignored.
<object_category>	The object category indicates the type of annotated object, (i.e., ignored regions (0), pedestrian (1), people (2), bicycle (3), car (4), van (5), truck (6), tricycle (7), awning-tricycle (8), bus (9), motor (10), others (11))
<truncation>	The score in the DETECTION result file should be set to the constant -1. The score in the GROUNDTRUTH file indicates the degree of object parts appears outside a frame (i.e., no truncation = 0 (truncation ratio 0%), and partial truncation = 1 (truncation ratio 1% ~ 50%)).
<occlusion>	The score in the DETECTION file should be set to the constant -1. The score in the GROUNDTRUTH file indicates the fraction of objects being occluded (i.e., no occlusion = 0

	(Occlusion ratio 0%), partial occlusion = 1 (occlusion ratio 1% ~ 50%), and heavy occlusion = 2 (occlusion ratio 50% ~ 100%)).

An example of this structure is shown in the figure 31. Each line in the annotation text file, corresponds to a detected object in the image, with all its data elements defined in the table above.

The index of the object category is <object_category>, our script makes the extraction of it sequentially from each line in every file of the dataset and performs the necessary transformation mentioned in the previous section in order to get the new distribution of annotations.

```
0,1026,57,93,1,1,0,0
0,982,29,88,1,1,1,1
229,679,162,178,1,4,0,0
336,546,121,149,1,5,0,0
156,662,49,59,1,3,0,0
388,46,48,42,1,5,0,1
230,240,148,67,1,4,0,2
575,357,100,79,1,4,0,0
146,292,164,91,1,4,0,2
295,178,152,61,1,4,0,2
```

Figure 31: Example of VisDrone annotation format

After getting the output of our script, and verifying its correctness, another transformation should be performed from the resulting annotation structure to the model's annotation structures, for the reason that every object detection model has an architecture designed for a specific structure.

There are two main annotation formats for the object detection models, the YOLO .txt format and the PASCAL VOC .xml format, and each has its own structure and included information regarding the bounding boxes.

YOLO models have a text format for their annotations, which we will use to train our YOLO variants models. Each text file has the same name as the image, and consists of lines of the next format:

<object-class> <x> <y> <width> <height>

Where:

- <object-class> the class index, an integer from 0 to (number of classes -1)
- <x> <y> are the center of the rectangle.
- <width> <height> are the height and the width of the image
- <x> <y> <width> <height> are float values resulting from normalization from pixels to the range 0.0 to 1.0, using next formulas:
    - <x> = <absolute_x> / <image_width>
    - <y> = <absolute_y> / <image_height>
    - <height> = <absolute_height> / <image_height>
    - <width> = <absolute_width> / <image_width>

So for example for an image named image01.jpg, the annotation file would be a .txt file named image01.txt containing:

```
1 0.716797 0.395833 0.216406 0.147222
0 0.687109 0.379167 0.255469 0.158333
1 0.420312 0.395833 0.140625 0.166667
```

Figure 32: YOLO annotation format example

For the PASCAL VOC annotation files, they are different from the YOLO format and structure, for they are .xml format and have a structure shown in the figure 33.

```
<annotation>
  <folder>Train</folder>
  <filename>01.png</filename>
  <path>/path/Train/01.png</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>224</width>
    <height>224</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>36</name>
    <pose>Frontal</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <occluded>0</occluded>
    <bndbox>
      <xmin>90</xmin>
      <xmax>190</xmax>
      <ymin>54</ymin>
      <ymax>70</ymax>
    </bndbox>
  </object>
</annotation>
```

Figure 33: PASCAL VOC annotation format

1.3.1.2 Ships dataset

In this section, we present the ships dataset, its collection, filtering, labeling steps, and tools used in every step.

Presentation of the dataset

The ships are a very essential element to the construction of models, as they are an object necessary to detect in our industrial zones, but VisDrone does not have this category, and other existing dataset on the internet are either containing non-industrial ships or ships from a different point of view than what we are interested in, for example datasets for ships in satellite images. But since this is not compatible with the two constraints discussed earlier.

Our goal in this section is to construct a dataset from scratch containing images and annotations of industrial ships. To do this, we seek to collect images of ships from the internet, preprocess and clean them, and label them after that.

Industrial ships dataset is a dataset we collected from google images, using a tool for collecting images from web pages called Imageye, which is a Google Chrome extension. For adding

diversity to the dataset, we made searches on Google Image using multiple keywords and multiple languages (English and French).

Four categories of industrial ships were included in the dataset to add some diversification that would help the model to generalize afterwards. The four categories are: Container ships, Bulk ships, Tanker ships and reefer ships. Next four figures show those types:

Figure 34: Container ship

Figure 35: Bulk ship

Figure 36: Tanker ship

Figure 37: Reefer ship

Preprocessing steps

Downloading a large set of images to construct a dataset needs to be a very careful process so as to include as less noise as possible, because noisy data will certainly affect our model's accuracy after.

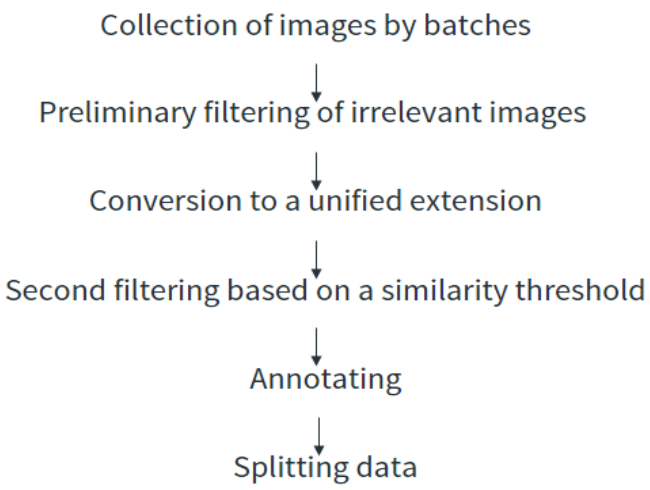


Figure 38: Ships dataset preprocessing steps

The design of the collection and preprocessing aims to maximize relevant and diverse data, to help the model generalize, and on the other hand, to minimize noise in it. The collection and preprocessing steps are shown in the figure 38.

- Collection of images by batches:

  In the first step, we search for multiple keywords relevant to industrial ships, like specific types of ships (container ships, bulk ships, tanker ships, reefer ships ...), or in general (industrial ships ...), and in different languages (English and French). The Imageye Chrome extension gives us the ability to download all displayed images on a page of Google Images, as the figure 39 shows, we consider the downloaded set of images from every resulting page as a batch.

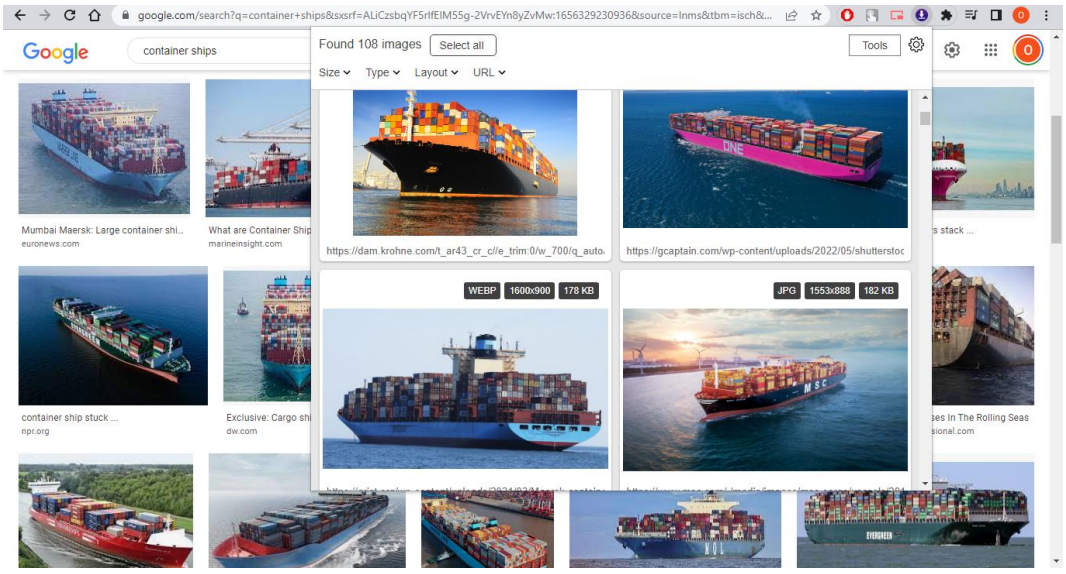


Figure 39: Collecting and filtering images from Google Images using Imageye extension

- Preliminary filtering of irrelevant images

  The same extension allows us to do the preliminary filtering, in order to eliminate irrelevant data that would result of our search, like:

  - Infographics about ships, animations ...
  - Images with different viewpoints (Satellite images, images from within the ship...).
  - Large titles or much text
  - …

  Not eliminating those anomalies would result in biasing the model while training.

- Conversion to a unified extension

  The dataset should contain images of the same type, that is supported by the training pipeline dataset loading scripts, that is why we convert all images to a unified extension: jpg. Raw images are of different extensions: jpg, png, webp, tiff…etc.

- Second filtering based on a similarity threshold

  For the reason that we download the image sets by batches from many web pages, it is possible that we download an image many times and repeating this throughout the dataset might increase the possibility of the model's overfitting when training.

After this step, the number of images in our dataset is: 1228 images.

- Annotating

  The labeling of images is done manually to every single image of the dataset using a tool called LabelImg, which a software that can be installed on Windows, and by drawing bounding boxes around every object in every image in the folder containing our dataset, that software automatically generates the annotation files for both formats, YOLO .txt format or PASCAL VOC .xml format.

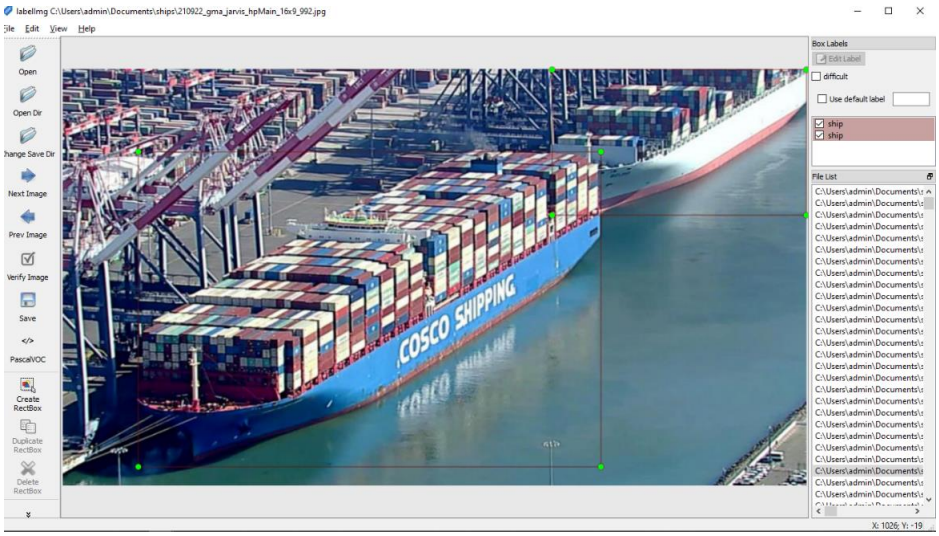


Figure 40: Labeling images using LabelImg

The figure 40 shows the interface of LabelImg as well as an example of an image containing two ships.

- Splitting data

  The final step is splitting the collected data to a training and a validation set. This is done by simply moving about 20% of images chosen randomly to another separate folder.

I.3.2 Models

In this part, we will discuss the pipelines used to train and evaluate the models used. We will also show the hardware configurations used for those process, and the final results of their evaluation.

I.3.2.1 Training Pipelines

There are two main training pipelines that we are going to adopt in the process of training and evaluation. And this is due to the different environments and frameworks necessary to the execution of every model, the two pipeline are:

- YOLOv5 pipeline:

  This is the pipeline we used to train and evaluate YOLOv5, with all its variants: Nano, Small, Medium, Large, Xlarge [19].

- TensorFlow Object Detection API pipeline (TFOD):

  This is the pipeline we used to configure the environment that is suitable for training and evaluating the following models: Faster R-CNN, SSD, EffecientDet and CenterNet.

YOLOv5 Pipeline

The YOLOv5 pipeline needs an environment with specific requirements, starting from Python 3.8 and Pytorch 1.7. The training needs our dataset to be organized following a specific directory structure and a configuration file with a .yaml extension, which shows the model functions paths to both training and evaluation data, as well as some other configurations.
YOLOv5 repository on GitHub already has a configuration file that corresponds to VisDrone dataset, we reconfigure it to become compatible with our dataset, by specifying the root path to the dataset directory, and both relative training and evaluation sets, as well as the number of classes end their names, as the figure 41 shows.

When the environment, the dataset and the configuration file are ready we lunch the training using the next parameter values:

- Input size: 1024px, because objects in the VisDrone are taken from a viewpoint of drones, objects tend to look small, so it would be better for the input size to be as large as possible.

- Batch size: 32

- Number of epochs: 50

- Model:
    - yolov5s.pt, the small version of the architecture in the first experience.
    - yolov5m.pt, the medium version of the architecture in the second experience.

```
# YOLOv5  by Ultralytics, GPL-3.0 license
# VisDrone2019-DET dataset https://github.com/VisDrone/VisDrone-Dataset by Tianjin University
# Example usage: python train.py --data VisDrone.yaml
# parent
# ├── yolov5
# └── datasets
#     └── VisDrone  ← downloads here (2.3 GB)


# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: /content/visdrone  # dataset root dir
train: VisDrone2019-DET-train/images
val: VisDrone2019-DET-test/images

# Classes
nc: 6  # number of classes
names: ['person', 'bicycle', 'car', 'van', 'truck', 'motor']
```

Figure 41: Configuration file for the YOLO model

The figure 42 shows the execution result of the training code cell:

```
!python train.py --img 1024 --cfg yolov5s.yaml --hyp hyp.scratch-high.yaml --batch 32 --epochs 50 --data VisDrone.yaml --weights yolov5s.pt
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 26/26 [01:03<00:00,  2.42s/it]
                 all       1610      71033      0.479      0.412      0.398      0.217

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      21/49     15.4G   0.04868    0.1241  0.009691      1222      1024: 100% 220/220 [15:14<00:00,  4.16s/it]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:   8% 2/26 [00:04<00:57,  2.40s/it]WARNING: NMS time
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 26/26 [01:03<00:00,  2.43s/it]
                   all       1610      70905      0.504      0.402      0.398      0.217

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      22/49     15.4G   0.04799    0.1233  0.009475       718      1024: 100% 220/220 [15:05<00:00,  4.12s/it]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:   8% 2/26 [00:04<00:55,  2.29s/it]WARNING: NMS time
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 26/26 [01:04<00:00,  2.47s/it]
                   all       1610      70905      0.488      0.415      0.402       0.22

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      23/49     15.4G   0.04834    0.1238  0.009455       874      1024: 100% 220/220 [15:08<00:00,  4.13s/it]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 26/26 [01:05<00:00,  2.51s/it]
                   all       1610      71033      0.486      0.417      0.404      0.221

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      24/49     15.4G   0.04817    0.1245   0.00949       987      1024: 100% 220/220 [15:17<00:00,  4.17s/it]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 26/26 [01:04<00:00,  2.49s/it]
                   all       1610      71033      0.478      0.414      0.397      0.218

      Epoch   gpu_mem       box       obj       cls    labels  img_size
      25/49     15.4G   0.04799    0.1231  0.009409       861      1024: 100% 220/220 [15:18<00:00,  4.18s/it]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 26/26 [01:02<00:00,  2.42s/it]
                   all       1610      71033      0.513      0.405       0.41      0.225
```

Figure 42: Training output

The happening improvement of many parameters during training is shown in the previous figure, we can see that the mean average precision hit 41% in the epoch number 25/49, a detailed summary of results is shown in the results section.

Code and notebooks can be found in the following link: https://drive.google.com/drive/folders/1QF9Gq9TXFOOIFENz5cvMxZ3anZgCVLrU?usp=sharing

TensorFlow Object Detection API pipeline

This is the pipeline used to train: Faster R-CNN, SSD, EfficientDet and CenteNet, and might be used to train even other models that are available in the TensorFlow 2 Object Detection Model Zoo. The figure 43 shows a part of the Model Zoo.

The model zoo [20] provides a collection of detection models pre-trained on the COCO 2017 dataset. We have the possibility to choose among many input sizes and backbone architectures, which represent the classification architectures discussed in the state-of-the-art section

SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	35.6	Boxes
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	39.5	Boxes
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35.4	Boxes
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	39.6	Boxes
Faster R-CNN ResNet50 V1 640x640	53	29.3	Boxes
Faster R-CNN ResNet50 V1 1024x1024	65	31.0	Boxes
Faster R-CNN ResNet50 V1 800x1333	65	31.6	Boxes
Faster R-CNN ResNet101 V1 640x640	55	31.8	Boxes
Faster R-CNN ResNet101 V1 1024x1024	72	37.1	Boxes
Faster R-CNN ResNet101 V1 800x1333	77	36.6	Boxes
Faster R-CNN ResNet152 V1 640x640	64	32.4	Boxes
Faster R-CNN ResNet152 V1 1024x1024	85	37.6	Boxes
Faster R-CNN ResNet152 V1 800x1333	101	37.4	Boxes
Faster R-CNN Inception ResNet V2 640x640	206	37.7	Boxes

Figure 43: The model zoo pre-trained models

.

We download the pre-trained model and save it under a defined folder organization as the figure 44 shows, the example for two downloaded pre-trained models.

The configuration of the training pipeline is done using the pipeline.config file, where we configure the following parameters:

- Number of classes: num_classes.
- Batch size: batch_size.
- Path to the pre-trained checkpoints: fine_tune_checkpoint.
- The type of fine tuning: fine_tune_checkpoint_type, which is in our case the detection.
- Training on a TPU or not parameter: use_bfloat16.
- Path to the label map: label_map_path.
- Path to TF_record training file: tf_record_input_reader.input_path.
- Path to TF_record evaluation file: eval_input_reader.input_path.

A part of this pipeline.config file is displayed in the figure 45.

```
training_demo/
├─ ...
├─ pre-trained-models/
|  ├─ efficientdet_d1_coco17_tpu-32/
|  |  ├─ checkpoint/
|  |  ├─ saved_model/
|  |  └─ pipeline.config
|  └─ ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/
|     ├─ checkpoint/
|     ├─ saved_model/
|     └─ pipeline.config
└─ ...
```

Figure 44: Folder structure

The label map is a file with extension .pbtxt, which helps map from the name of every class to its corresponding index.

```
  fine_tune_checkpoint: "pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckp
  num_steps: 25000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection" # Set this to "detection" since we want to be training the
  use_bfloat16: false # Set this to false if you are not training on a TPU
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "annotations/label_map.pbtxt" # Path to label map file
  tf_record_input_reader {
    input_path: "annotations/train.record" # Path to training TFRecord file
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "annotations/label_map.pbtxt" # Path to label map file
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/test.record" # Path to testing TFRecord
  }
}
```

Figure 45: Part of pipeline.config file

The images and annotations should be transformed to a special format called TFrecord, which is a simple format for storing a sequence of binary records.

Training and evaluation processes should be executed separately in parallel, as this is the approach of the TensorFlow 2 Object Detection API, this helps to not stop the training process after the end of every epoch, but working on Google Colab makes this challenging, for that we can't execute more than one cell at once in one session.

The solution to this problem is to work on with more than one session and execute on cell in every session, while saving the new checkpoints in Google Drive from the session of training and retrieving them from there in the session of evaluation. We can work with a third session as well for the monitoring of both training and evaluation results at the same time using TensorBoard.

The next figure shows the output of the process of training of the Faster R-CNN model with a ResNet 101 backbone:

Faster RCNN ResNet 101

```
!python model_main_tf2.py --model_dir=/content/gdrive/MyDrive/visdrone_faster_rcnn_resnet101_v3 --pipeline_config_path=models/my_faster_rcnn/pipeline.config
 'Loss/BoxClassifierLoss/localization_loss': 0.4166441,
 'Loss/RPNLoss/localization_loss': 0.3311504,
 'Loss/RPNLoss/objectness_loss': 0.018485537,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 1.0540736,
 'learning_rate': 0.020448763}
I0513 15:58:10.278759 140550645852032 model_lib_v2.py:708] {'Loss/BoxClassifierLoss/classification_loss': 0.2877935,
 'Loss/BoxClassifierLoss/localization_loss': 0.4166441,
 'Loss/RPNLoss/localization_loss': 0.3311504,
 'Loss/RPNLoss/objectness_loss': 0.018485537,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 1.0540736,
 'learning_rate': 0.020448763}
INFO:tensorflow:Step 50400 per-step time 0.438s
I0513 15:58:54.090582 140550645852032 model_lib_v2.py:707] Step 50400 per-step time 0.438s
INFO:tensorflow:{'Loss/BoxClassifierLoss/classification_loss': 0.35627282,
 'Loss/BoxClassifierLoss/localization_loss': 0.27397117,
 'Loss/RPNLoss/localization_loss': 0.30365443,
 'Loss/RPNLoss/objectness_loss': 0.06536381,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.9992623,
 'learning_rate': 0.02038466}
I0513 15:58:54.090908 140550645852032 model_lib_v2.py:708] {'Loss/BoxClassifierLoss/classification_loss': 0.35627282,
 'Loss/BoxClassifierLoss/localization_loss': 0.27397117,
 'Loss/RPNLoss/localization_loss': 0.30365443,
 'Loss/RPNLoss/objectness_loss': 0.06536381,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.9992623,
```

Figure 46: TFOD training output

The next figure shows the output of the process of evaluation:

Faster RCNN ResNet 101

```
[ ] !python model_main_tf2.py --model_dir=/content/gdrive/MyDrive/visdrone_faster_rcnn_resnet101_v4 --pipeline_config_path=models/my_faster_rcnn/pipeline.config --checkpoint_dir=/conter
DONE (t=5.19s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.120
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.262
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.096
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.064
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.191
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.259
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.045
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.179
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.272
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.191
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.386
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.400
INFO:tensorflow:Eval metrics at step 46000
I0514 18:43:01.460887 140353232164736 model_lib_v2.py:1015] Eval metrics at step 46000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP: 0.120206
I0514 18:43:01.495289 140353232164736 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP: 0.120206
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.50IOU: 0.262167
I0514 18:43:01.497112 140353232164736 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP@.50IOU: 0.262167
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.75IOU: 0.096193
I0514 18:43:01.498520 140353232164736 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP@.75IOU: 0.096193
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (small): 0.064500
I0514 18:43:01.499831 140353232164736 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP (small): 0.064500
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (medium): 0.191127
I0514 18:43:01.501249 140353232164736 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP (medium): 0.191127
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (large): 0.259249
I0514 18:43:01.502718 140353232164736 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP (large): 0.259249
```

Figure 47: TFOD evaluation output

Code and notebooks can be found in the following link:

https://drive.google.com/drive/folders/1QF9Gq9TXFOOIFENz5cvMxZ3anZgCVLrU?usp=sharing

I.3.2.2 Hardware configuration

For the machine specifications, we worked on Google Colab's environment, that provided us with the next hardware specifications:

- GPU: Tesla P100 with 16Gb of memory.

- RAM: memory of 12Gb, possible 25Gb environment.
- Live Disk of 120Gb to 188Gb.

The storage of checkpoints for the task of parallel training and evaluation is done using Google Drive, as well as the storage of all related data:

- Maximum capacity of 100Gb.

As discussed earlier, the training and evaluation procedures follow a parallel structure in the TensorFlow Object Detection API pipeline, while it is done sequentially in the YOLO pipeline, as the figure 48 shows, the sequential way is displayed on the left, while the parallel one is displayed on the right:

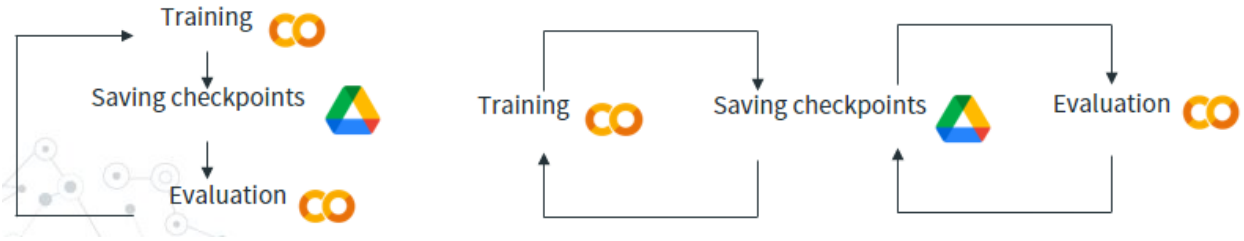


Figure 48: On the left: Sequential training and evaluation procedure, on the right: parallel procedure

I.3.2.3 Results and discussion

Table 2: Training results

Model	BackBone	Input size	Batch size	Epochs / Steps	mAP@.5	mAP@.5 :.95	Speed (ms) COCO	mAP@.5 On COCO (default batch size)	Model Size (Mb)
YOLOv5	s* 270	1024	32	83e	43.2	24.1	6.4	37.2	14
YOLOv5	m**369	1024	32	95e	43.2	24.1	8.2	45.2	40.51
Faster R-CNN	ResNet50	1024	4	>100000s	22.0	9.5	65	31	4.35
Faster R-CNN	ResNet101	1024	2	>100000s	27.9	12.5	72	37.5	7.36
SSD	MobileNet	640	8	23000	22.3	11.9	39	28.2	6.66
EffecientDet	D1	640	2	21000	14.3	7.1	54	38.4	22.54
CenterNet	-	-	-	-	-	-	-	-	

Table 2 contains many parameters and indicators:

- **Backbone:** the classification architecture used inside the detection model; the number indicates the number of layers.
- **The input size:** the input size of the image in pixels.
- **Batch size:** the number of samples per batch.
- **Epochs / steps:** the number of epochs in the case of YOLO, and steps in the case of TFOD API.
- **mAP:** Mean Average Precision for a level of intersection over union.
- **Speed:** the number of seconds to process one frame.

For Faster R-CNN, SSD and EffecientDet, we couldn't use larger batch size with the available hardware and using the dataset size. And this affected the mAP of those models, which means that with better hardware configuration, we would certainly converge to better results.

The batch size has also a link to the input size of the model, so that when the input size is larger, the batch size needs to be smaller to allow for more space in the graphical card, otherwise the process of training would crash.

For example, Faster R-CNN, with a ResNet50, can use up to only 2 samples in the batch size, while in the training pipeline.config we find that the default batch size is at lease 24. For some models it can be even 128 sample in the batch size. Having a larger batch size would help the model converge to better mean average precision, which means that for many of the TensorFlow Object Detection API, the mAP can get improved much more using more powerful hardware.

Finally, we conclude that in terms of accuracy, speed, and size of the model, YOLOv5s is the best fit for our constraints.

After these results, we decided to train only YOLOv5s on the ships dataset and this yield to a mAP value of 45%

In the following, we show the inference output images of the trained models on videos from the test set:

Figure 49: Output of four models on the first video

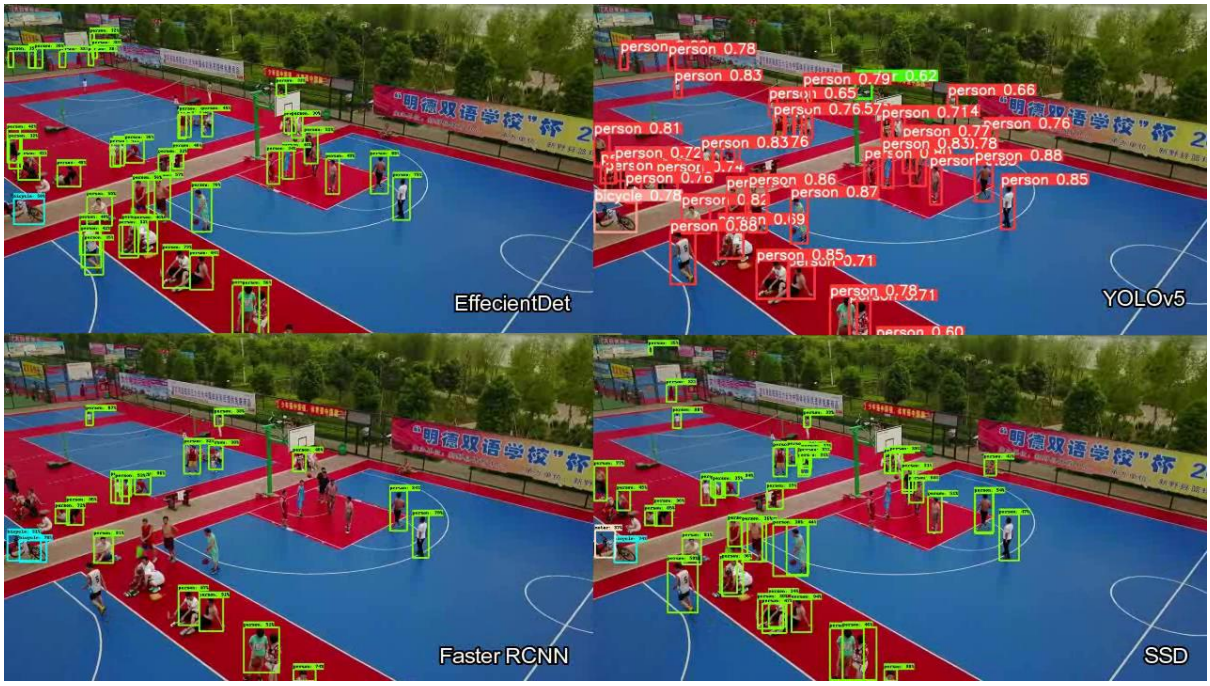


Figure 50: Output of four models on the second video

Figure 51: Output of four models on the third video

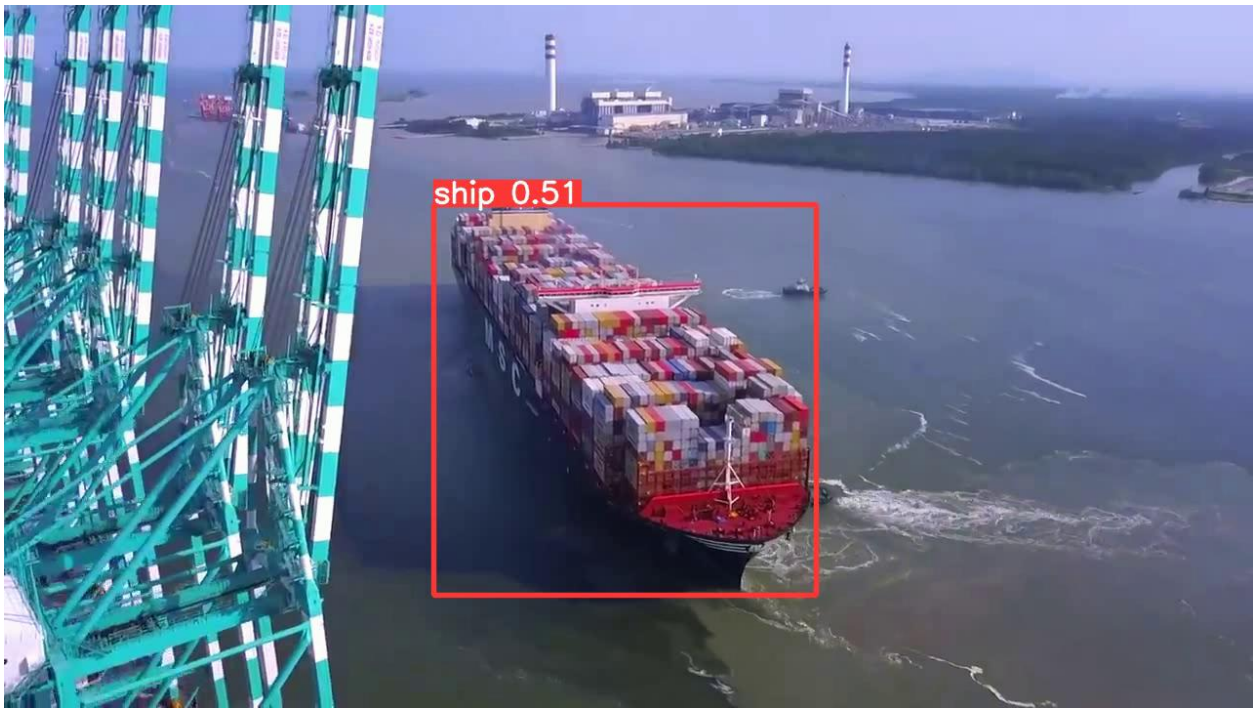


Figure 52: Output of YOLOv5s on a video containing ships

Conclusion

It turns out after this work that the deployment of an object detection model with high performance metrics should consider the YOLOv5s model, trained on our personalized and collected datasets, and this proves how powerful is the YOLOv5 architecture and design in the context of object detection in general.

YOLOv5 has proven not to only have the best accuracy metrics, but also to be the fastest among all of the other detectors. Furthermore, being the best in the two constructed and personalized datasets, proves the generalizability and ability to adapt with varying contexts and objects of the model, which adds a huge preferability to it by the computer vision community, and show again the point of having to migrate from traditional techniques to the deep neural networks.

Building datasets was a crucial step in the overall methodology, taking into consideration the dependency of the final results on it, while knowing that the used models have already proven their performance on other state of the art datasets, which puts us facing another challenge of collecting and annotating data correctly so as not to bias the model. But judging by the results, it is obvious that those processes were done to some high extent of correctness, since the models were able to learn and extract clear patterns from it.

General conclusion and perspectives

The work on this computer vision part up to this point opens the door to many improvement possibilities, and further integration of more advanced features, while planning for the designing of a vast and robust visual system having industrial supervision as a goal.

Object detection can be considered as a building block of industrial vision systems, especially in the context of supervision and surveillance, and adding more features would take these processes to another level of information collection and decision-making assistance. Among those features we can talk about object tracking, human action recognition, frame segmentation…etc.

The basics of computer vision and the deep learning tools discussed in this part would certainly be the start of every element of perspectives that can be further aimed for, because of the heavy dependency on these building blocks as a solid ground of very vision application. This fact was the motivation for investing time in studying and practicing basic concepts and treatments, since any other development would follow the same basis, only the approach and the methodology that would depend on the type of the problem.

On another hand, this work on constructing datasets and training detection models can be further enhanced by collecting more data and by more cleansing of the existing one, we can also introduce more powerful hardware configuration or just by making the pipeline more compatible with available one. Furthermore, we can also enhance the process of evaluation by integration advanced statistical tests to measure more accurately the related performance metrics.

Part 2: Navigation

The second part of the project concerns the autonomous navigation of UAVs for surveillance. The solution currently envisaged is to use commercial drones, most of which contain navigation and obstacle avoidance modules. In this case, a solution would work on the existing modules and adapt them according to the possible surveillance scenarios. At the research level, the second objective of the internship is to study autonomous navigation methods based on Deep Reinforcement Learning, and to test them in simulation.

The first chapter explains what navigation using coverage path planning is, and what are the requirements that a robot must have in order to perform a coverage operation, and since we aim to study autonomous navigation methods based on Deep Reinforcement Learning, this part also has a detailed proof that the coverage path planning is an NP-hard problem and using heuristics or Reinforcement Learning is probably the best way to handle it.

The second chapter presents a way of modelling the problem mathematically and introducing an optimal algorithm for online coverage path planning that has a logarithmic complexity. Other smarter algorithms making use of Deep Reinforcement Learning are to be studied and practiced in in the rest period of the internship.

Chapter 1: Online Coverage Path Planning

In this chapter, we will define the problem of online coverage path planning, as well the NP-hardness of problems and provide a proof of the NP-hardness of our coverage path planning problem.

II.1.1 Definition

CPP is the task of determining a trajectory that enable a moving agent to travel over every area or volume of an area of interest, while avoiding every collapse with obstacles. This is an essential problem in many applications of robotics, such as lawn mowers, vacuum cleaning robots, autonomous underwater vehicles, and many other interesting applications.

The requirements that a robot must meet to perform a coverage operation, as defined in one of the earliest works on CPP [21], are as follows:

- Robot must move through all the points in the target area covering it completely.
- Robot must fill the region without overlapping paths.
- Continuous and sequential operation without any repetition of paths is required.
- Robot must avoid all obstacles.
- Simple motion trajectories (straight lines or circles) should be used (for simplicity in control).
- An optimal path is desired under available conditions

Sometimes, not all these criteria can possibly be satisfied at once, especially in complex environments. Therefore, a priority consideration among them is required.

This problem is NP-hard, as we will demonstrate in the next section, and this makes the computational time required to solve increase drastically when the dimensions of the problem increase.

Coverage path planning algorithms can be classified as either off-line or on-line. Off-line algorithms rely only on stationary information, and the environment is assumed to be known in advance. However, assuming full prior knowledge of the environment might be unrealistic in many scenarios. On the other hand, on-line algorithms do not assume full prior knowledge of the environment to be covered and utilize real-time sensor measurements to sweep the target space. Thus, these later algorithms are also called sensor-based coverage algorithms.

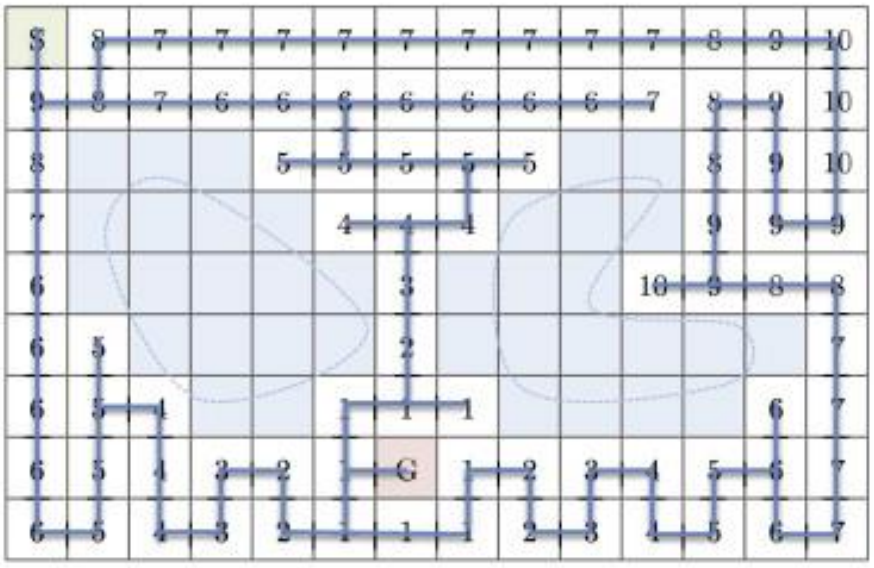


Figure 53: Example of grid based CPP

Surveys about coverage path planning show that there has been many methods trying to find an optimal solution to its constraints such us:

- Classical Exact Cellular Decomposition Methods
- Morse-based Cellular Decomposition
- Landmark-based Topological Coverage
- Grid-based Methods (see figure 53)
- Using Deep Neural Networks and Reinforcement Learning.

II.1.2 NP-hardness of CPP problem

In this part, we introduce the concept of computational complexity and its significance to the algorithmic problem solving, and then we provide a proof of the NP-hardness of the coverage path planning problem.

II.1.2.1 Computational complexity:

Computational complexity is a sub-field of computer science that studies the amount of computing resources that a given algorithm consumes when it runs trying to solve a problem. Particular focus is on time and memory requirements.

Analysis of algorithms in terms of complexity helps predict, before writing the code, how fast an algorithm is and the amount of resources needed for its execution. In addition to that, it helps classify and compare the practical difficulty of solving problems using specific algorithms. The complexity of a problem is the complexity of the best algorithms that solves it.

The amount resources required to run an algorithm generally depends on the input size, that's why the complexity is expressed as a function of the input size $n \rightarrow f(n)$.

Complexity of an algorithm depends either on the requirements in terms of time or in space, that why we talk about two types of complexity:

- Time complexity:
  Generally, it indicates the number of required elementary operations on an input of size n, where elementary operations are assumed to take a constant amount of time on a given computer and change only by a constant factor when run on a different computer.
- Space complexity:
  Generally, it indicates the amount of memory required by an algorithm on an input size n.

Examples of time complexities:

- Constant time O (1):
  When the algorithm is not dependent on the input size n.
  Example: A function that returns the first element of an array, irrespective of the length of the array(n), the runtime to get the first element in an array of any length is the same.
- Linear time O(n):

An algorithm is said to have a linear time complexity when the running time increases linearly with the length of the input. When the function involves checking all values in input data, such a function has Time complexity with order O(n).

Example: summation of all elements in an array.

- Logarithmic time O(log(n)):

  Example: Binary Search

- Quadratic time $O(n^2)$:

  The runtime of the algorithm is directly proportional to the square of the size of the input.

  Example: Multiplying two n-digit numbers by a simple algorithm.

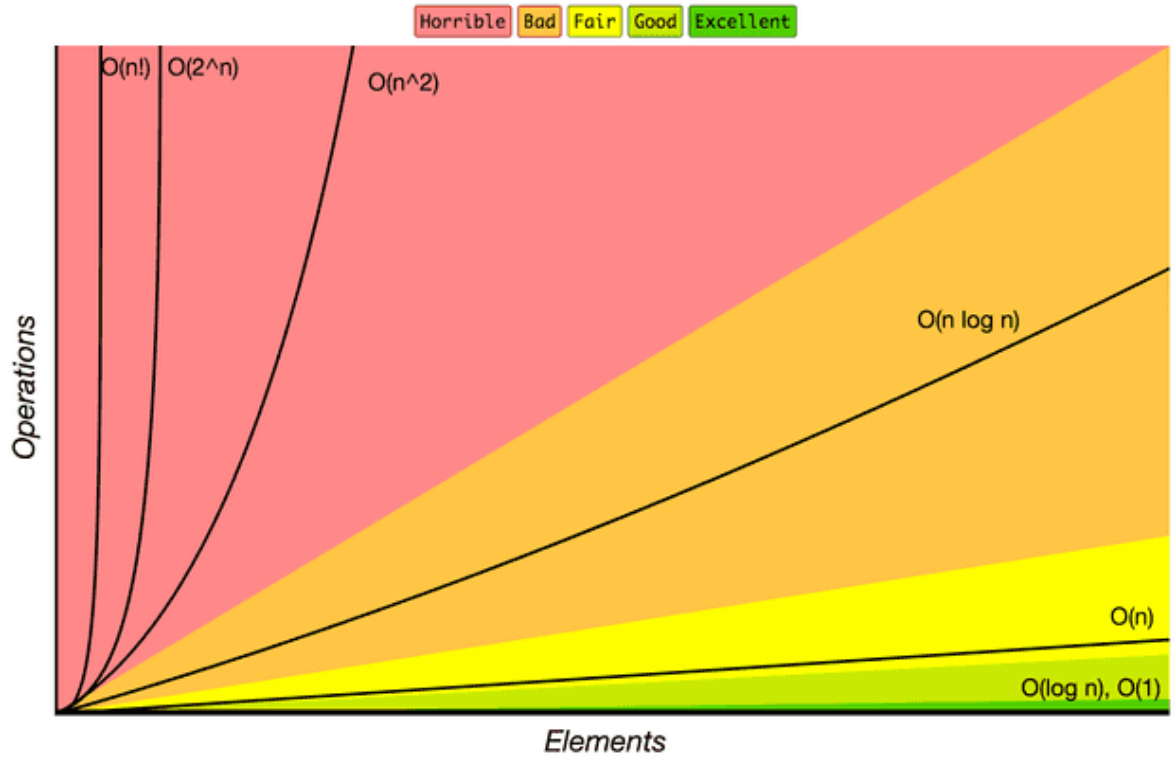


Figure 54: Computational complexity curves

As we can see in the figure 54, depending on the complexity of the algorithm, the amount of time needed to solve a problem may increase drastically, and might even reach unrealistic durations, and this is what gives a significant importance to the study of computational complexity, and give the reduction of it to a minimum amount the most priority when designing an algorithm.

In the context of coverage path planning, the study of the coverage path planning problem will give us insights to the types of algorithms that we should focus on, as this depends highly on the problem's complexity.

II.1.2.2 NP-Class:

Depending on the complexity of the problem, there are many classes that a problem might belong to, and this is relative to its time and space consumption while executed. And a complexity class is then a set of problems with related complexity.

There are many complexity classes:

- P Class.
- NP Class.
- CoNP Class.
- NP Hard.
- NP Complete.

P class corresponds to the polynomial time complexity class. It contains all decision problems that can be solved using a polynomial amount of computation time. The solution is a P problem is easy to find, and it can be solved in both theory and practice.

NP class is the set of Non-deterministic Polynomial Time complexity problems, The solutions to this type of problems can be verified in a polynomial time, but on the other hand, many of those problems take exponential time to solve, as far as no one can tell that it can be solved in polynomial time.

An NP-hard problem is at least as hard as the hardest problem in NP. Therefore, an NP-hard problem is a problem that can be translated into one for solving any NP-hard problem, and being able to solve it in polynomial time means that all other problems in the NP-hard class can also be solved in polynomial time as well, and this the famous question of: Is P=NP?

The next section shows a detailed proof of NP-hardness of the coverage path planning problem by reducing a known NP-hard to it, which is the lawn mowing problem.

II.1.2.3 Proof of NP-hardness:

The way to prove that a problem is NP-hard, is to show that another known NP-hard problem can be reduced to it, only then we can say that our problem is at least as hard as the known NP-hard problem, which makes it NP-hard itself, as the definition of this class states.

The CPP is an application of, or at least has similar characteristics in the approach with, many other problems that are proven to be NP-hard, like the well-known traveling salesman problem, where an agent must find the shortest possible route that visits every city once end returns to the starting point, while given a set of cities and the distance between every pair of them.

In this proof, we focus on the "lawn mowing" problem, which is very similar to the CPP problem in its formulation. Thus, we study NP-hardness proof of the shortest path problems: "lawn mowing" and the "milling", and project it on the CPP problem.

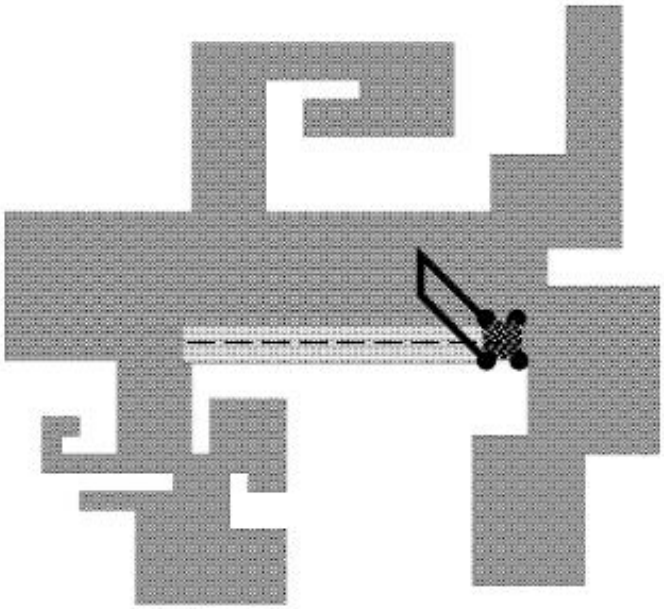


Figure 55: The lawn mowing problem

Those problems are defined as follows: given a region in the plane and given the shape of a "cutter" (typically, a circle or a square), find a shortest path for the cutter such that every point within the region is covered by the cutter at some position along the path. In the milling version of the problem, the cutter is constrained to stay within the region.

Consider the following two closely related problems:

- For a given region covered by grass, find a short path along which to move a lawn mower, such that all the grass is cut.
- Automatically generating tool paths for Numerically Controlled pocket machining: Given a workpiece, a cutter head, and the shape of a "pocket" to be milled in the workpiece, determine a route for the cutter such that the cutter removes exactly the material that lies within the pocket.

The difference between this milling problem and the lawn mowing problem is that in the milling problem we do not allow the cutter to exit the region (pocket) that it must cover, while in the lawn mowing problem it is permitted for the cutter to "mow" over non-grass regions (e.g., one may push the lawn mower over the sidewalk while cutting the grass).

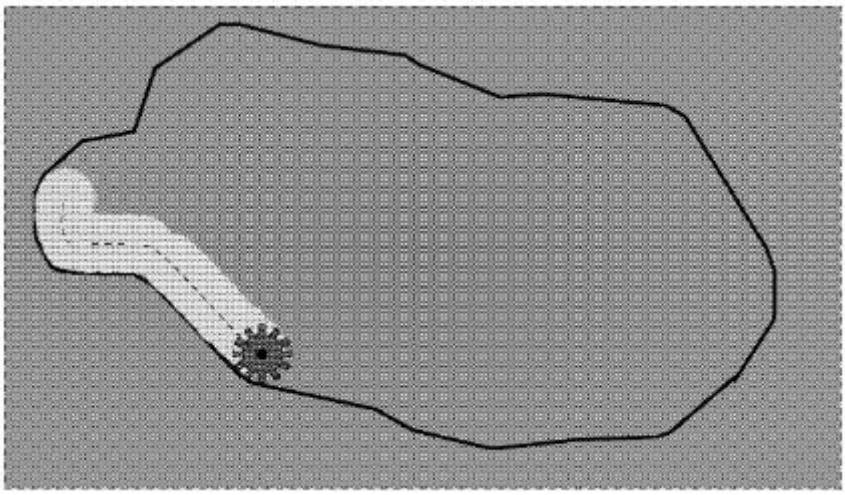


Figure 56: The milling problem

Let R be a planar region, that describes the grass to be mowed or the pocket to be machined. R might consist of many connected polygonal regions. And let χ be a cutter, which is assumed to be either a circle or an axis-aligned square. We refer to the cutter χ by its CenterPoint. And we denote by $\chi(p)$ the placement of χ at the point $p \in \mathbb{R}^2$.

We denote by π the path that covers all points of region R by the placement χ, such that: $R \subseteq \bigcup_{p \in \pi} \chi(p)$ for a lawn mower tour, and that: $R = \bigcup_{p \in \pi} \chi(p)$ for the milling tour, because no placement should go out of the region to respect the machining constraints.

Given all these problem definition guidelines, the goal in this section is to prove the next theorem: The lawn mowing problem for a connected polygonal region is NP-hard for the case of an aligned unit square cutter χ.

The proof to this theorem makes use of the reduction from HAMILTONIAN CIRCUIT IN PLANAR BIPARTITE GRAPHS WITH MAXIMUM DEGREE 3 which is an NP-hard problem to the problem of HAMILTONIAN CIRCUIT IN GRID GRAPH, because the "lawn mowing" problem and the "milling" problem can be mathematically modeled as seeking a Hamiltonian circuit in a grid graph, and from there we work on the reduction from HAMILTONIAN CIRCUIT IN PLANAR BIPARTITE GRAPHS WITH MAXIMUM DEGREE 3 to it [22].

But first we need to go through a set of definitions of some related mathematical concepts:

- Circuit:
  A sequence of adjacent nodes starting and ending at the same node. Circuits never repeat edges. However, they allow repetitions of nodes in the sequence.
- Hamiltonian circuit:
  This circuit is a closed path that visits every node of a graph exactly once.
- Planar graph:
  A graph that can be embedded in the plane.
- Bipartite graph:
  A bipartite graph, also called a bigraph, is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent.

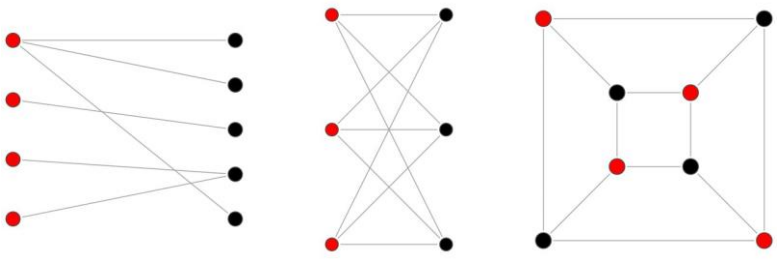


Figure 57: Bipartite graphs

- Degree of a graph:
  It is the largest vertex degree in a graph, and the degree of a vertex is simply the number of vertices connecting it.

First, a planar bipartite graph G with n vertices, each of maximum degree 3 can be represented by a grid graph $\bar{G}$ having a number of vertices m of order n, such that $\bar{G}$ has a Hamiltonian circuit if and only if G has a Hamiltonian circuit. Next, we define the (polygonal) region R to be the union of all placements of χ (a unit square) at the (grid) vertices of $\bar{G}$. Figure 59 shows an example of this construction that corresponds to the bipartite graph shown in Figure 58

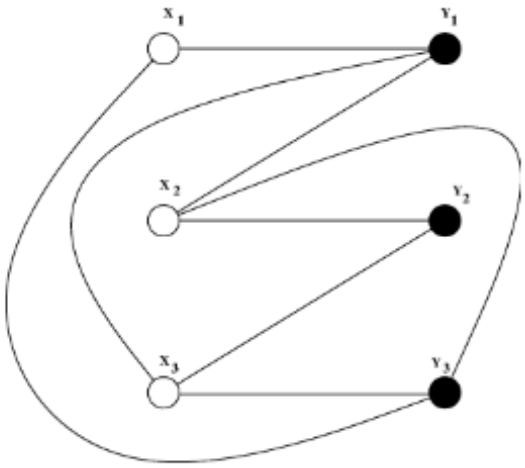


Figure 58: A planar bipartite graph G, with maximum vertex degree 3

We can see that the existence of a tour of length m on the grid vertices of $\bar{G}$ implies the existence of a lawn mower tour of length m. On the other hand, a lawn mower tour of length m can mow all of R (which has area m) only if no point in the region is mowed more than once. This means that the tour partitions the region R into nonoverlapping strips (rectangles) of width 1; clearly, these strips must have integer length. Since traveling a strip corresponds to traveling the associated grid vertices, this implies that a lawn mower tour of length m induces a tour of length at most m in the grid graph. (See Figure 56.)

Thus, we can reconstruct the problem in both directions. Therefore, the HAMILTONIAN CIRCUIT IN PLANAR BIPARTITE GRAPHS WITH MAXIMUM DEGREE 3 which is an NP-can certainly be reduced to the problem of HAMILTONIAN CIRCUIT IN GRID GRAPH.

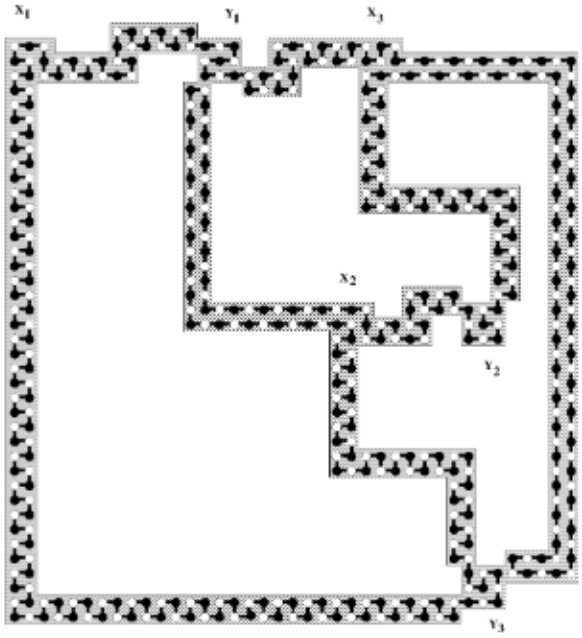


Figure 59: The construction used in proving the NP-hardness of the lawn mowing problem

Having this result at hand shows clearly that the original problem of coverage path planning is NP-hard. Due to this NP-hardness of the problem, optimum solution can be approximated using various methods which we will discuss in the next chapter.

The Covering problem in mathematics and computer science

Definition and results

The covering problem has been studied for a long time in both combinatorial mathematics and computer science. Covering problems are computational problems that ask whether a certain combinatorial structure covers another, or how large the structure has to be to do that. [23]

Covering problems are considered to be minimization problems as they try to minimize the occurrences of the covering set elements for an objective of optimization. On the other hand, the associated dual problem is the packing problem, which is a maximization problem, seeking to find the optimal combination of standard elements to be packed in a bigger container.

The next figure two different configurations of cercle packing in a plane:

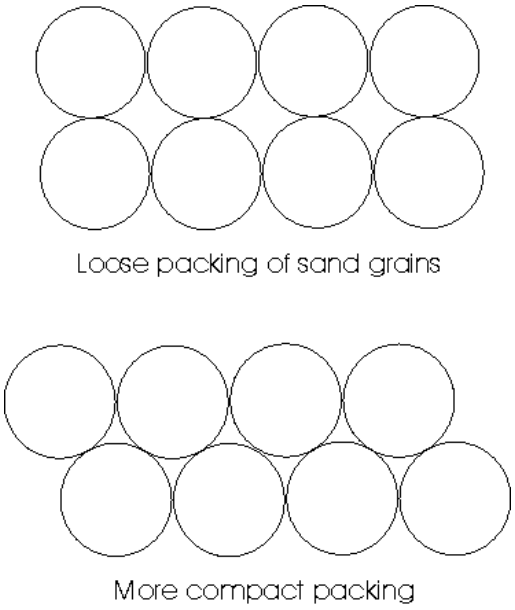


Figure 60: Two circle configurations in the plane

The goal of circle packing problem is to find an arrangement of identical circles that covers as much area as possible starting first within a contained space and then expanding to the infinite plane. The main objective in this operation is to maximize the density of the package, meaning covering the largest possible proportion of the given area using nonoverlapping circles. Mathematically, the density is defined as follows:

$$Density(C_r) = \frac{Area(C_r)}{Area(R^n)}$$

Where C_r is the circle if radius r and R^n is the 2-dimensional space R^2.

The arrangement of the circles has a significant effect on this density and therefore on the quality of the packing. Circles can be arranged in a lattice and represent a symmetrical pattern. It is proven by the Axel Thue Theorem that the densest circle packing in the plane is the regular hexagonal lattice. The density of this lattice is $\frac{\pi}{12} \simeq 0.90690$

This result is correct for an infinite plane, but for a finite region with limits, we might encounter an inconsistency of packing efficiency. For example, the image shows the case of both square and hexagonal lattices in 4x4 square region where the result of the previous theorem is not accurate:

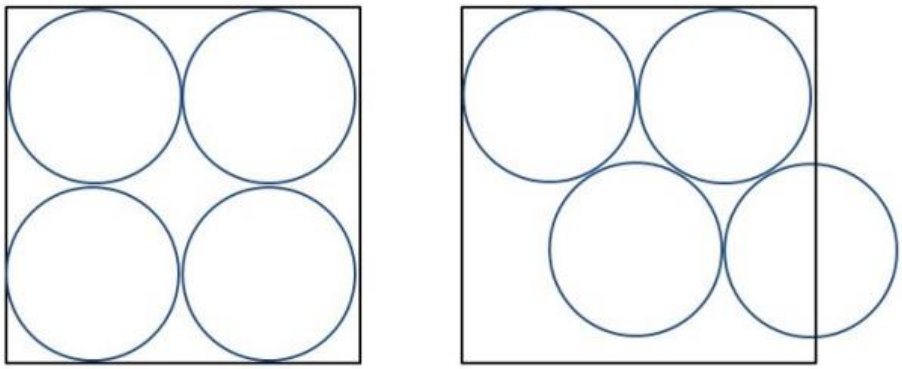
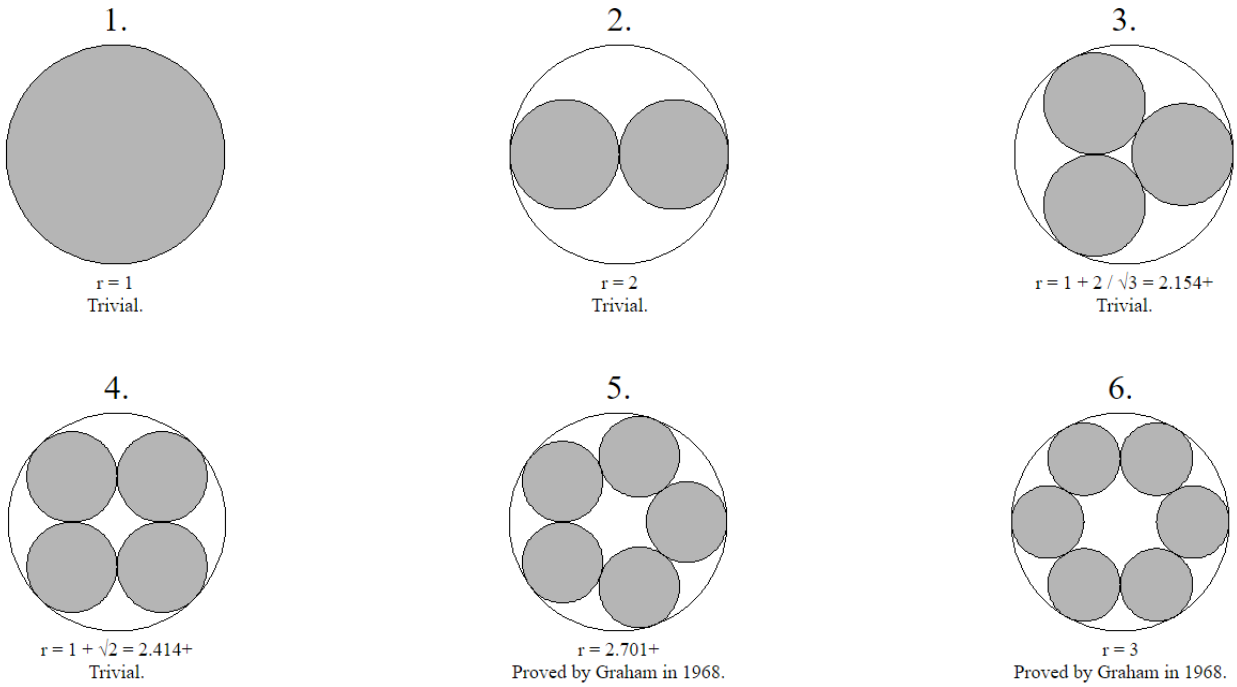


Figure 61: Circle lattices in finite space

Studies in the field of packing elements inside a container have considered many combinations: Circles in squares, squares in squares, circles in triangles, circles in hexagons, circles in circles and many more other ones. And in every case, optimal configurations have been proven for every number of elements to be contained (n).

The following pictures show how n unit circles can be packed inside the smallest known circle of radius r. [24]



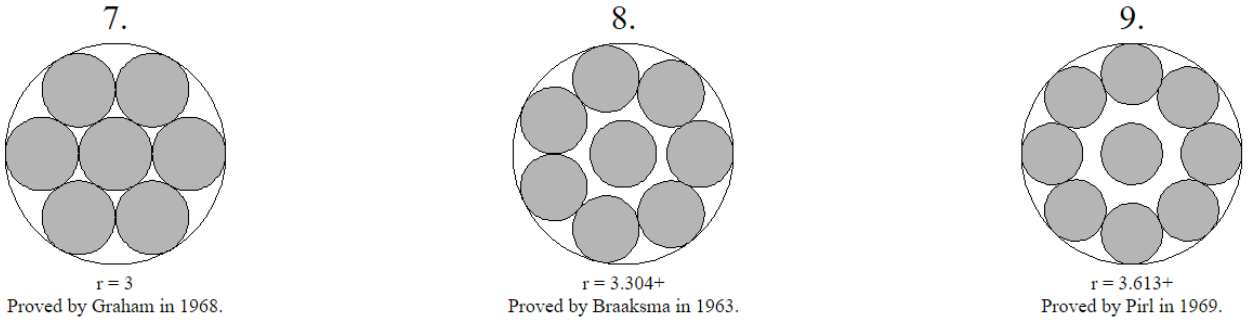


Figure 62: Circle packing in a circle

And the list goes on to a very large number n, providing us with a configuration for each number n, proven using mathematics or computer science.

On the hand, the covering problem, as defined earlier, ask whether a certain combinatorial structure covers another, or how large the structure has to be to do that. The next figure shows an example of covering a circle of radius r, using n=19-unit circles:

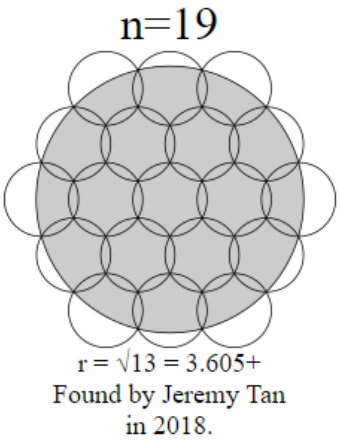


Figure 63: Circle coverage in a circle, case of n=19

The methodology used to find the proof of every case of n insures the complete coverage of the containing circle, without nonoverlapping condition. And this is the difference between the packing and the covering.

Next, we show different proven results for the problem of circle covering using unit circles, and the associated radius of the containing circle:

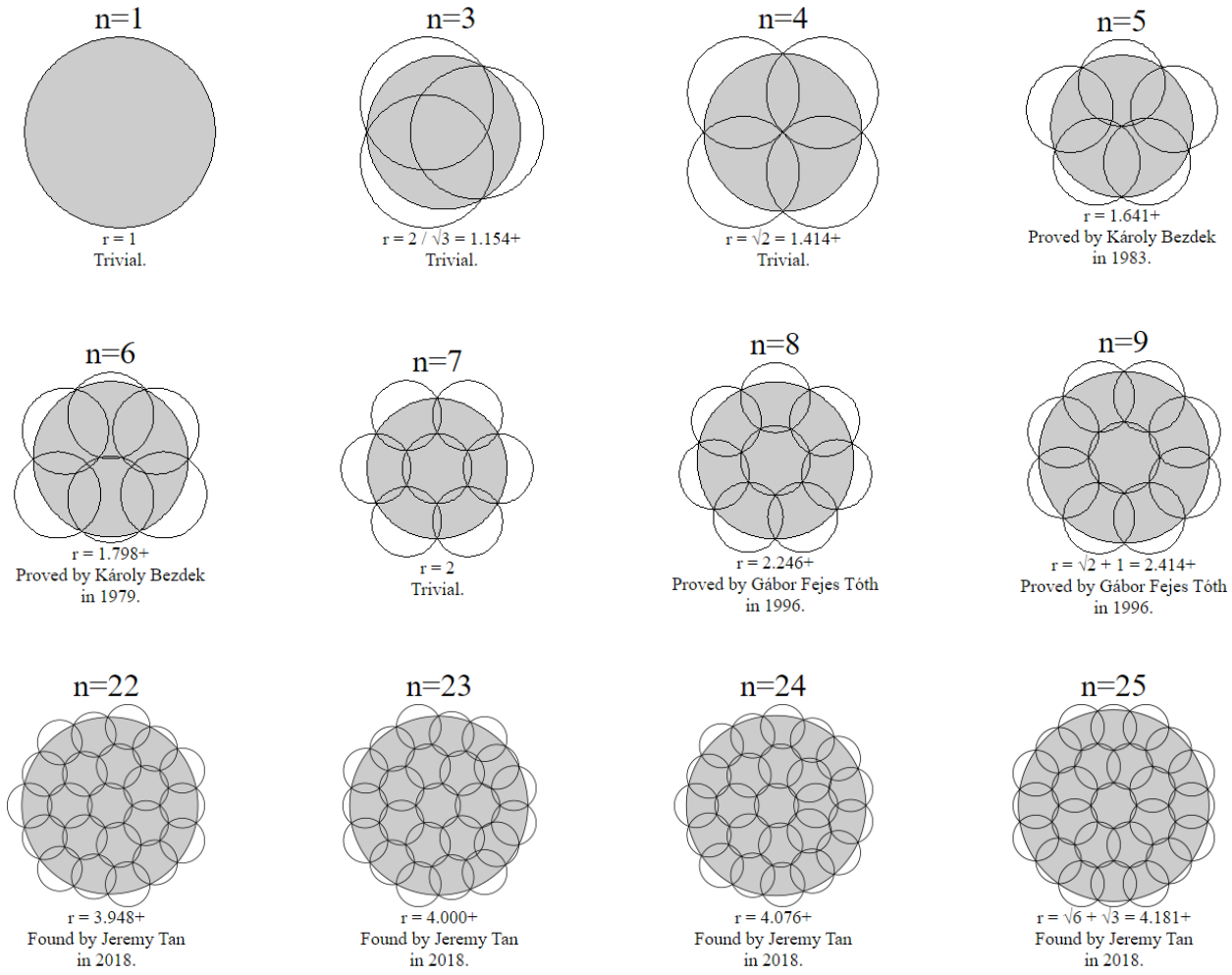


Figure 64: Circle coverage using unit circles

Those results come in individual contributions from mathematicians, and for every number of n separately. For example, for n=8, 9 and 10, GABOR FEJES T'OTH [25] shows that the maximum radius of a circular disk that can be covered by n closed unit circles r_n, follows the next formula:

$$r_n = 1 + 2\cos\left(\frac{2\pi}{n-1}\right) \quad for\ n = 8, n = 9\ and\ n = 10.$$

Many of the previous mathematical results contribute to the latest ones in the field of coverage problems. For example, a theorem proven by Richard Kershner and published in 1939[26]. This theorem is stated as follows: Let M denote a bounded plane point set and let $N(\epsilon)$ be the minimum number of circles of radius ϵ which can cover M. Then:

$$\lim_{\varepsilon \to 0} \pi\epsilon^2 N(\epsilon) = \left(\frac{2\pi\sqrt{3}}{9}\right) meas\ \bar{M}$$

Where $\bar{M}$ denotes the closure of M.

Applications of the mathematical coverage results

These shown results in both fields of packing or covering sets can be used in many industrial and technical applications, we can cite for example a 2016 paper entitled: Unmanned Aerial Vehicle with Underlaid Device-to-Device Communications: Performance and Tradeoffs [27]. Where the authors study and design a wireless communication system using UAVs.

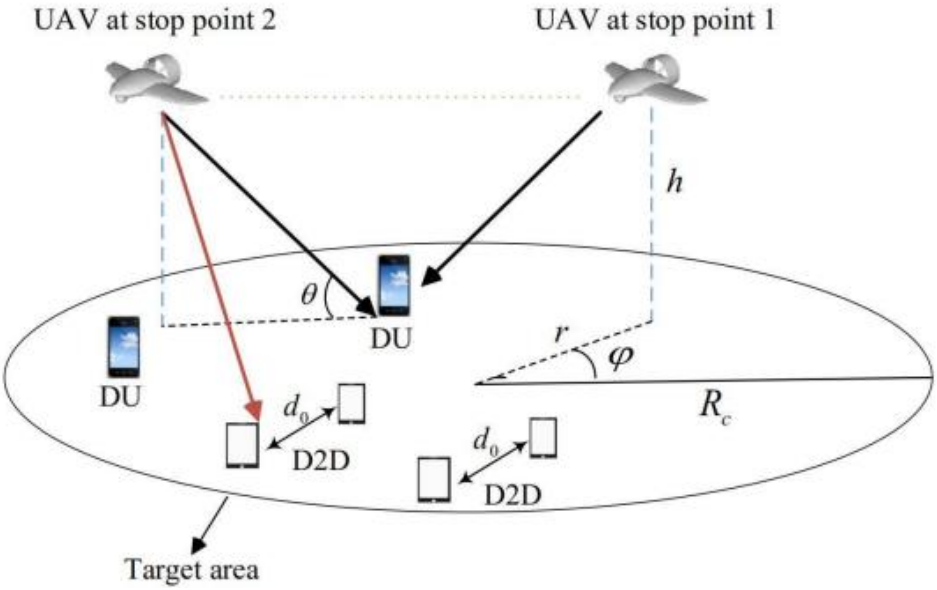


Figure 65: Network model

The figure above shows the network model. There are two types of users in the network, the downlink users (DUs) which receive data directly from the UAV, and the D2D users which communicate directly with one another. The goal of this design is to ensure the complete coverage of any specific area that could be bounded by a circular region.

The stop points are the points are the points where the UAVs stop to transmit data with the DUs, and they need to be determined in an optimal way to guarantee the complete coverage of the wider disk. This is where the mathematical results discussed in the previous section come to action, and the proposed distribution of the stop points is exactly the one proposed by the five-disk covering problem, and the distribution changes in terms of the number of stop points according to the mathematical results discussed earlier.

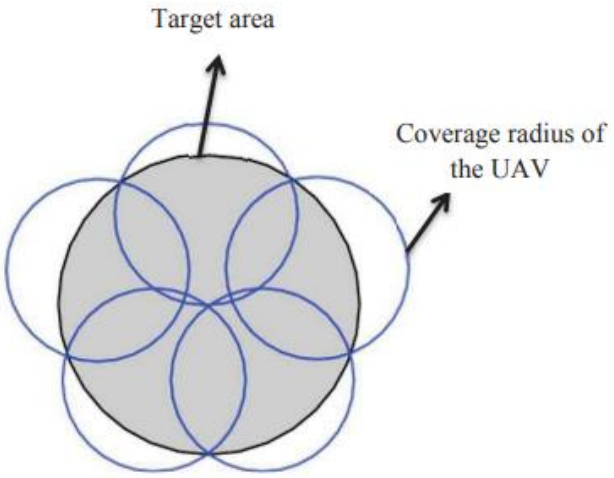


Figure 66: Coverage of a circle using 5-unit circles

Mathematical coverage and our project

Having a basic knowledge in the mathematics of the covering problems is essential, in order to have a clear idea about the results and methodologies behind it. In fact, as we will see in the proposed method, we use a continually constructed set of the same items (circles) to cover region.

But unfortunately, the use of the previously mentioned results wouldn't probably be applicable in our case for the next reasons:

- The shape of the container region is not necessarily standard (circle, square...), but it can be of any complicated shape.
- The results are not general, but for every number of items in the covering set, we have a result proven in its special way. Therefore, extending the dimension of the problem wouldn't be possible beyond existing results.
- The coordinates of the distributed items are not specified.
- The region might contain obstacles and the coverage is done in an online way.

For those reasons, we study and propose different approaches of dealing with problem in the next section.

Chapter 2 Approaches to solve CPP problem

We discuss in this chapter mainly an optimal online coverage path planning with energy constrains and the methodology and how it would insure the correctness and the convergence of the algorithm.

II.2.1 Optimal online CPP with energy constraints

This paper [28] introduces a way of navigation using coverage path planning of a polygonal environment that might contain obstacles. The robot doing the navigation has a limited energy budget, and every time it runs out of its energy, a pullback to the charging station is necessary, this means that after a full charging of the battery, the distance that the robot can move is limited, and therefore, visiting all points of the environment in one tour may not be possible, and thus planning a multiple path trip is necessary.

Online coverage means that not only the planar polygonal environment P is not known a priori to the robot, but also no prior knowledge of the obstacles and their shapes is considered before the navigation. But on the other hand, building the map of the environment will make use of a GPS sensor, and avoiding obstacles will be done using an obstacle detection sensor such as the laser rangefinder.

The environment is represented as a grid of LxL size cells, and the robot as well is a assumed to have the same size, and moves in a rectilinear way in P. This robot is initially at the unique charging station and its energy consumption is assumed to be proportional to the distance traveled, the length of the maximum path allowed after being fully charged is supposed as B. This means that it can move [B/L] cells, thus the important result, that an environment P cannot be fully covered in one path if its number of cells exceeds [B/L] +1.

The goal of OnlineCPP is to find a set of paths $\Pi = \{\pi 1, \ldots, \pi n\}$ for the robot so that:

- Condition (a): Each path πi starts and ends at S,
- Condition (b): Each path πi has length $|\pi i| \leq B$, and
- Condition (c): The paths in Π collectively cover the environment P, i.e., $\bigcup_{i=1}^{n} \pi_i = P$.

For a goal of optimizing two performance metrics:

- The number of paths (or number of visits to S) in Π, denoted as $|\Pi|$, is minimized, and

- The total lengths of the paths in Π, is minimized.

Figure 60 shows an illustration of an environment with a charging station and two obstacles.

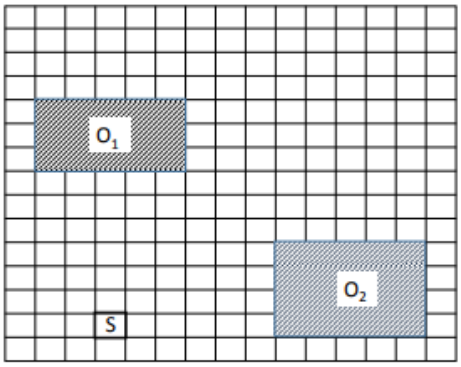


Figure 67: Example of environment with a charging station and two obstacles

This technique of coverage makes use of the Depth First Search algorithm. The robot traverses the cells in forward and backward modes, while constructing incrementally the environment. And the DFS in done level by level to control the charging cost, and the depth of every level is computed based on its energy budget B.

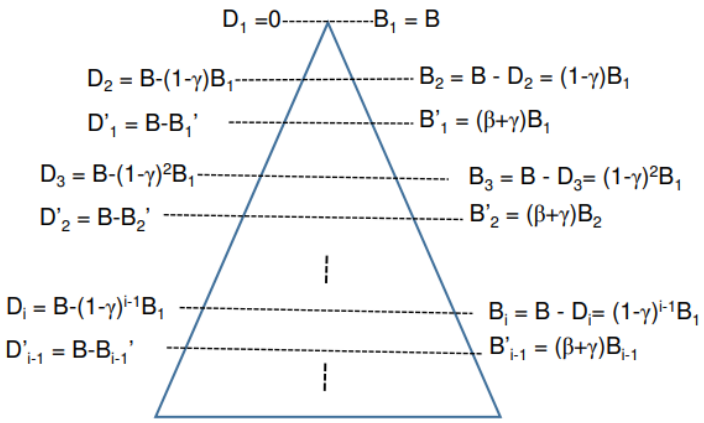


Figure 68: Partitioning of the tree map

Figure 61 shows how the partition of the tree map is calculated. The tree map is a graph constructed incrementally while discovering the environment. As shown in figure 62.

Initially, the robot is at the charging station, this means that the tree map T_p has only one node S. Keeping in mind that a cell might have at most four adjacent nodes if it not in the boundaries or is next to an obstacle. Thus, the robot can move in four directions: West, North, East, and South. Priority here follows the clockwise order starting from the cell on the left.

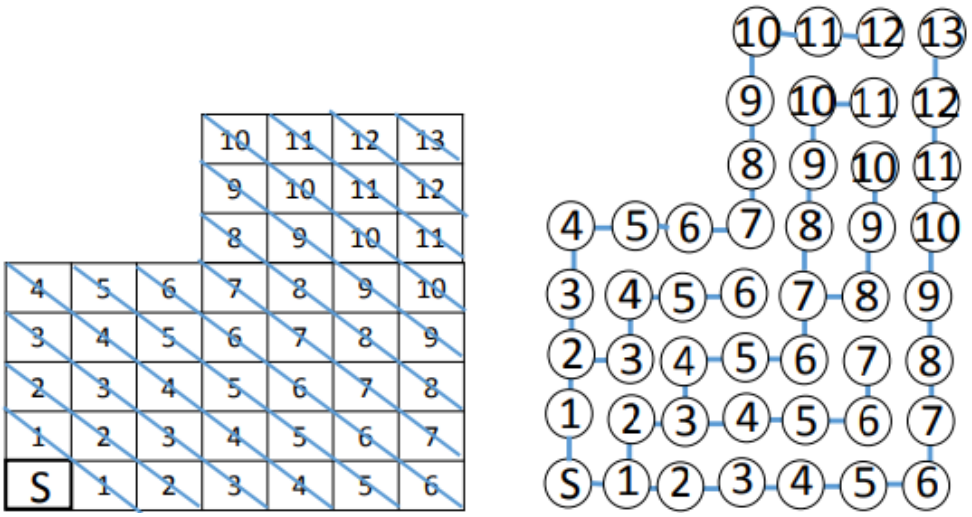


Figure 69: An example of the tree map on the right constructed from the environment P on the left

The robot picks the first cell that is not occupied by an obstacle and adds it to the tree map as shown in the figure 62. The distance from the charging station is also taken into consideration and is associated to every cell in T_p as shown on the right.

The partitioning is done on-the-fly while covering the environment P. $D_1, D_2, \ldots D_i$ denote the level boundaries. The charging station is at level $D_1 = 0$ and the nodes and also the nodes up to depth D_2. The level 2 consists of nodes from D_2 up to D_3, and this structure goes on the same way. B_i denotes the energy budget of the robot the level D_i. D_i' is also defined, such that $D_{i+1} \leq D_i' \leq D_i$.

Calculating all parameters is shown in the figure 61 and they are dependent on γ , β such that $0 < \gamma$, $\beta < 1$. γ and β determine the correctness of the algorithm, for the values $\beta = 3/4$ and $\gamma = 1/10$ ensure that the algorithm correctly and fully covers the environment

Covering nodes of level-i and some cells of level-(i+1) helps in providing extra information about the level i+1 and makes the algorithm analysis easier for both lower and upper bounds on the performance metrics. And in every level, Depth First Search (DFS) is the technique used to traverse all cells of it.
The main idea behind our algorithm is to incrementally explore the environment P by the robot r to fully cover P, while at the same time r constructing a tree map TP of P to keep track of the new frontiers that need to be visited by it to solve online CPP. The algorithms used are displayed below:

Algorithm 1: OnlineCPPAlg

Input: The charging station S and the available energy budget B for r that is initially at S; the environment P is not known to r except that S is inside P and P has a boundary of radius at most $B/2$ centered at S;
$N \leftarrow \{S\}$;
for $i = 1, 2, \ldots, \lceil \log_{\frac{1}{1-\gamma}}(\frac{B}{2}) - 1 \rceil$ **do**
  $D_i \leftarrow \lfloor B - (1-\gamma)^{i-1} B \rfloor$;
  $D_{i+1} \leftarrow \lfloor B - (1-\gamma)^{i} B \rfloor$;
  $B_i = B - D_i$;
  $B'_i \leftarrow \lceil (\beta + \gamma) B_i \rceil$;
  $D'_i \leftarrow \lfloor B - B'_i \rfloor$;
  while there is at least a node of T_P within depth D'_i (from S) that is yet to be visited **do**
    Cover$(S, i, D_i, D'_i, D_{i+1}, N)$;
  $N \leftarrow N'$;

Algorithm 2: Cover$(S, i, D_i, D'_i, D_{i+1}, N)$

if *there is at least one unvisited node in tree* T_P *between depth* D_i *and* D'_i *(*D'_i *inclusive)* **then**
  $v \leftarrow$ the leftmost unexplored node on T_P that is closest from S (the root of T_P);
  move to a node $v_i \in N$ that is closest to v using a path in T_P;
  move to v from v_i through a shortest path in T_P;
  $D_v \leftarrow$ the distance from S to v;
  $B_{remain} = B - D_v$;
  while $B_{remain} \geq D_{v'}$ *for any node* v' *between depth* D_i *and* D'_i *(inclusive)* **do**
    explore the unvisited nodes between depth D and D' (inclusive) using a DFS traversal;
    insert each new node visited in tree T_P making a child appropriately;
    decrease B_{remain} by 1 for each new node the traversal visits;
    $N' \leftarrow$ a set of nodes of T_P that are at depth D_{i+1} (note that $D_i \leq D_{i+1} \leq D'_i$);
  return to S following the tree T_P;

Running this algorithm in a 2D discrete environment with budget B set to 32 would look like the next figure:

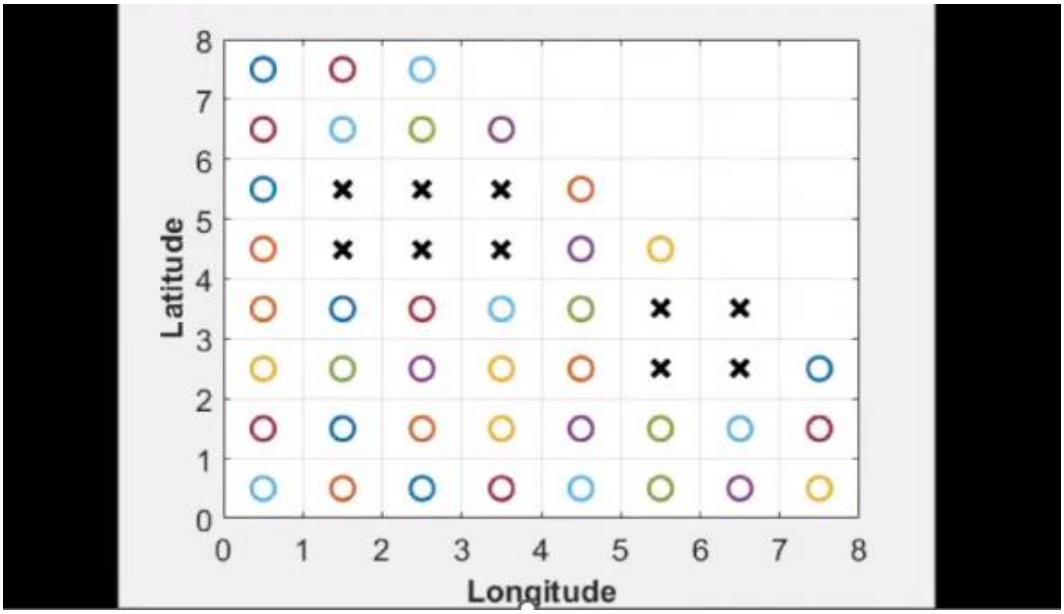


Figure 70: Online coverage path planning algorithm in the environment

II.2.2 Proposed method using reinforcement learning

Online coverage path planning is a relatively complex problem, in a way that handling it properly will certainly a powerful model, designed to take into account major uncertainties about it. And this is due to the fact the environment is continue and might have any configuration in terms of shape or obstacle distribution.

Coverage path planning and reinforcement learning

Reinforcement leaning is highly recommended candidate solution to this type of problems, considering its complexity and uncertainty. However, the majority of proposed methods of solving the coverage path problem are considered to be in a discrete environment. For example, according to a paper named "UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning" [29]. The design of the system is shown in the following figure:

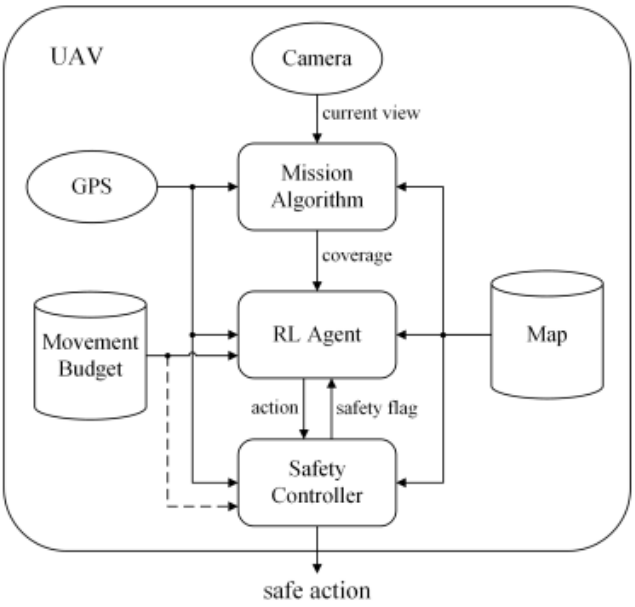


Figure 71: UAV navigation system design

This method uses deep Q-Learning with a reward function based upon the next elements:

- r_{cov} (positive) coverage reward for each target cell that is covered by the UAV's field of view for the first time.
- r_{sc} (negative) safety penalty in case the safety controller (SC) rejects the agent's proposed action.
- r_{mov} (negative) constant movement penalty that is applied for every unit of the movement budget the UAV uses.
- r_{crash} (negative) penalty in case the UAV runs out of movement budget without having safely landed in a landing zone.

Proposed method

The most critical part in the design of a reinforcement learning algorithm is certainly the reward function, as it needs to take into account all important aspects of the agent behavior. For our case, we consider the following aspects in the design of our reward function:

- The robot must cover as maximal area as possible.
- The robot must avoid redundant paths.
- The must avoid local minimums.
- The robot must avoid obstacles.

The expression of the reward function is as follows:

$$R = \ Cumulated\ covered\ area + distribution\ entropy\ - obstacle\ closeness$$

Where $Cumulated\ covered\ area$'s main goal is to ensure that the agent covers as much as possible of the targeted area while at the same time avoids redundant paths. We proceed in this part by calculating the new covered area and try to maximize it. For that, we define virtual circular ranges C_i, which a new circle after every movement of the agent with the possibility of intersection with previous covered area, and we try to maximize the following cumulated formula:

$$A_i = A_{i-1} - \sum_{k=1}^{i-1} C_i \cap C_k$$

Where:

- A_i the approximated cumulated area at the instance i, it is always less or equal to the actual covered area. It is not equal to the actual covered area because the sum of intersections between the new circle and all the previous ones might subtract the region of intersection, if there is any, between the previous circles themselves. And as the following picture shows, the red area would be subtracted twice in this case:

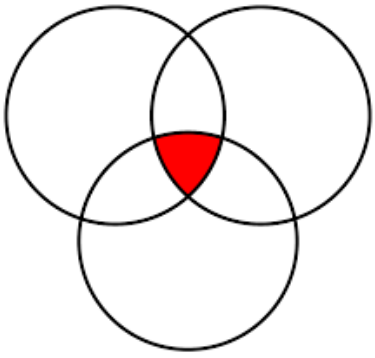
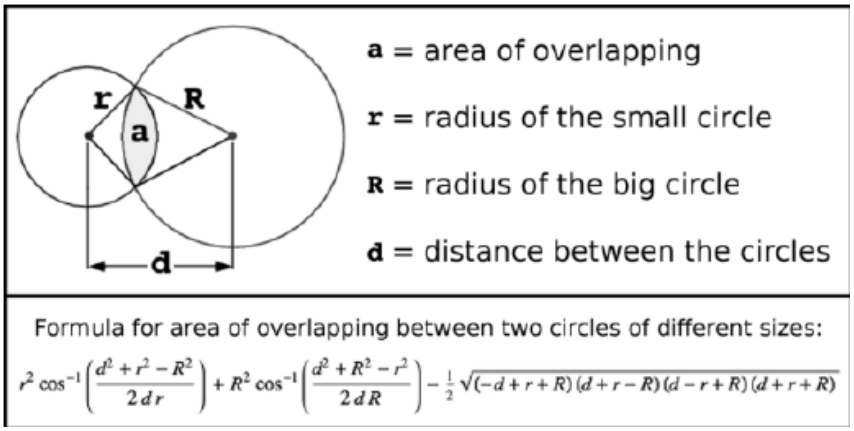


Figure 72: Intersection of three circles

This particularity is what contributes to realizing the second aspect, as maximizing the total calculated value means less intersection between previous circles, which means less redundant paths.

- C_i The circle at instance i, defined by its center coordinates and radius.
- $C_i \cap C_k$: The scalar value of intersection between the two circles, calculated using the following formula:



The role of the $distribution\ entropy$ part in the reward function is to help the agent get out of the local minimums, for example if getting stuck inside a covered sub-region that is part of the overall area with no direction that would improve the covered area calculated area. The agent would probably stay infinitely there, or at least for a long illogical time.

The proposed way to handle this case is by adding entropy to the formula, as maximizing the distribution entropy, using the coordinates of every circle in the overall agent path, would try to ensure getting away from the more concentrated region.

The entropy used in this part is the Kozachenko and Leonenko (1987) estimator, which uses k-nearest neighbor distances to compute the entropy of distribution.

The last part of the reward function is the obstacle avoiding part, that is standard in the field of navigation in general and uses the data captured from the sensors to guarantee that the agent does not get too much closer from the detected obstacles, the formula of this part is:

$$r_{obs} = \min_i\left(\frac{\|d_i\|}{d_s} - 1\right)$$

Where:

- i: index corresponding to a sensor.
- d_i: The distance captured from sensor i.
- d_s: The security distance that the closest distance from the obstacle should be larger than it.

Conclusion and perspectives

This part studied the navigation aspect of the drone. The formulation of the problem helps in determining and approximating the right track of the solution design. As the optimal solution of the coverage path planning problem, there are other promising ways to tackle this problem.

The optimal solution proposed has a guaranteed convergence and correctness, but on another hand, Reinforcement Learning has proven unprecedented efficiency in handling similar problems to the coverage path planning. We aim in the rest of the internship to work on handling this problem using Reinforcement Learning.

Studying, constructing, and comparing between the results of many methods of handling this same problem would give us a clear vision on the best way to achieve the wanted results, as done in the first part. This would also highly contribute to the improvement of the general efficiency of the system and justify the need to replace traditional surveillance systems.

Furthermore, the development of coverage path planning methods, wouldn't only contribute the improvement of the surveillance and supervision processes, but it would also affect many other applications in the general robotics navigation area, like in the search and rescue process, agriculture, mechanical machining and many more other applications.

References

[1] What is Computer Vision? | IBM. (n.d.). IBM - Deutschland | IBM. https://www.ibm.com/topics/computer-vision

[2] Contributors to Wikimedia projects. (2001, October 25). Computer vision - Wikipedia. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Computer_vision

[3] Mihajlovic, I. (2019, April 25). Everything You Ever Wanted To Know About Computer Vision. Here's A Look Why It's So Awesome. Medium. https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

[4] Contributors to Wikimedia projects. (2011, July 20). Deep learning - Wikipedia. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Deep_learning

[5] Contributors to Wikimedia projects. (2016, June 22). *ImageNet - Wikipedia*. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/ImageNet

[6] Deng, J., Dong, W., Socher, R., Li, L., Kai Li, & Li Fei-Fei. (2009). ImageNet: A large-scale hierarchical image database. 2009 IEEE Conference On Computer Vision And Pattern Recognition. doi: 10.1109/cvpr.2009.5206848

[7] Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings Of The IEEE, 86(11), 2278-2324. doi: 10.1109/5.726791

[8] Krizhevsky, A., Sutskever, I., & Hinton, G. (2017). ImageNet classification with deep convolutional neural networks. Communications Of The ACM, 60(6), 84-90. doi: 10.1145/3065386

[9] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., & Anguelov, D. et al. (2015). Going deeper with convolutions. 2015 IEEE Conference On Computer Vision And Pattern Recognition (CVPR). doi: 10.1109/cvpr.2015.7298594

[10] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference On Computer Vision And Pattern Recognition (CVPR). doi: 10.1109/cvpr.2016.90

[11] Rosebrock, A. Deep learning for computer vision with Python. Volumes 1, 2 and 3.

[12] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference On Computer Vision And Pattern Recognition (CVPR). doi: 10.1109/cvpr.2016.91

[13] Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions On Pattern Analysis And Machine Intelligence, 39(6), 1137-1149. doi: 10.1109/tpami.2016.2577031

[14] Faster R-CNN for object detection. (2022). Retrieved 3 July 2022, from https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46

[15] Mean Average Precision (mAP) Explained: Everything You Need to Know. (2022). Retrieved 3 July 2022, from https://www.v7labs.com/blog/mean-average-precision

[17] Zhu, P., Wen, L., Du, D., Bian, X., Fan, H., Hu, Q., & Ling, H. (2021). Detection and Tracking Meet Drones Challenge. IEEE Transactions On Pattern Analysis And Machine Intelligence, 1-1. doi: 10.1109/tpami.2021.3119563

[18] GitHub - VisDrone/VisDrone2018-DET-toolkit: Object Detection in Images toolkit for VisDrone2019. (2022). Retrieved 3 July 2022, from https://github.com/VisDrone/VisDrone2018-DET-toolkit

[19] Train Custom Data 📌 - YOLOv5 Documentation. (2022). Retrieved 3 July 2022, from https://docs.ultralytics.com/tutorials/train-custom-datasets/

[20] models/tf2_detection_zoo.md at master · tensorflow/models. (n.d.). GitHub. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

[21] Galceran, E., & Carreras, M. (2013). A survey on coverage path planning for robotics. Robotics And Autonomous Systems, 61(12), 1258-1276. doi: 10.1016/j.robot.2013.09.004

[22] Arkin EM, Fekete SP, Mitchell JS. Approximation algorithms for lawn mowing and milling. Computational Geometry 2000;17(1–2):25–50.

[23] Wikimedia Foundation. (2022, September 2). Covering problems. Wikipedia. Retrieved September 4, 2022, from https://en.wikipedia.org/wiki/Covering_problems

[24] Erich's Packing Center. Erich-friedman.github.io. (2022). Retrieved 9 September 2022, from https://erich-friedman.github.io/packing/.

[25] Goodman, J. (2011). Combinatorial and computational geometry. Cambridge Univ. Press.

[26] Kershner, R. (1939). The Number of Circles Covering a Set. American Journal Of Mathematics, 61(3), 665. https://doi.org/10.2307/2371320

[27] Mozaffari, M., Saad, W., Bennis, M., & Debbah, M. (2016). Unmanned Aerial Vehicle With Underlaid Device-to-Device Communications: Performance and Tradeoffs. IEEE Transactions On Wireless Communications, 15(6), 3949-3963. https://doi.org/10.1109/twc.2016.2531652

[28] Sharma, G., Dutta, A., Kim, J.-H.: Optimal online coverage path planning with energy constraints. In: AAMAS, pp. 1189–1197 (2019)

[29] Theile, M., Bayerlein, H., Nai, R., Gesbert, D., & Caccamo, M. (2020). UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning. 2020 IEEE/RSJ International Conference On Intelligent Robots And Systems (IROS). https://doi.org/10.1109/iros45743.2020.9340934

Appendix

Some VisDrone customization captures:

```
[ ] %cd /content/visdrone/VisDrone2019-DET-test/annotations

    /content/visdrone/VisDrone2019-DET-test/annotations
```

```
[ ] import glob
    labels_files = glob.glob("*.txt")
    labels_files = sorted(labels_files)
```

```
[ ] person_count = 0
    bicycle_count = 0
    car_count = 0
    van_count = 0
    truck_count = 0
    motor_count = 0

    for i in range(len(labels_files)):

      with open(labels_files[i], 'r') as f:
        lines = f.readlines()
        content = [x.strip() for x in lines]

        bbox = []

        for x in content:

          y = x.split(',')

          first = y[0] + "," + y[1] + "," + y[2] + "," + y[3] + "," + y[4]
          last = y[6] + "," + y[7]
```

```
          if y[5] == '1' or y[5] == '2':
            bbox.append(first + ",0," + last)
            person_count += 1

          elif y[5] == '3':
            bbox.append(first + ",1," + last)
            bicycle_count += 1

          elif y[5] == '4':
            bbox.append(first + ",2," + last)
            car_count += 1

          elif y[5] == '5':
            bbox.append(first + ",3," + last)
            van_count += 1

          elif y[5] == '6':
            bbox.append(first + ",4," + last)
            truck_count += 1

          elif y[5] == '10':
            bbox.append(first + ",5," + last)
            motor_count += 1

      with open(labels_files[i], 'w') as f:
        f.truncate(0)
        for line in bbox:
          f.write('%s\n' % line)
```

```
[ ] print("person count:", person_count)
    print("bicycle count:", bicycle_count)
    print("car count:", car_count)
    print("van count:", van_count)
    print("truck count:", truck_count)
```

Some annotation transformation to YOLOv5 captures:

Replacing VisDrone original annotations with YOLOv5 annotations

```
[ ] def find_x(bbox_left, bbox_width, image_width):
      image_width = float(image_width)
      absolute_x = int(bbox_left) + 0.5 * int(bbox_width)
      x = absolute_x / image_width
      return str(x)

    def find_y(bbox_top, bbox_height, image_height):
      image_height = float(image_height)
      absolute_y = int(bbox_top) + 0.5 * int(bbox_height)
      y = absolute_y / image_height
      return str(y)

    def find_width(bbox_width, image_width):
      image_width = float(image_width)
      width = int(bbox_width) / image_width
      return str(width)

    def find_height(bbox_height, image_height):
      image_height = float(image_height)
      height = int(bbox_height) / image_height
      return str(height)
```

```
[ ] %cd /content/visdrone/VisDrone2019-DET-test/annotations
```

```
/content/visdrone/VisDrone2019-DET-test/annotations
```

```
[ ] import glob
    labels_files = glob.glob("*.txt")
    labels_files = sorted(labels_files)
```

```
for i in range(len(labels_files)):

  with open(labels_files[i], 'r') as f:
    lines = f.readlines()
    content = [x.strip() for x in lines]

    bbox = []

    for x in content:

      y = x.split(',')

      bbox_left = y[0]
      bbox_top = y[1]
      bbox_width = y[2]
      bbox_height = y[3]

      image_width = images_size[i][0]
      image_height = images_size[i][1]

      place_0_value = y[5]
      place_1_value = find_x(bbox_left, bbox_width, image_width)
      place_2_value = find_y(bbox_top, bbox_height, image_height)
      place_3_value = find_width(bbox_width, image_width)
      place_4_value = find_height(bbox_height, image_height)

      bbox.append(place_0_value + " " + place_1_value + " " + place_2_value + " " + place_3_value + " " + place_4_value)

  with open(labels_files[i], 'w') as f:
    f.truncate(0)
    for line in bbox:
      f.write('%s\n' % line)
```

Some TFOD training pipeline captures:

```
[ ] !mkdir Tensorflow
    %cd Tensorflow

    !git clone https://github.com/tensorflow/models.git

    import tensorflow as tf
    print(tf.__version__)

    !pip install pycocotools==2.0.1

    %cd /content/Tensorflow/models/research
    !protoc object_detection/protos/*.proto --python_out=.

    !cp object_detection/packages/tf2/setup.py .
    !python -m pip install --use-feature=2020-resolver .
```

```
/content/Tensorflow
Cloning into 'models'...
remote: Enumerating objects: 72952, done.
remote: Counting objects: 100% (129/129), done.
remote: Compressing objects: 100% (78/78), done.
remote: Total 72952 (delta 63), reused 107 (delta 49), pack-reused 72823
Receiving objects: 100% (72952/72952), 579.29 MiB | 18.33 MiB/s, done.
Resolving deltas: 100% (51633/51633), done.
2.8.0
Collecting pycocotools==2.0.1
  Downloading pycocotools-2.0.1.tar.gz (23 kB)
Requirement already satisfied: setuptools>=18.0 in /usr/local/lib/python3.7/dist-packages (from pycocotools==2.0.1) (57.4.0)
Requirement already satisfied: cython>=0.27.3 in /usr/local/lib/python3.7/dist-packages (from pycocotools==2.0.1) (0.29.28)
Requirement already satisfied: matplotlib>=2.1.0 in /usr/local/lib/python3.7/dist-packages (from pycocotools==2.0.1) (3.2.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0.1) (1.4.2)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0.1) (1.21.6)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0.1) (3.0.8)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0.1) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0.1) (2.8.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib>=2.1.0->pycocotools==2.0.1) (4.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib>=2.1.0->pycocotools==2.0.1) (1.15.0)
```

```
!pip list|grep opencv

!pip uninstall opencv-python-headless
!pip install opencv-python-headless==4.1.2.30
```

```
!python object_detection/builders/model_builder_tf2_test.py
```

```
[ ] %cd /content/Tensorflow
    !mkdir workspace scripts
    %cd workspace/
    !mkdir training_demo
    %cd training_demo/
    !mkdir annotations exported-models images models pre-trained-models
    %cd ../..
    %cd scripts
    !mkdir preprocessing
```

```
/content/Tensorflow
/content/Tensorflow/workspace
/content/Tensorflow/workspace/training_demo
/content/Tensorflow
/content/Tensorflow/scripts
```

```
[ ] %cd /content/Tensorflow/workspace/training_demo/annotations
    !wget https://raw.githubusercontent.com/dronefreak/VisDrone-dataset-python-toolkit/master/training/labelmap.pbtxt # should be renamed to label_map.pbtxt
```

```
/content/Tensorflow/workspace/training_demo/annotations
--2022-05-16 02:37:23--  https://raw.githubusercontent.com/dronefreak/VisDrone-dataset-python-toolkit/master/training/labelmap.pbtxt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 464 [text/plain]
Saving to: ‘labelmap.pbtxt’
```

```
[ ] %cd /content/Tensorflow/scripts/preprocessing
    !wget https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/_downloads/da4babe668a8afb093cc7776d7e630f3/generate_tfrecord.py

/content/Tensorflow/scripts/preprocessing
--2022-05-09 10:19:13--  https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/_downloads/da4babe668a8afb093cc7776d7e630f3/generate_tfrecord.py
Resolving tensorflow-object-detection-api-tutorial.readthedocs.io (tensorflow-object-detection-api-tutorial.readthedocs.io)... 104.17.32.82, 104.17.33.82, 2606:4700::6811:2152, ...
Connecting to tensorflow-object-detection-api-tutorial.readthedocs.io (tensorflow-object-detection-api-tutorial.readthedocs.io)|104.17.32.82|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6410 (6.3K) [text/x-python]
Saving to: 'generate_tfrecord.py'

generate_tfrecord.p 100%[===================>]   6.26K  --.-KB/s    in 0s

2022-05-09 10:19:13 (72.7 MB/s) - 'generate_tfrecord.py' saved [6410/6410]
```

```
[ ] !python generate_tfrecord.py -x /content/Tensorflow/workspace/training_demo/images/train -l /content/Tensorflow/workspace/training_demo/annotations/labelmap.pbtxt -o /content/Tensor

Successfully created the TFRecord file: /content/Tensorflow/workspace/training_demo/annotations/train.record
```

```
[ ] !python generate_tfrecord.py -x /content/Tensorflow/workspace/training_demo/images/test -l /content/Tensorflow/workspace/training_demo/annotations/labelmap.pbtxt -o /content/Tensorf

Successfully created the TFRecord file: /content/Tensorflow/workspace/training_demo/annotations/test.record
```

```
[ ] %cd /content
    !wget http://download.tensorflow.org/models/object_detection/tf2/20200711/efficientdet_d1_coco17_tpu-32.tar.gz

/content
--2022-05-16 10:43:31--  http://download.tensorflow.org/models/object_detection/tf2/20200711/efficientdet_d1_coco17_tpu-32.tar.gz
Resolving download.tensorflow.org (download.tensorflow.org)... 142.251.10.128, 2404:6800:4003:c00::80
Connecting to download.tensorflow.org (download.tensorflow.org)|142.251.10.128|:80... connected.
HTTP request sent, awaiting response... 200 OK
```

Faster RCNN ResNet 101

```
[ ] !python model_main_tf2.py --model_dir=/content/gdrive/MyDrive/visdrone_faster_rcnn_resnet101_v3 --pipeline_config_path=models/my_faster_rcnn/pipeline.config

 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.60440654,
 'learning_rate': 0.020512857}
INFO:tensorflow:Step 50300 per-step time 0.437s
I0513 15:58:10.278433 140550645852032 model_lib_v2.py:707] Step 50300 per-step time 0.437s
INFO:tensorflow:{'Loss/BoxClassifierLoss/classification_loss': 0.2877935,
 'Loss/BoxClassifierLoss/localization_loss': 0.4166441,
 'Loss/RPNLoss/localization_loss': 0.3311504,
 'Loss/RPNLoss/objectness_loss': 0.018485537,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 1.0540736,
 'learning_rate': 0.020448763}
I0513 15:58:10.278759 140550645852032 model_lib_v2.py:708] {'Loss/BoxClassifierLoss/classification_loss': 0.2877935,
 'Loss/BoxClassifierLoss/localization_loss': 0.4166441,
 'Loss/RPNLoss/localization_loss': 0.3311504,
 'Loss/RPNLoss/objectness_loss': 0.018485537,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 1.0540736,
 'learning_rate': 0.020448763}
INFO:tensorflow:Step 50400 per-step time 0.438s
I0513 15:58:54.090582 140550645852032 model_lib_v2.py:707] Step 50400 per-step time 0.438s
INFO:tensorflow:{'Loss/BoxClassifierLoss/classification_loss': 0.35627282,
 'Loss/BoxClassifierLoss/localization_loss': 0.27397117,
 'Loss/RPNLoss/localization_loss': 0.30365443,
 'Loss/RPNLoss/objectness_loss': 0.06536381,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.9992623,
 'learning_rate': 0.02038466}
```

```
[ ] import time
    import numpy as np
    from PIL import Image
    import matplotlib.pyplot as plt
    import warnings
    warnings.filterwarnings('ignore')   # Suppress Matplotlib warnings

    from object_detection.utils import label_map_util
    from object_detection.utils import visualization_utils as viz_utils
```

```
[ ] PATH_TO_LABELS = '/content/gdrive/MyDrive/visdron_tfrecords/labelmap.pbtxt'

    category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
                                                                        use_display_name=True)
```

```
[ ] PATH_TO_SAVED_MODEL = '/content/gdrive/MyDrive/exported_faster_rcnn101/saved_model'

    print('Loading model...', end='')
    start_time = time.time()

    detect_fn = tf.saved_model.load(PATH_TO_SAVED_MODEL)

    end_time = time.time()
    elapsed_time = end_time - start_time
    print('Done! Took {} seconds'.format(elapsed_time))
```

```
Loading model...Done! Took 23.22641348838806 seconds
```

```
[ ] def load_image_into_numpy_array(path):
        """Load an image from file into a numpy array.

        Puts image into numpy array to feed into tensorflow graph.
        Note that by convention we put it into a numpy array with shape
```

```
[ ] i = 0

    for image_path in IMAGE_PATHS:

        print('Running inference for {}... '.format(image_path), end='')

        image_np = load_image_into_numpy_array(image_path)

        # Things to try:
        # Flip horizontally
        # image_np = np.fliplr(image_np).copy()

        # Convert image to grayscale
        # image_np = np.tile(
        #     np.mean(image_np, 2, keepdims=True), (1, 1, 3)).astype(np.uint8)

        # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
        input_tensor = tf.convert_to_tensor(image_np)
        # The model expects a batch of images, so add an axis with `tf.newaxis`.
        input_tensor = input_tensor[tf.newaxis, ...]

        # input_tensor = np.expand_dims(image_np, 0)
        detections = detect_fn(input_tensor)

        # All outputs are batches tensors.
        # Convert to numpy arrays, and take index [0] to remove the batch dimension.
        # We're only interested in the first num_detections.
        num_detections = int(detections.pop('num_detections'))
        detections = {key: value[0, :num_detections].numpy()
                      for key, value in detections.items()}
        detections['num_detections'] = num_detections

        # detection_classes should be ints.
```

```
[ ]     viz_utils.visualize_boxes_and_labels_on_image_array(
            image_np_with_detections,
            detections['detection_boxes'],
            detections['detection_classes'],
            detections['detection_scores'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=200,
            min_score_thresh=.30,
            agnostic_mode=False)

        image_rgb = cv2.cvtColor(image_np_with_detections, cv2.COLOR_RGB2BGR)
        cv2.imwrite(IMAGE_PATHS[i] ,image_rgb)
        i = i + 1
        print('Done')
```

```
Running inference for 0000131.jpg... Done
Running inference for 0000176.jpg... Done
Running inference for 0000162.jpg... Done
Running inference for 0000086.jpg... Done
Running inference for 0000067.jpg... Done
Running inference for 0000108.jpg... Done
Running inference for 0000144.jpg... Done
Running inference for 0000036.jpg... Done
Running inference for 0000188.jpg... Done
Running inference for 0000074.jpg... Done
Running inference for 0000111.jpg... Done
Running inference for 0000159.jpg... Done
Running inference for 0000204.jpg... Done
Running inference for 0000058.jpg... Done
Running inference for 0000216.jpg... Done
Running inference for 0000045.jpg... Done
Running inference for 0000158.jpg... Done
Running inference for 0000048.jpg... Done
Running inference for 0000046.jpg... Done
Running inference for 0000214.jpg... Done
Running inference for 0000201.jpg... Done
Running inference for 0000211.jpg... Done
Running inference for 0000037.jpg... Done
```

```
[ ] import cv2
    import numpy as np
    import glob
```

```
[ ] labels_files = glob.glob("/content/vis_data/uav0000137_00458_v/*.jpg")
    labels_files = sorted(labels_files)
```

```
[ ] img_array = []
    for filename in labels_files:
      img = cv2.imread(filename)
      height, width, layers = img.shape
      size = (width, height)
      img_array.append(img)

    out = cv2.VideoWriter('/content/rcnn101_v2.avi', cv2.VideoWriter_fourcc(*'DIVX'), 12, size)


    for i in range(len(img_array)):
        out.write(img_array[i])
    out.release()
```

```
[ ] %cd /content/vis_data
```

```
/content/vis_data
```

```
[ ] !rm -rf uav0000138_00000_v
```