NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF NAVAL ARCHITECTURE & MARINE ENGINEERING
DIVISION OF MARINE ENGINEERING

Diploma Thesis

Recognition of Mechanical Components with Artificial Intelligence and Machine Learning Methods

Nikos Afentoulis

Thesis Committee:

Supervisor:	**C. I. Papadopoulos,**	**Assoc. Prof. NTUA**
Members:	**L. Kaiktsis,**	**Prof.NTUA**
	A. Gkinis,	**Assoc. Prof. NTUA**

Athens, July 2019

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΝΑΥΠΗΓΩΝ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΝΑΥΤΙΚΗΣ ΜΗΧΑΝΟΛΟΓΙΑΣ

Διπλωματική Εργασία

Αναγνώριση Μηχανολογικών Εξαρτημάτων με Μεθόδους Τεχνητής Νοημοσύνης και Μηχανικής Μάθησης

Νίκος Αφεντούλης

Εξεταστική επιτροπή:

Επιβλέπων:	**Χ. Ι. Παπαδόπουλος,**	**Αναπλ. Καθ. ΕΜΠ**
Μέλη:	**Λ. Καϊκτσής,**	**Καθ. ΕΜΠ**
	Α. Γκίνης,	**Αναπλ. Καθ.ΕΜΠ**

Acknowledgments

The present diploma thesis marks the completion of my studies in the National Technical University of Athens in the School of Naval Architecture & Marine Engineering. It has been a very intensive period for me, not only in the scientific area, but also on a personal level. Thus, I would like to express my gratitude to the people that have supported me and helped me throughout this period.

Firstly, I would like to thank my thesis advisors Associate Professor Christos Papadopoulos and Associate Professor Alexandros Ginis. Both have always been willing to help me every time I needed their guidance and advice.

I would also like to express special thanks to the two PhD students, Katerina Giannisi and Orestis Liaskos, who I have been working with during my thesis the past year. Both have vitally supported me and encouraged me throughout this period. Without their crucial assistance I would have never been able to reach the current result of my diploma thesis.

Furthermore, I would also like to thank my friends and classmates for their friendship and support throughout the last five years of my studies.

Last, I would also like to thank everyone else, outside NTUA, who with their words and actions have motivated me and inspired me throughout this difficult year.

Abstract

The engine room of a modern vessel, where all machinery for the operation and propulsion is located, is a large and extremely complicated compartment. It contains all types of equipment, such as the main engine, electric generators, pumps and complex piping systems. Therefore, there is a huge need for having an accurate description of both what type of machinery is included, as well as its exact location in the engine room. Having access to this kind of information can solve various time-consuming and challenging tasks, such as visual inspection of faulty parts, maintenance of machinery, purchasing of spare parts and path planning for retrofitting new systems.

Recent progress in the field of artificial intelligence and machine learning has led to huge potential for applications such as computer vision and object recognition. Specifically, the combination of the improvements into architectures of artificial neural networks and especially convolutional neural networks, as well as the advances in image processing algorithms and the substantial advances in computational power of computer systems, have made the task of real-time object recognition an achievable goal. However, performing object recognition using a machine or deep learning approach requires large amount of accurate training data, which is usually difficult and time consuming to collect.

The present diploma thesis aims to clarify the basic concepts and parameters of artificial (and especially) convolutional neural networks, and their impact on the fields of machine learning and object recognition. Additionally, it focuses on reviewing the current state-of-the-art object recognition algorithms and on analysing their way of function and their evolution over time.

Furthermore, the work in the current thesis attempts to develop a robust object recognition framework that can identify and localize mechanical components of complex systems, such as piping networks. The approach in the current problem is based on object recognition in 2D images using deep learning techniques, and especially convolutional neural networks. Furthermore, the current thesis proposes a method that aims to automate the process of generating a large and accurate dataset which is required to train a custom object detection CNN-based network.

Keywords

Object Recognition, Machine Learning, Deep Learning, Neural Networks, Convolutional Neural Networks

Σύνοψη

Σε ένα πλοίο, ο χώρος του μηχανοστασίου, όπου βρίσκεται όλος ο μηχανολογικός εξοπλισμός για την λειτουργία και την πρόωση του πλοίου, αποτελεί ένα μεγάλο και ιδιαίτερα πολύπλοκο χώρο. Ειδικότερα, περιέχει εξοπλισμό, όπως η κύρια μηχανή, ηλεκτρογεννήτριες, αντλίες, αλλά και πολύπλοκα συστήματα σωληνώσεων. Για αυτόν τον λόγο, υπάρχει σημαντική ανάγκη για μία ακριβή περιγραφή, τόσο του περιεχομένου των μηχανημάτων που υπάρχουν, όσο και της ακριβής θέσης του στον χώρο του μηχανοστασίου. Η πρόσβαση σε τέτοιου είδους πληροφορίες μπορεί να συντελέσει στην επίλυση ιδιαίτερα δύσκολων προβλημάτων, που απαιτούν πολύ χρόνο για την επίλυσή τους, όπως η αναγνώριση προβληματικών εξαρτημάτων, η συστηματική συντήρηση μηχανημάτων, αλλά και οι παραγγελίες ανταλλακτικών εξαρτημάτων.

Η πρόσφατη πρόοδος στον τομέα της τεχνητής νοημοσύνης και της μηχανικής μάθησης έχει οδηγήσει σε τεράστιες δυνατότητες όσον αφορά πολλές διαφορετικές επιστημονικές περιοχές, όπως η όραση υπολογιστών και η αναγνώριση αντικειμένων/προτύπων. Συγκεκριμένα, ο συνδυασμός των βελτιώσεων στις αρχιτεκτονικές των τεχνητών νευρωνικών δικτύων και ειδικότερα των συνελικτικών νευρωνικών δικτύων, όπως επίσης και η πρόοδος στους αλγορίθμους επεξεργασίας εικόνας , καθώς και η σημαντική αύξηση της διαθέσιμης υπολογιστικής ισχύος των συμβατικών υπολογιστικών συστημάτων, έχουν καταστήσει το πρόβλημα της αναγνώρισης αντικειμένων σε πραγματικό χρόνο μία σχετικά απλή διαδικασία. Ωστόσο, η αναγνώριση αντικειμένων με μεθόδους μηχανικής η βαθιάς μάθησης απαιτεί την ύπαρξη μεγάλου και έγκυρου υλικού για εκπαίδευση, το οποίο είναι δύσκολο και απαιτεί πολύ χρόνο για να συλλεχθεί.

Αρχικά, η παρούσα διπλωματική εργασία στοχεύει στην επεξήγηση των βασικών εννοιών και παραμέτρων των τεχνητών νευρωνικών δικτύων καθώς επίσης και στην σημαντική επίδραση που έχουν στους τομείς της μηχανικής μάθησης και της αναγνώρισης αντικειμένων. Επιπρόσθετα, η εργασία αυτή εστιάζει στην βιβλιογραφική ανασκόπηση των αλγορίθμων αναγνώρισης αντικειμένων τελευταίας τεχνολογίας και ευρείας χρήσης και επικεντρώνεται στον τρόπο λειτουργίας τους αλλά και στην χρονική εξέλιξή τους.

Επίσης, η παρούσα εργασία επιχειρεί να δημιουργήσει ένα εύρωστο υπολογιστικό δίκτυο αναγνώρισης αντικειμένων, το οποίο έχει την δυνατότητα να ταυτοποιεί το είδος αλλά και να προσδιορίζει την ακριβή θέση μηχανολογικών εξαρτημάτων σε σύνθετα συστήματα στο μηχανοστάσιο ενός πλοίου, όπως είναι τα δίκτυα σωληνώσεων. Η συγκεκριμένη προσέγγιση βασίζεται στην αναγνώριση αντικειμένων σε 2-διάστατες εικόνες με χρήση μεθόδων βαθιάς μηχανικής μάθησης, και ειδικότερα συνελικτικών νευρωνικών δικτύων. Επιπρόσθετα, η παρούσα διπλωματική εργασία προτείνει μία μέθοδο η οποία στοχεύει στην αυτοματοποίηση της διαδικασίας δημιουργίας δεδομένων για την εκπαίδευση ενός δικτύου αναγνώρισης αντικειμένων με χρήση ΣΝΝ.

Λέξεις Κλειδιά

Αναγνώριση Αντικειμένων/Προτύπων, Μηχανική Μάθηση, Βαθιά Μηχανική Μάθηση, Νευρωνικά Δίκτυα, Συνελικτικά Νευρωνικά Δίκτυα.

Table of Contents

Nomenclature

AI	Artificial Intelligence
OR	Object Recognition
ANN	Artificial Neural Network
MLP	Multilayer Perceptron
σ	Activation Function
CNN	Convolutional Neural Network
lr	Learning Rate
mAP	Mean Average Precision
YOLO	You Only Look Once Algorithm
YOLOv3	You Only Look Once Algorithm Version 3
FC	Fully Connected Layer
ReLU	Rectified Linear Unit
LReLU	Leaky Rectified Linear Unit
BN	Batch Normalization
IoU	Intersection over Union

List of Figures

1. Introduction

1.1 Artificial Intelligence

Artificial Intelligence (AI) is the simulation of human intelligence processes by machines, and especially computer systems. These processes include learning - the acquisition of information and rules for using the information, reasoning – using rules to reach approximate or definite conclusions, and self-correction. Particular applications of AI are computer vision, natural language processing and self-driving cars. In the field of Artificial Intelligence, a large number of tools has been developed to solve the most difficult problems in computer science. A few of the most general of these methods are: Search and Optimization, Fuzzy Logic, Probabilistic method for uncertain reasoning, Classifiers and statistical learning methods and Artificial Neural Networks.

1.2 Machine Learning

Machine learning (ML), which is a subset of Artificial Intelligence is a category of algorithms and statistical models that computer systems use to perform specific tasks. This allows computer systems to be more accurate in predicting outcomes without being explicitly programmed, relying on patterns they recognize in data they process. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output, while updating outputs as new data becomes available. These algorithms are based on sample data, known as training data which comprises of a set of inputs and their desired outputs. Computer systems search through this data looking for patterns and they adjust program actions accordingly. A few of the most commonly used machine learning algorithms are: Decision trees, k-Means Clustering, k-Nearest Neighbour and Reinforcement learning.

1.3 Deep Learning

Deep Learning is a subset of machine learning and involves learning based on artificial neural networks. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labelled data and neural network architectures that include many layers, such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks. While traditional neural networks only contain 2-3 hidden layers, deep neural networks can have as many as 150. Some of the most common application of Deep Learning include self-driving cars, image recognition and market price forecasting. Although deep learning was first

introduced in the 1980s, it has only recently become so useful, due to the fact that it requires large amounts of data. For example, driverless car development requires millions of images and thousands of hours of video sequence. Furthermore, deep learning also requires substantial computing power, such as high-performance GPUs which enable reducing training time for a deep learning network from weeks to hours, or even less.

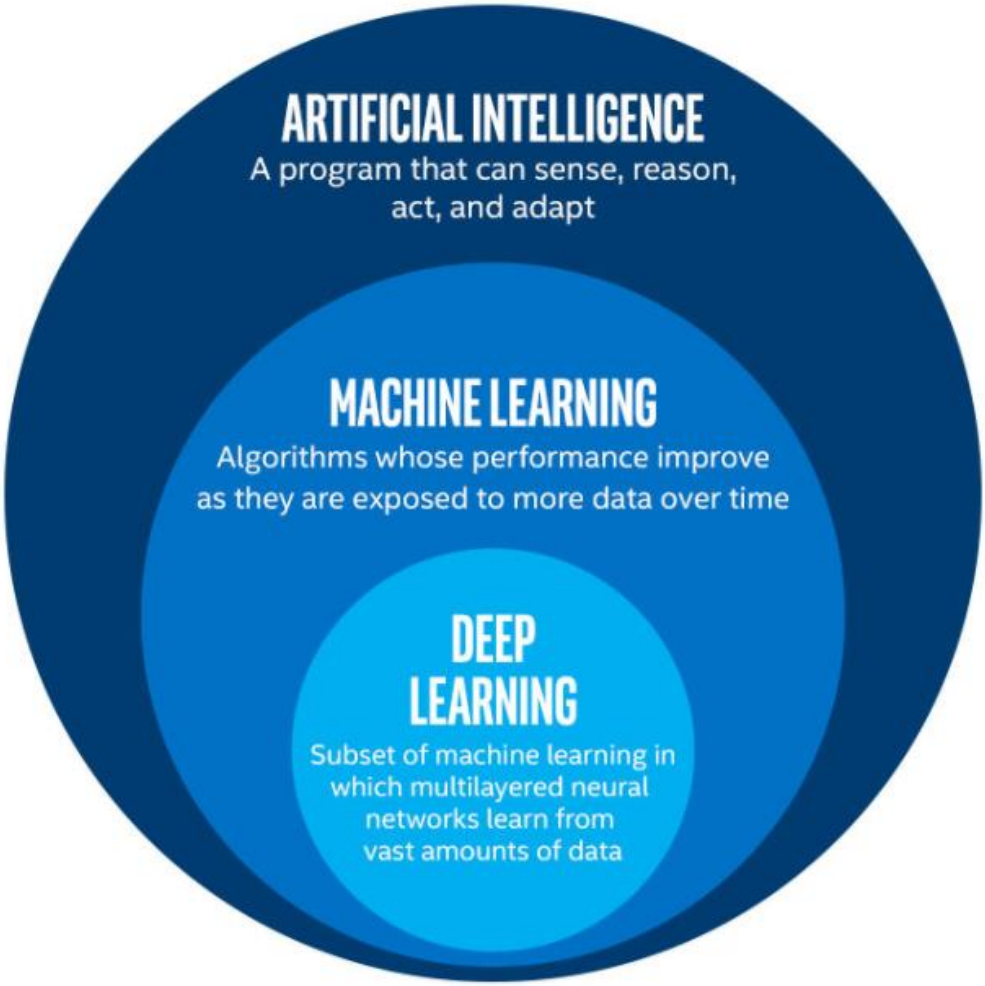


Figure 1: Artificial Intelligence, Machine Learning and Deep Learning areas

1.4 Object Detection

Object detection, which is a subset of object recognition, is the process of finding instances of objects of a certain class in images, such as animals, humans or cars. It has a lot of applications in various areas of computer vision tasks, such as face detection, video surveillance, image retrieval and tracking movement of objects.

Its basic concept is based on the fact that every object class has its own unique features that help in classifying this particular class. For example, all circles are round, which means that when looking for circles, objects that have a constant distance from a fixed point (center) are sought in an image. At the same way, squares have perpendicular corners with equal side

lengths. Therefore, when attempting to detect square objects, such features are searched. Similarly, in face detection, eyes, nose, lips can be found through extracting unique features such as skin color or distance between eyes.

1.5 Object Recognition

Object recognition is a field of computer vision technology which focuses on finding and identifying objects in an image or a video sequence. When humans look at a certain image or video, they can easily spot people, objects, scenes or other visual details. The goal of object recognition is to teach computers to do what comes naturally to humans: to gain a level of understanding of what the image contains.

This concept, which has been introduced for more than half a decade has recently become one of the most exciting fields in Computer Vision and Artificial Intelligence. Considering the development of Convolutional Neural Networks architectures, backed by big training data and advanced computing technology have enabled computers to immediately recognize all objects in a scene. This ability has many applications, including automated vehicles, cancer detection, face detection, etc. It does not only focus on identifying objects in images, but also localizing them. This means that the exact location of a certain object in an image is defined. This allows for multiple objects to be identified and located within the same image, such as identifying multiple pedestrians and stop signs in driverless cars.

1.6 Object Recognition Techniques

Most methods that are used in the field of object recognition and object detection fall into machine learning or deep learning – based approaches. Both techniques learn to identify objects in images, however; they differ in their execution. For machine learning approaches, classification methods are implemented in order to define special features of objects. On the other hand, deep learning-based methods include techniques which are able to end-to-end detect objects while specifically defining features and mostly depend on convolutional neural networks (CNNs).

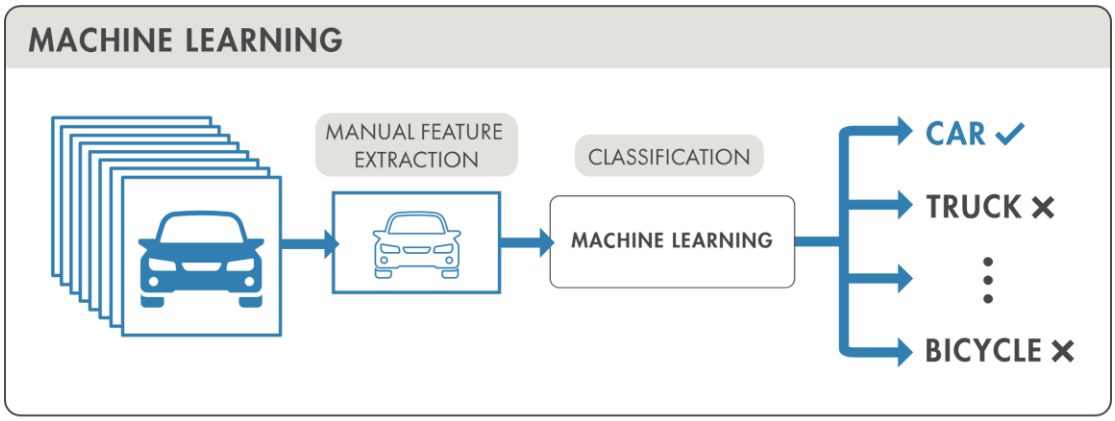
1.6.1 Object Recognition using Machine Learning

Machine learning techniques for object recognition are also popular and offer different approaches than deep learning. Specifically, to perform object recognition with a standard machine learning approach, it is important to have a collection of images or video and have all the relevant features selected in each image. For example, a feature extraction algorithm might extract edge or corner features that can be used to differentiate between classes in the given data. These extracted features are added to the machine learning model, which will learn to separate these features into their distinct categories by performing classification using algorithms such as support vector machine (SVM), Nearest Neighbour or Naive Bayes. This information will be used when analysing and classifying new objects. There is a variety of machine learning algorithms and feature extraction methods, which offer many combinations to create an accurate object recognition model, which can achieve accurate results with minimal data. Furthermore, machine learning methods offer the flexibility to choose the best combination of features and classifiers for learning.

1.6.2 Object Recognition using Deep Learning

Deep learning techniques, such as CNNs are used to automatically extract features in order to identify an object. For example, a CNN can learn to identify differences between cats and dogs by analysing thousands of training images and learning the features that distinguish cats from dogs. In order to perform object recognition using a deep learning model, it is mandatory that this model is trained with a dataset. The model can either be trained from scratch, or a pretrained deep learning model can be used. In the first case, a very large dataset should be gathered, and the architecture of the model should be designed. More specifically, a very wide set of labelled data should be considered in order to include a range of images with different sizes, poses and types of the object. Aside from that, the architecture components of the network, such as weights and layers should be set. This approach can produce impressive results, but it requires large amount of data. The second approach involves fine-tuning of a pre-trained model, so new data containing unknown classes are fed to an existing network. This method is much less time-consuming.

Choosing between machine learning and deep learning- based approaches depends highly on the problem to be solved and on the available computational power. In many cases, machine learning can be an effective technique especially when the features or characteristics of the image are the best to distinguish objects of different classes. Furthermore, choosing a deep learning approach depends on the number of labelled training images and on the availability of computational power, since substantial power helps to decrease significantly the training time.



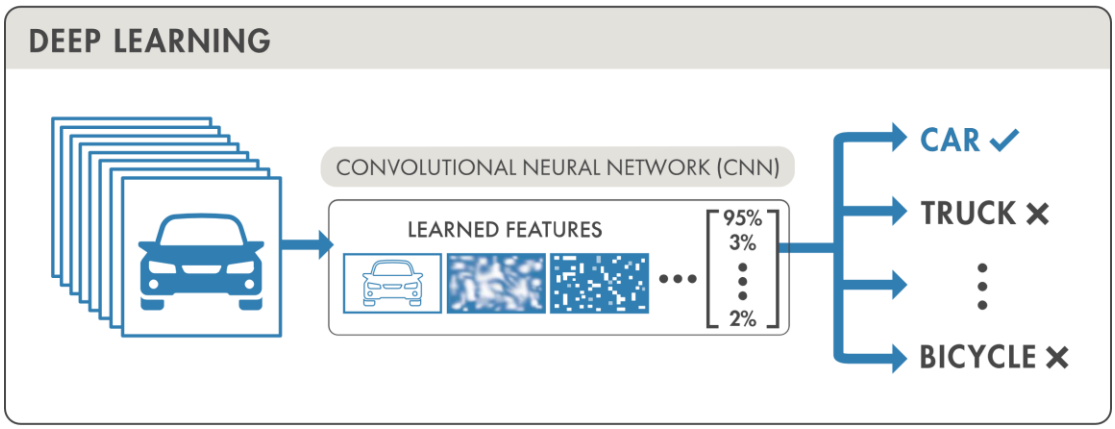


Figure 2: Machine Learning and Deep Learning pipelines for Object Recognition tasks

1.7 Impact of Machine Learning and Object Detection in Manufacturing Industry

The recent advances in Artificial Intelligence which have improved significantly computer vision have surely a great impact on almost every industry. Manufacturing is without a doubt an industry in which significant progress has been noted due to the improvement of object detection algorithms. That is because these advances have given machines "the ability to see", which is a concept that companies, factories, etc have been missing until very recently. This would make factory robots more capable and able to interact with – and take instructions from humans. It is, therefore, expected that the whole industry will change drastically, since a lot of manufacturing processes can now be completely automated. Below are mentioned some of the main cases that computer vision, and particularly object recognition can be implemented.

1.7.1 Quality Management

Quality control and analysis are integral parts of every industrial process and are still highly reliable on human visual inspection. This makes it a difficult task, since human vision must constantly adapt to changing conditions and products and distinguish parts which do not match the quality standards. With the introduction of computer vision, this process can be completely automated with incredible speed, since faulty parts can be easily separated from good parts. Particularly, machine vision tools have been developed that can find microscopic defects at resolutions well beyond human vision, using machine learning algorithms trained on sample images. For example, computers can learn to identify defects, such as scratches, cracks or bad assemblies. This is a very useful solution, especially for large manufacturing industries, where a massive quantity of products has to be tested and time is a very valuable commodity for the business.

1.7.2 Inventory Management

Inventory management is a time-consuming task, which is, however, crucial for the smooth operation of a company. Inefficient inventory management can be harmful for an industry, in terms of both capital and time. Computer vision enables manufacturers to automatically count and localize every stored or outgoing product through object detection or identify misplaced items, therefore, improving significantly the accuracy of inventory management. This helps businesses to avoid wasting money in inaccurate orders due to ordering the wrong type or quantity of products, as well as having an accurate overview of the current stored inventories. Furthermore, object recognition can be applied by companies as a tool for automating content organization, as well as in accelerating content retrieval.

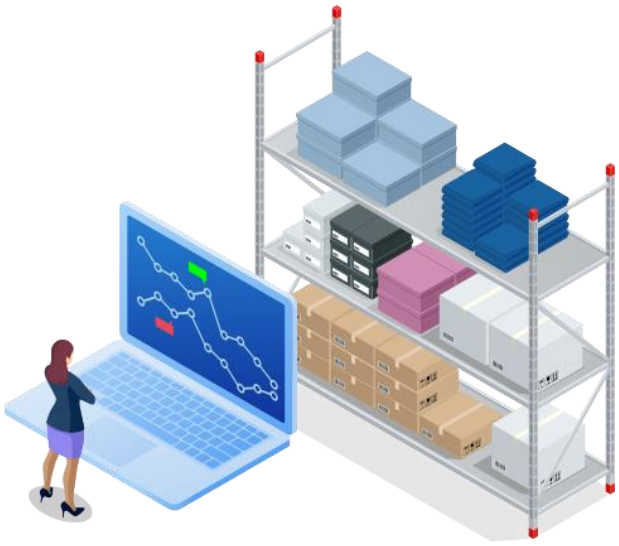


Figure 3: Application of Object Recognition on inventory management

1.7.3 Assembly Line

Today, assembly lines are almost fully automated, even for the most complex products, such as cars. However, every movement of a robotic arm depends on the way it has been programmed to function. By introducing computer vision to the assembly line, machines are able to identify and localize components accurately, therefore eliminating error. This way, robots are taught to autonomously locate, grasp and place different components accurately. Therefore, assembly line is much more efficient and flexible.

Figure 4: Application of Object Recognition on assembly line

2. Artificial Neural Networks – Preliminaries

An artificial neural network (ANN) is a computational system which is inspired by the biological neural networks that make up human brains. Such systems have the ability of "learning" by considering examples. For example, in image recognition they might learn to identify images that contain cats by considering example images that have been labelled as "cat" or "no cat". They do this by automatically extracting certain identifying features from the training material that they process. They are excellent tools for finding patterns which are far too complex and numerous for a human to extract.

These networks consist of a set of connected and parametrized computational nodes, called neurons, which intend to replicate the function of a human neuron. These nodes receive an input signal and produce and output signal. Neurons are organized in three types of layers:

- **Input layer**: brings the initial data into the system for further processing by subsequent layers of artificial neurons.
- **Hidden layer:** a layer in between input and output layers, where neurons take in weighted inputs and produce an output.
- **Output Layer:** the last layer of neurons that produces the desired outputs for the system.

Each neuron receives each output of the neurons of the previous layer as input. Each single input is multiplied by a weight, w_{ij}, where i indicates the number of the input and j the number of the neuron. The value of the weight increases or decreases the strength of the signal at a specific connection. After being multiplied by the respective weights, the inputs are all summed and feed a function inside the neuron, called "activation function".

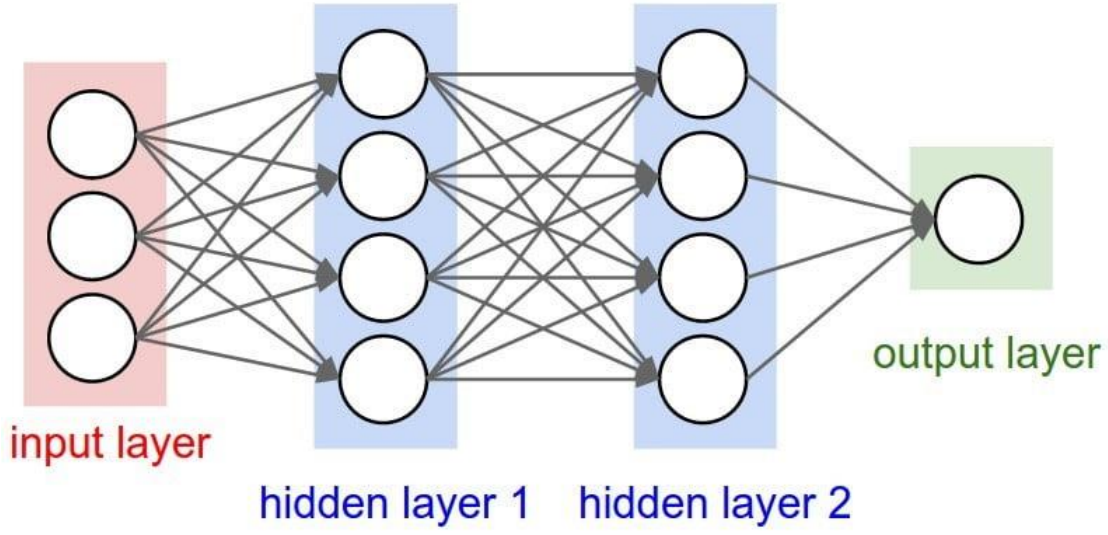


Figure 5: Basic architecture of an Artificial Neural Network

2.1 Activation Function

In artificial neural networks, the activation function of a node defines the output of that node given an input or a set of inputs. The output of this function is the total output of the neuron, which will be an input for the neurons of the following layer. The main role of an activation function is to decide whether a neuron should be activated or not, depending on its input. This is inspired by biological neurons. There are several activation functions, as each on has different results in different problems. Therefore, the choice of different activation functions is mostly dependent on the architecture of the network and the results one obtains when using them. An activation function may be linear or non-linear, however; a network with linear activation function will only be able to learn linear problems, since summing all layers in the network will give another linear function. Non-linear activation functions are preferred in more complex problems. Some of the desirable properties of an activation function are continuous differentiability, finite range and smoothness.

Some of the most widely used activation functions are the Sigmoid Function, the *Hyperbolic Tangent (tanh)*, the *Rectified linear Unit (ReLU)* and the *Leaky Rectified Linear Unit (LReLU).* All the above functions share the same basic behaviour but each one of them has best results in some particular problems. For example, in the area of Object Recognition, the ReLU functions is mostly preferred. In the following image the graphs of the functions are shown:

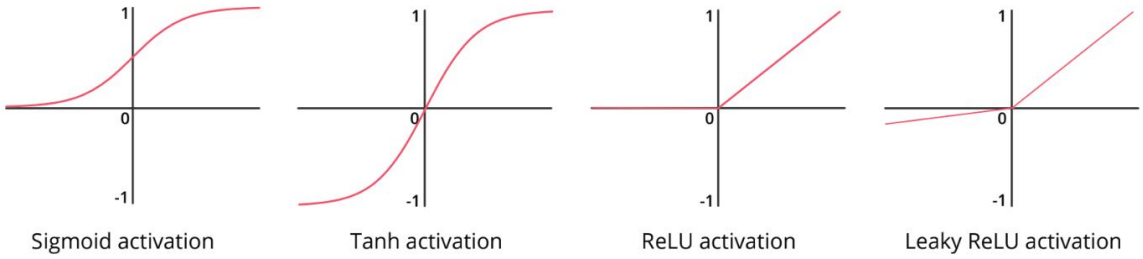


Figure 6: Basic activation functions

The sigmoid function maps the input to (0,1) and is defined as following:

$$\sigma_{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

It is one of the most widely used activation functions as it has the properties of smoothness and continuous differentiability. However, in cases where lower layers of the neural network have gradients near to 0, it may lead to slow convergence and instability. This issue is referred to as *vanishing gradient* problem, as the gradient becomes so small in the earlier layers of a deep neural network that it barely affects the weights of the earlier layers, thus failing to optimize the initial weights. It has a characteristic “S”-shaped graph and is widely used in binary classification.

The tanh function is very similar to the sigmoid function with the main difference being that it maps the input to the (-1,1) range. Like the sigmoid, it also has the same problem of *vanishing gradient*. It is defined as following:

$$\sigma_{tanh}(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

The ReLU function is among the most widely used and it has the following simple form:

$$\sigma_{ReLU}(x) = \max(0, x)$$

One advantage of the ReLU function is that, unlike, sigmoid and tanh functions it does not have convergence problems, since the gradient is either 0 or 1. Furthermore, if the input of the neuron is negative, it automatically converts it to zero, therefore, another advantage is that it does not activate all neurons. This makes the network more efficient and easier for computation. However, the ReLU function is dealt with the *dying ReLU* problem. This is when the ReLU activation function always outputs zero value for any input, which means that there is no discrimination between different inputs. Once in this state, it is unlikely to recover, as the gradient will also be zero, therefore, weights will not be altered. It should be noted that ReLU functions can only be used in the hidden layers of the neural network.

LReLU function is another version of the ReLU function, and is defined as following:

$$\sigma_{LReLU}(x) = \max(0, x) - a\, max(0, -x)$$

The main difference with ReLU is that LReLU allows a small gradient when the input is negative. This solves the dying problem which ReLU is dealt with, therefore, makes the system more stable.

2.2 Bias

It must be noted that each neuron includes an additional important parameter, called “bias”, or “threshold”. Just as the weights, bias is a learnable parameter and its value determines, whether the activation output from a neuron is going to be propagated forward through the network. Therefore, it determines whether or not or by how much a neuron will operate by moving the entire activation function to the left or right, upward or downwards, in order to generate the required output values. Therefore, the “bias” increases the flexibility of the neural network to fit the given data.

Instead of the weighted sum of the inputs from the previous layer being passed to the activation function, the bias term is also added to that sum and then the entire sum feeds the activation function. At the beginning of the training procedure, biases’ values are initialized to random numbers or zeros, and are updated continually during training.

2.3 Model of Artificial Neuron

The mathematical interpretation of a neuron *j* for example is a function f: $\mathbb{R}^N \rightarrow \mathbb{R}$, defined as:

$$f(\boldsymbol{x}; \boldsymbol{w_j}) = \sigma\,(w_{0j} + \sum_{I=1}^{N} w_{ij} x_i + \theta_j)$$

where σ: $\mathbb{R} \rightarrow \mathbb{R}^N$ is the activation function, $\mathbf{w_j} \in \mathbb{R}^{N+1}$ is the vector of weights and θ_j is the threshold term of the neuron.

For clarity reasons, the term w_{oj} may be ignored, and thus write in vector form:

$$f(\boldsymbol{x}; \boldsymbol{w_j}) = \sigma(\boldsymbol{w_j^T x})$$

As it can be seen in the image below, each neuron receives as inputs $\boldsymbol{x} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_N}]$ the outputs of the neurons of the previous layer. Then, these inputs are multiplied by the corresponding weights $\boldsymbol{w_{ij}}$, and then they are all summed. This sum is then passed to the activation function, which produces the final output of the neuron $\boldsymbol{y_j}$.

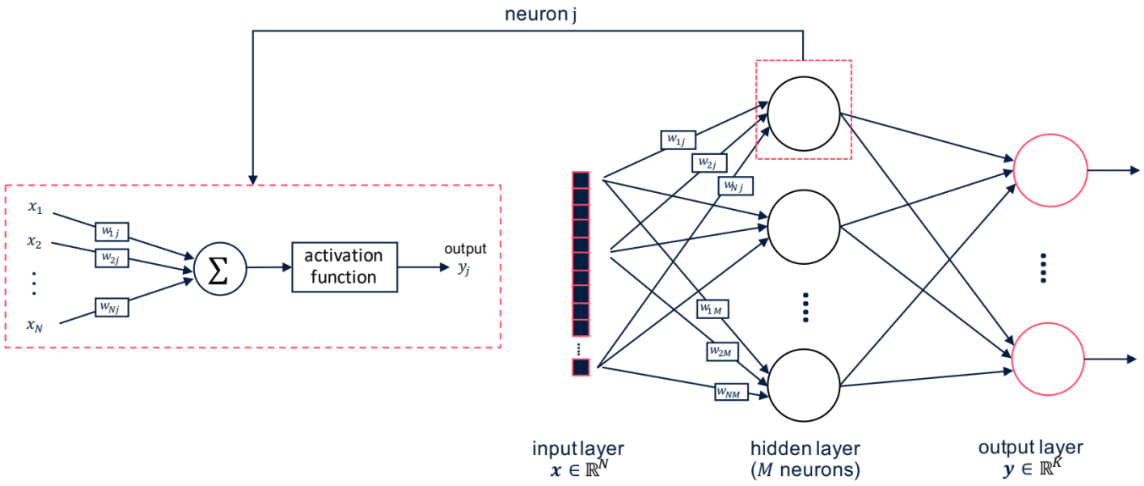


Figure 7: Schematic representation of a typical feed-forward artificial neural network

2.4 Feedforward Neural Networks

A feedforward neural network is an artificial neural network where its connections between nodes do not form a cycle. The information moves only towards the forward direction, from the input nodes, through the hidden nodes and to the output nodes. This means that the output of any layer does not affect the same layer. It is the simplest form of ANNs. Representative examples of such networks are the Single-layer Perceptron and the Multi-layer Perceptron.

2.5 Feedback Neural Networks

A feedback or recurrent neural network (RNN) is a neural network, where its connections can form loops. This means that information flows in both directions, both forward, and backwards. Outputs are fed back to the network, which gives the network a kind of memory, since an output a neuron reached at a time step, *t-1*, affects the output it will reach at a time step, *t*. These systems are dynamic, since their state is changing continuously until they reach an equilibrium. Therefore, compared to feedforward networks, these systems can get very complicated but are very powerful.

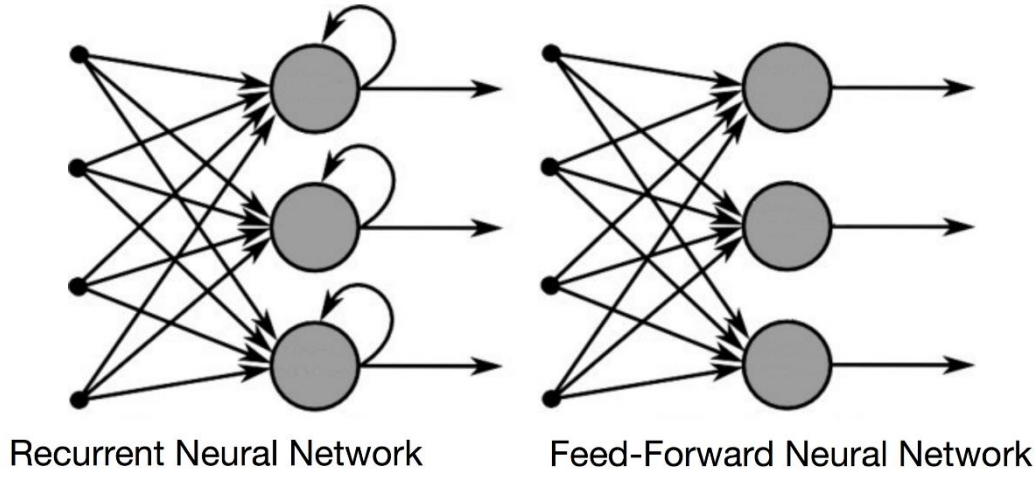


Figure 8: Feedforward and Feedback Neural Networks

2.6 Typical Neural Network Types

2.6.1 Perceptron

Perceptrons are the first and simplest form of artificial neural networks which consists of a single layer of output nodes. The inputs are fed directly to the outputs through being multiplied by the respective weights. The sum of the products of weights and inputs is calculated in each neuron, and if the value is above a predefined threshold (bias) the neuron

is activated and the output is usually assigned to unit value. Otherwise, it is assigned to the deactivated value, which is usually -1 or 0. This is done by using the threshold activation function, which makes the output of each neuron 1 or 0. This special attribute of perceptrons makes such networks ideal for classification tasks, (i.e. decide whether an input belongs to a certain class or not). While perceptrons perform great in some tasks, they also have some weaknesses. Because the output of the neuron is either 0 or 1, a small change in the weights or bias of any single perceptron in the network may cause the output to completely flip (e.g. from 0 to 1). That flip may then cause the behaviour of the rest of the network to change in a very complicated way. For that reason, sigmoid neurons might be used instead of perceptrons. Additionally, perceptrons are only capable of learning linearly separable problems, so multilayer perceptrons were later introduced in order to counter this issue.

2.6.2 Multilayer Perceptron (MLP)

Multilayer Perceptrons are typical feedforward neural networks composed by one or more hidden layers. Except for the input nodes, contrarily to perceptrons, each node uses a nonlinear activation function, which enables MLPs to solve problems that are not linearly separable. An MLP with one hidden layer is capable of approximating any continuous function. Their architecture is quite simple, since they are composed of fully connected layers, which means that each node in one layer connects with a certain weight w_{ij} in the following layer. Multilayer perceptrons are often applied to supervised learning problems, which means that they are trained on a set of ideal input-output pairs using backpropagation algorithm. These concepts will be elaborated in the following chapters.

2.7 Learning

An integral part of neural networks and of Artificial Intelligence in general, is the concept of learning. This concept is referred to computing the best values for all dependent-free variables in a system through a learning algorithm, in order to have an ideal behaviour. In the case of neural networks, this means calculating the values of weights and thresholds. There are two basic types of learning: supervised learning and unsupervised learning.

2.7.1 Supervised Learning

Supervised learning requires some type of “supervision” that indicates which is the ideal behaviour of the system and corrects it when it performs mistakes. The desired behaviour is determined by a set of ideal inputs-outputs that define which is the ideal output of the system when it is fed by a certain input. During the procedure of training, the real output of the network differs from the desired one; the difference of these two outputs is the error of the ANN, and it is used by the training algorithm in order to define new values for weights and biases.

Training algorithms are usually repetitive tasks, which means that the sets of ideal input-output are provided consecutively to the network, and each time the error is calculated, which indicates the change of weights and biases. This procedure is terminated either when

the algorithm reaches a certain number of iterations or when the error drops down of a desired threshold. Before training takes place, the dependent-free parameters of the systems (biases and weights) should be initialized to some values, which are usually considered as small positive numbers.

2.7.2 Unsupervised learning

Unsupervised learning, on the other hand, is a class of machine learning techniques to find patterns in data. Unlike supervised learning, training set does not include any labels, which means that the input variables are not provided with any corresponding outputs. Therefore, algorithms must detect certain structures in the data. Success in such kind of networks is determined by whether the network is able to reduce an associated cost function.

Figure 9: Unsupervised and Supervised Learning

2.8 Objective Function

In all problems that involve optimization an objective or loss or cost function is a function of inputs that outputs a real number representing a "cost", which is associated with the certain input. An objective function is a measure of how good a prediction model is in terms of being able to predict the expected outcome. In an optimization problem, the objective is to minimize the loss function.

The learning problem for a neural network is formulated as searching for a set of weights $\{W^{(l)}\}_{l=2}^{L}$ that minimizes a given loss function $E : \mathbb{R}^{K \times K} \rightarrow \mathbb{R}$, which is a function that measures the inconsistency between the output predicted by the neural network (y) and the true value $\widehat{(y)}$. The learning problem can also be formulated as searching for a function that best maps a set of inputs to their correct output. When training a neural network, the objective is to minimize the objective function and bring output as close as possible to the

correct value, which is achieved by using gradient descent algorithm. An example of a common loss function used in neural networks is the L2-norm, which calculates the squared difference between the output predicted and the actual value:

$$E(y, \hat{y}) = \sum_{i=1}^{K} (y_i - \hat{y}_i)^2$$

Some other loss functions are the Mean Absolute Error, Smooth Mean Absolute Error and the Log-Cosh Loss function.

In general, it is not possible to find a closed form expression for the minima of the loss function of a neural network, so the gradient of the loss function is calculated.

2.9 Back Propagation Algorithm

Backpropagation of error algorithm is one of the most widely used training algorithms in the case of neural networks. It is capable of training feedforward networks of any size and number of layers. Its objective is to calculate the values of weights and biases, which are often referred to as learnable parameters of an ANN, in order to minimize the loss function of the neural network. Given a neural network architecture and a loss function, back-propagation calculates the gradient of the loss function, ∇E with respect to all weights in order to achieve a local minimum using dynamic programming. This is a mathematical process called gradient descent, which is a first-order iterative optimization algorithm for finding the minimum of a function. That is, given a neural network and a loss function, back-propagation algorithm propagates the loss at the output layer backward, so that the gradient at all hidden layers can be calculated using the chain rule in order to adjust the weights at each neuron, as described below:

Assuming the output of the j-neuron is o_j, then:

$$o_j = f(net_j) \text{ then } net_j = \sum_j^i w_{ij} o_i + \theta_j$$

Then the error produced in the output of the neuron j is:

$$e_j = t_j - o_j$$

Summing the errors of all the neurons in the output layer:

$$E_P = \frac{1}{2} \sum_{j-output} (t_{pj} - o_{pj})^2$$

The weights and biases are updated as following:

$$\Delta_p w_{ji} = -a(\frac{\partial E_p}{\partial w_{ji}})$$

$$\Delta_p \theta_j = -a(\frac{\partial E_p}{\partial \theta_j})$$

where *a* describes the learning rate parameter.

The calculation of the partial derivative of the objective function with respect to a weight w_{ij} is done, using the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

Similarly, the partial derivative of the object function with respect to the biases is calculated.

As it can be seen from the image below, the loss function is calculated by considering the squared difference between the target and predicted output and summing these values for all output neurons. Then, the derivative of the loss function with respect to all weights is calculated using the chain rule, and by propagating the error backwards, the values of all weights are adjusted accordingly.

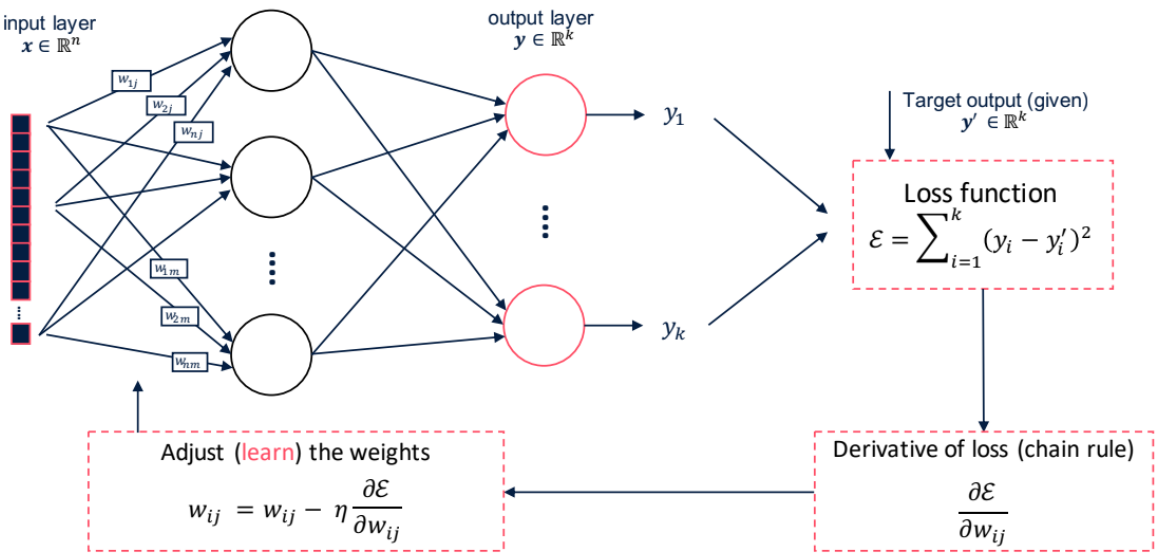


Figure 10: Schematic Representation of Backpropagation Algorithm

2.10 Optimization of Backpropagation Algorithm

2.10.1 Batch Normalization

Batch normalization is a technique applied in artificial neural networks in order to improve their efficiency and stability. While training a neural network, the values of the weights of the layers constantly change, therefore, the distribution of the inputs also changes. This makes training slower, especially in cases of very deep networks. This problem is known as *internal covariate shift*. More specifically, as long as the statistical distribution of the input keeps changing after some iterations, the hidden layers will keep trying to adapt to the new distribution, therefore, making convergence slower. So, the batch normalization algorithm is proposed in order to solve that issue. It involves a normalization of the inputs to a layer with zero mean value and unity standard deviation. This makes each layer in the network to learn faster, and also independently of the other layers. Furthermore, Batch Normalization makes neurons work in their linear regions of their activation functions, which improves learning performance. It also prevents the *vanishing gradient problem*, which sigmoid and tanh functions are dealt with.

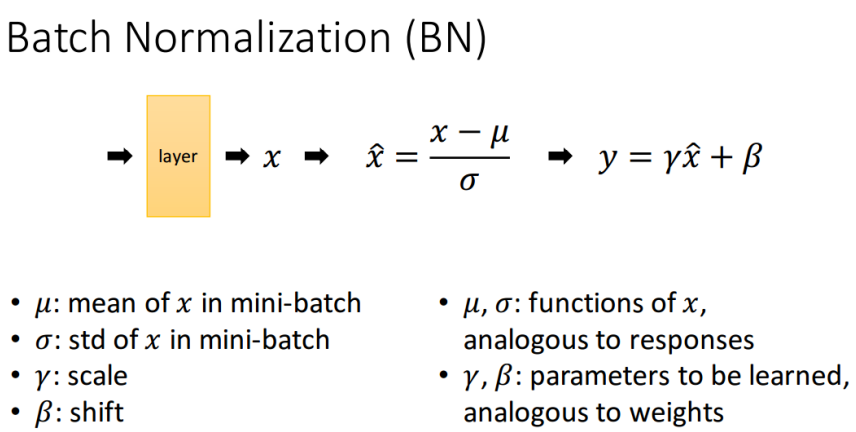


Figure 11: Batch Normalization (BN) applied to the inputs

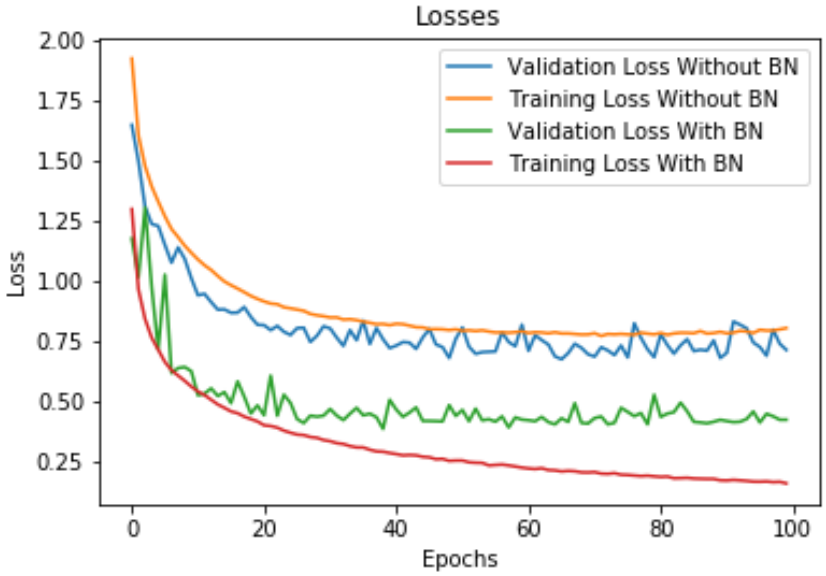


Figure 12: Average loss with and without Batch Normalization

2.10.2 Adaptive Learning Rate

Learning rate in neural networks in artificial neural networks is defined as a parameter which indicates how fast the network adapts the values of its weights, with respect to minimizing the loss function. The lower the learning rate is, the slower it will take the ANN to converge. However, if its value gets too great, the network might overshoot the minimum of the loss function, and it might cause divergence. The weight update formula is as following:

$$w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

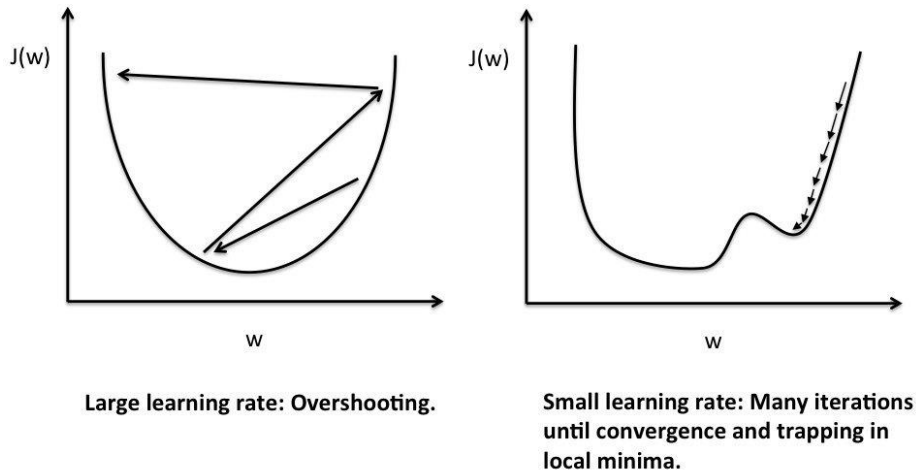


Figure 13: Large and small learning rate issues

2.10.3 Momentum Rate

In artificial neural networks, the error function comprises of local minimums, which the system can get stuck to, thinking it has reached the global minimum. To overcome such situations, a momentum term is considered in the objective function, which is a value between 0 and 1 that increases the size of the steps that are taken towards the minimum. Therefore, it helps the system jump from local minimums, and move towards the global minimum. A large momentum rate implies faster convergence, but the global minimum point might be skipped. In cases that the learning rate is large, momentum rate should be kept at lower values.

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}} + \gamma \Delta w_{ij}^{t-1}$$

,where η is the learning rate and γ is the momentum rate

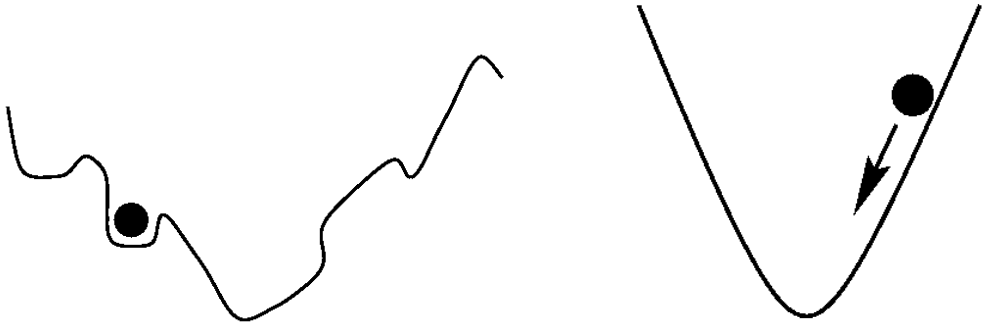


Figure 14: Network stuck into local minima

2.11 Issues in Artificial Neural Networks

Thanks to their huge number of parameters, artificial neural networks have a lot of freedom and can fit almost any complex dataset. This ability has allowed them to find applications in many areas, in which it has been difficult to make progress, such as image recognition, natural language processing, etc. However, sometimes, the complexity of these networks may become a potential weakness, since it can lead to overfitting or underfitting.

2.11.1 Overfitting

This situation occurs when the neural network is so closely fitted to the training set that it is difficult to generalize and make predictions for new, unseen data. In practice, detecting that a trained model is overfitting is a difficult task, so it is necessary that some steps should be followed during training. Specifically, it is advisable that a dataset should be divided into three parts – training set, cross-validation set and test set. The model learns by only

considering data from the first part, while cross-validation set is used to track progress of training and optimize the model. Furthermore, at the end of the training process, the test set is used, in order to evaluate the performance of the trained network.

Although until recently the most frequently recommended division of dataset would be: 60% training set, 20% validation set and 20% test set, when a dataset comprises of millions of entries, these proportions are no longer appropriate. In short, everything depends on the total size of the dataset, and in cases of millions of data, it could even be better to divide the set in 98/1/1 ratio.

2.11.2 Underfitting

This situation occurs when the trained model can neither fit the training data, nor generalize to new unseen data. Detecting an underfit model is obvious, since it will have poor performance on training data, and it is, of course, an unsuitable.

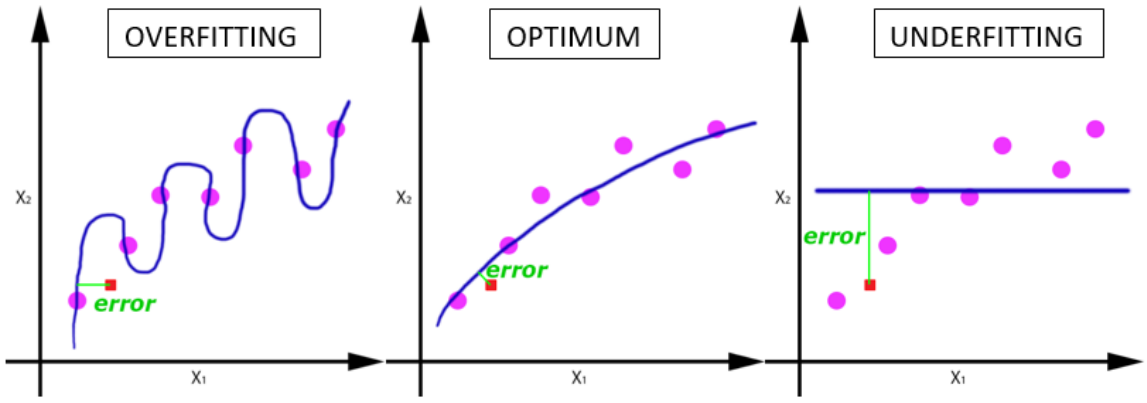


Figure 15: Description of Overfitted, Optimum and Underfitted model

3. Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specialized kind of deep feed-forward neural networks, mostly used in computer vision. These networks are very successful in processing data that have a known grid-like topology, such as time series data which can be thought as a 1-D grid samples at regular time intervals, or images, which can be thought as a 2-D grid of pixels. The term “convolutional” refers to the mathematical operation, convolution, which is employed to these kinds of networks. The main advantage of such networks is their ability to decrease the computational power required to process data through dimensionality reduction.

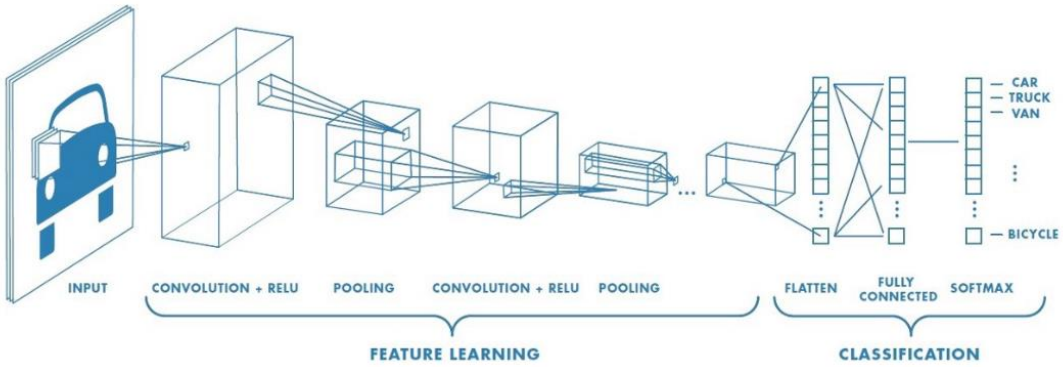


Figure 16*: Con*volutional neural network pipeline

3.1 Design

A convolutional neural network consists of input and output layers, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, ReLU layers, pooling layers, fully connected layers and normalization layers. The special concepts of these layers will be analytically discussed in the following chapters.

3.2 Convolution

In mathematics, convolution is a mathematical operation on two functions (say f and g) that produces a third function which expresses how the shape of one is modified by the other. Specifically, it is an integral that expresses the amount of overlap of function g, as it is shifted over function f. It is defined as following:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

In deep learning applications, the input (function *f*) is a multidimensional array and the second argument (function *g*), usually referred to as the kernel is a multidimensional array of parameters that are adapted by the learning algorithm.

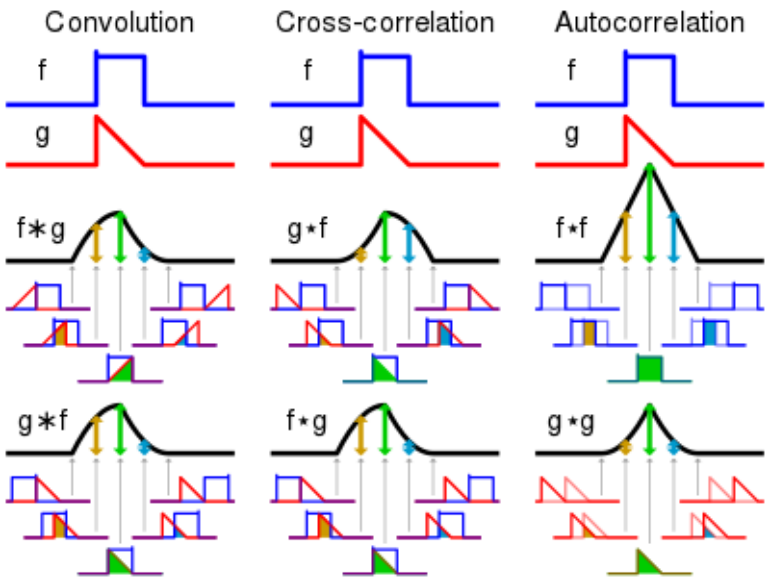


Figure 17: Convolution Operation between two functions

3.3 Convolutional Neural Networks in Deep Learning

The architecture of convolutional neural networks used in Deep Learning and specifically in Computer Vision is in analogy to the connectivity pattern of human brains and is inspired by the function of the Visual Cortex. More specifically, individual neurons respond to stimuli only in a restricted region of the visual field, known as the Receptive Field. The composition of all these overlapping fields, results into covering the entire visual area. Similarly, hidden neurons inside the convolutional neural network, or otherwise referred to as features, are activated only by a smaller region of the input. This means that the fundamental difference between a fully connected layer and a specialized convolutional layer, is that the fully connected layer learns global patterns in its global input space, whereas convolutional layers learn local patterns in small windows.

The goal of CNNs is to reduce the size of the input into a from which is easier for the network to process, without losing any features which are critical for getting the right prediction. This is very important, especially when massive datasets, such as in image recognition, are involved, where otherwise, it would be extremely computationally expensive for an MLP neural network to process an image. For example, a single neuron in an MLP which receives an image as input with size 64 x 64 x 3 (width, height, channel), will have 12,288 weights. Furthermore, a distinguishing feature of CNNs is that many neurons share the same weights, which reduces memory footprints, since a single vector of weights is used across all receptive fields sharing that vector.

In a regular artificial neural network, each neuron receives inputs from all neurons in the previous layer. Contrarily, in a convolutional neural network, instead of neurons receiving all inputs, they receive only a small patch of the input, and then apply the convolution operation with a weighted matrix, which will be referred to as a *kernel* by performing matrix multiplication and then pass the result to an activation function. Therefore, convolutional layers consider the spatial locality of their input. This means that convolutional layers learn spatial hierarchies of patterns by preserving spatial relationships. For example, a first convolutional layer maybe responsible for capturing low-level features such as edges, colour, gradient, orientation, etc. The second convolutional layer may be responsible for learning patterns composed of the basic elements that were learnt in the previous layer, and so on until the network can learn very complex patterns. This allows convolutional neural networks to efficiently learn increasingly complex and abstract visual content.

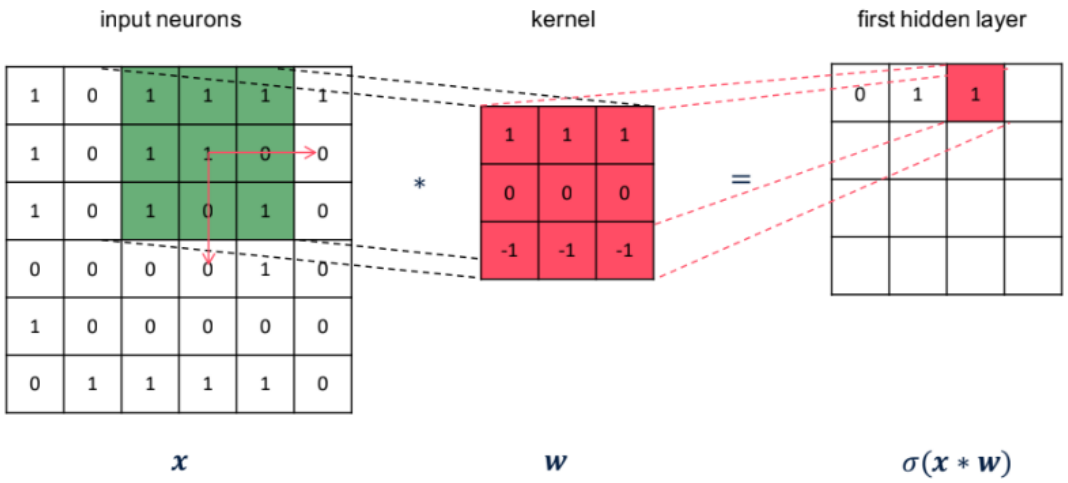


Figure 18: Convolutional layer takes input x and applies convolution operation with a 3 x 3 kernel, stride = 1 and padding = 0

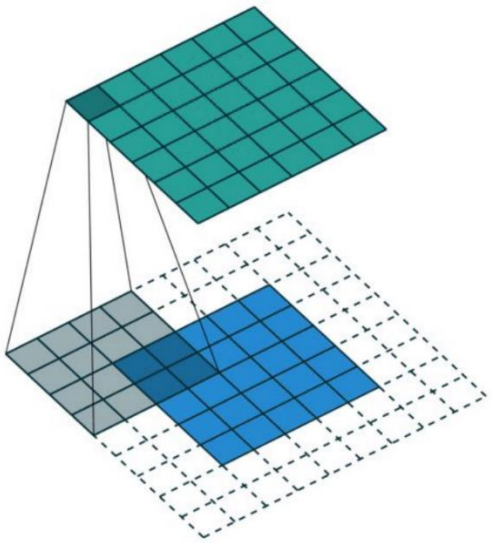


Figure 19: Kernel sliding over the image and saves the result to an array with lower dimension

3.4 Steps in Image Recognition with CNN

The kernel slides over the entire original image in both direction (columns and rows) until the entire image is parsed and saves each result as a separate smaller image. The size of the kernel's shift is defined by two parameters: *stride* and *padding*, with stride representing the movement along the two directions and padding zero-pads the border of the input. There are two types of convolutional operation –in the first one, the dimension of the convolved feature is reduced compared to the input, and in the second one the dimension of the convolved feature is either increased or remaining the same. In the first one, *Valid Padding* is applied, whereas in the latter one, *Same Padding*. Therefore, the convolution operation has three dependent-free parameters: stride, padding and the size of the kernel matrix. After this procedure, the original image has been turned into several smaller, equally sized image tiles, which looks like this:

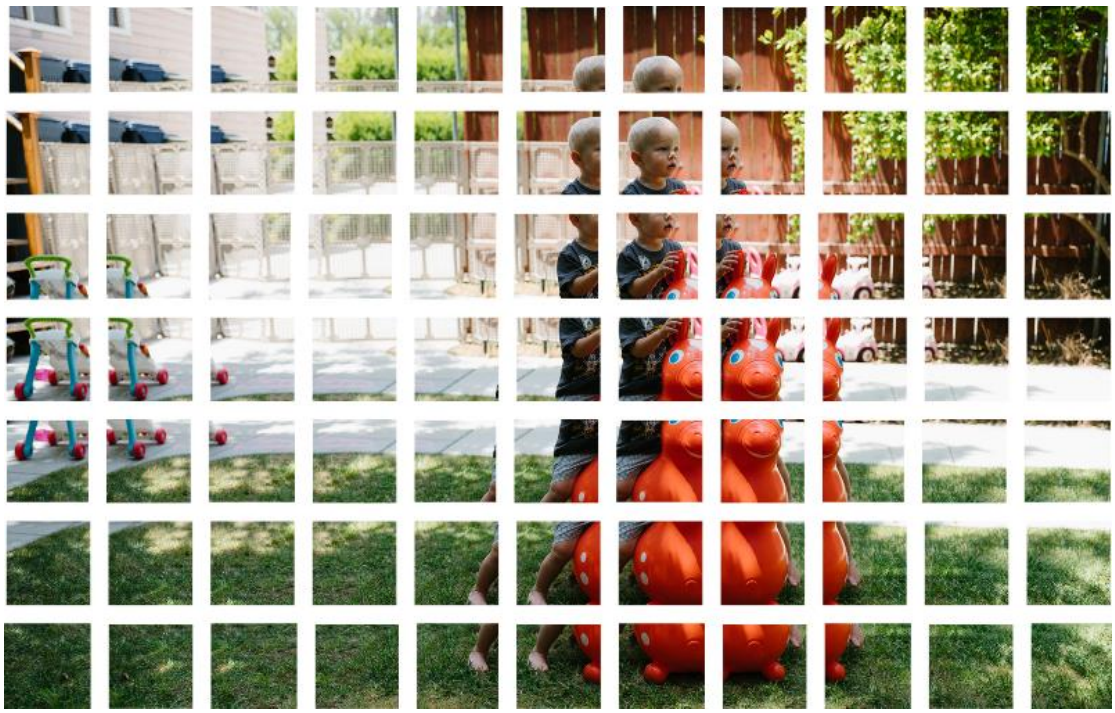


Figure 20*:* Feature map of image in a CNN

At this stage, instead of feeding the entire image to a neural network to perform recognition, every individual image tile is fed to a relatively small neural network, in which some outputs are produced. An important point is that the small neural networks that process all image tiles share the same weights. This means that every image tile is treated equally.

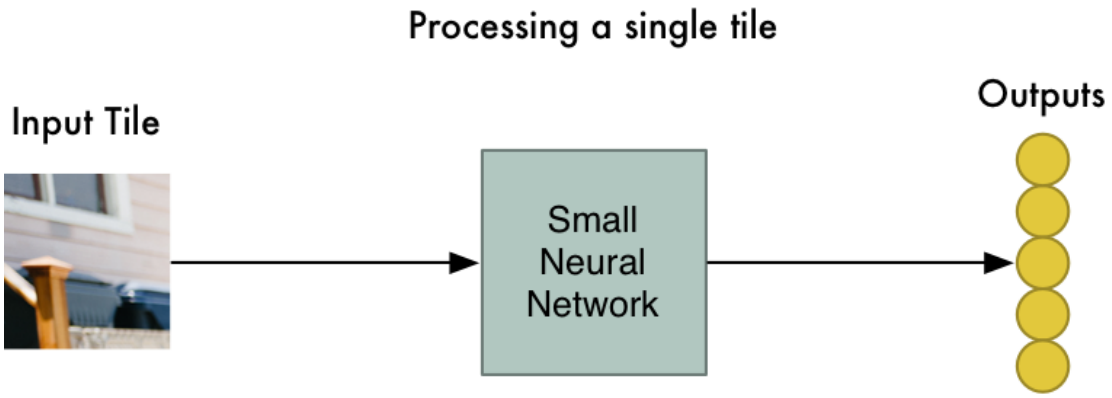


Figure 21: Processing of feature map in a CNN

The results from processing every image tile in the neural network are saved and stored in an output array, just as it would have been if the original image would have been processed. The array that contains the outputs, with each one representing the probability of the tile containing an object, is much smaller than the output array that would have been produced if the whole original image was being processed.

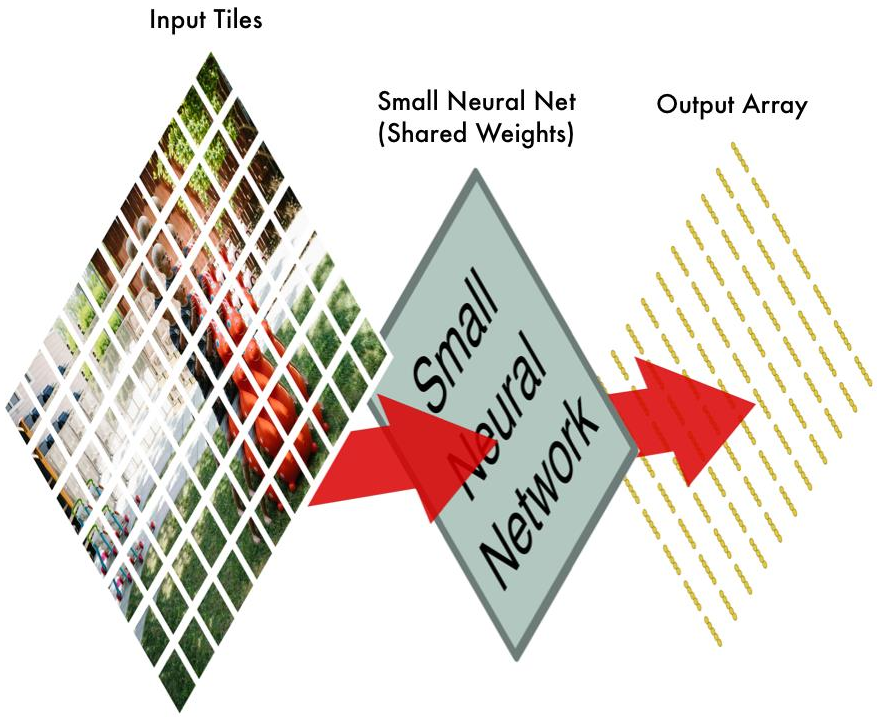


Figure 22: Overall process of an image through a CNN

3.4.1 Pooling layer

Convolutional neural networks usually accompany convolution layers with pooling layers, which are usually applied immediately after the convolutional layers. The basic goal of a pooling layer is to simplify the information collected by the previous layers and create a condensed version of the information contained in them. There are many ways to condense that information, but the most usual one is the max-pooling operation.

However, the size of the output array is still big, so in order to reduce its size, it is down sampled, using an algorithm called max pooling. The basic idea of this algorithm is to segment the output array in a square grid and keep the maximum values from each one. By applying this operation, if something interesting is found in the square grid, the most interesting is being kept to the max pooled array.

In a similar way as the max-pooling operation, the average pooling returns the average of all the values from the grid. Both algorithms are responsible for reducing the spatial size of the convolved feature. However, max pooling has an additional advantage, which is noise suppression. Specifically, it discards all the noisy activations and so performs de-noising. Hence, max pooling performs in general, better than average pooling.

The result of feeding all image tiles into smaller neural networks is an array that maps out which parts of the original image have the highest probability of containing an object. However, that array is still big, so Maxpooling algorithm is used, in order to reduce its size, while keeping the most interesting outputs.

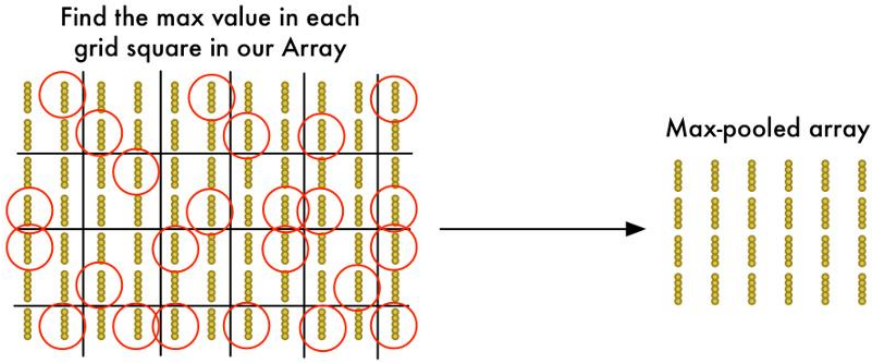


Figure 23: Maxpooling process

After reducing the size of the output array, it is then fed to another neural network. This final neural network will determine whether the image contains an object or not and will produce a final output. This neural network will be referred to as “fully connected” network.

In conclusion, the general pipeline of applying a convolutional neural network in an image is as follows:

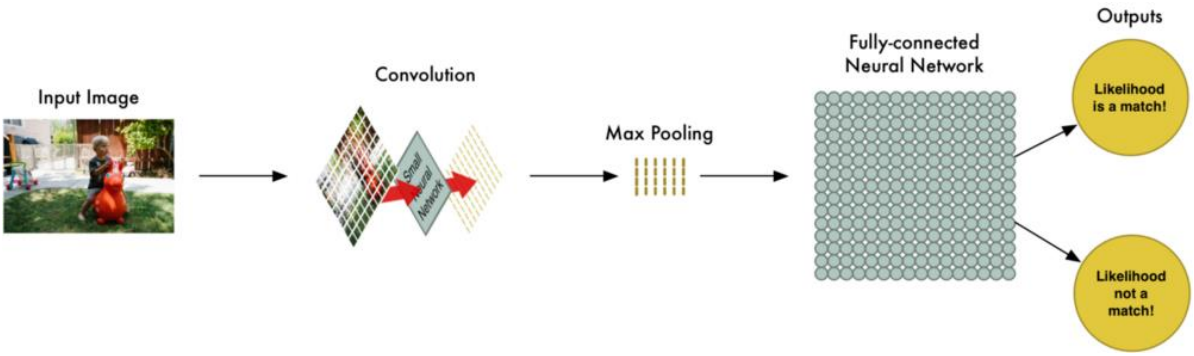


Figure 24: Image processing through a CNN-based object detection system

When designing an architecture for a network the above steps can be combined as many times as required to achieve a good performance, but of course, at the cost of more computational power. The basic idea is to have a large input and continually reduce its size, untill a single result is outputed.

More convolutional layers means that the network can extract more complicated features from the input. With added layers, the network has the ability to adapt to more high-level features as well, therefore understanding the entirety of the given image. For example, considering a problem of identifying cats in images, the first convolutional layer might be able to recognize, ears, the second one to recognize whiskers, and the last one to recognize entire cats, using its previous knowldege.

Of course, there is not a clear and straight-forward method of constructing the perfect architecture for a network. As most of the applications of machine learning, a lot of trial and error is involved, therefore, a lot of experimentation and testing is required.

4. State-of-the-art Object Detection Algorithms

4.1 R-CNN

4.1.1 General

R-CNN [1], which stands for Region-CNN is one of the state-of-the-art convolutional neural networks (CNN) based deep learning object detection algorithms. The main novelty of this approach is that it proposes a method where selective search is used in order to extract 2000 regions from the image, which are called region proposals. Therefore, instead of classifying a large number of regions, just 2000 regions can be used.

These 2000 proposed regions are fed to a convolutional neural network, which produces a feature vector as output, which consists of features extracted from the image. The network that performs the features extraction is called AlexNet, a neural network framework introduced in 2012, which outperformed all existing frameworks at the time (with 15.3% error rates). Its architecture consists of 5 convolutional layers and 3 fully connected layers, in which ReLU activation function is applied. During training, weights are updated as following:

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

, where learning rate starts from 0.01 and is decreased 3 times during training, momentum rate is set to 0.9 and weight decay parameter is 0.0005.

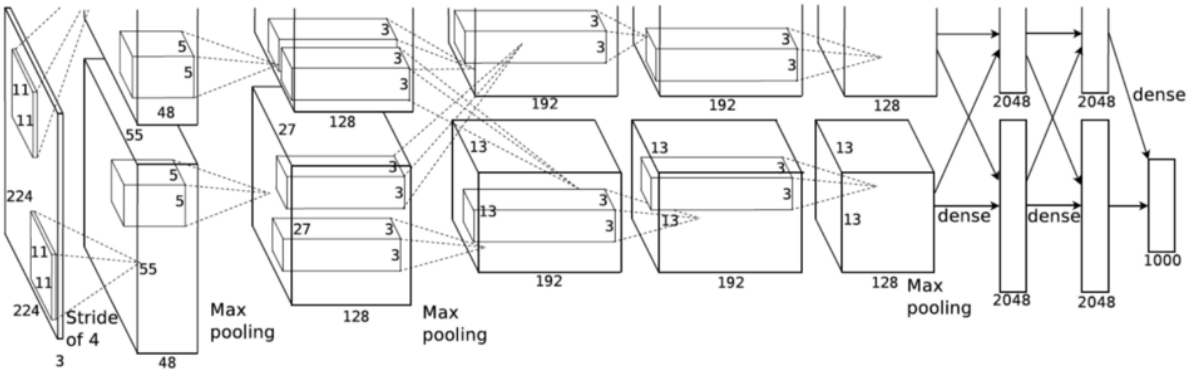


Figure 25: AlexNet Architecture

Following, the output is fed to an SVM (Support Vector Machine) classifier, which finds a hyperplane in an n-dimensional space (where n is the number of features) that distinctly classifies the data points. Therefore, the SVM classifies the presence of objects within that candidate region proposal.

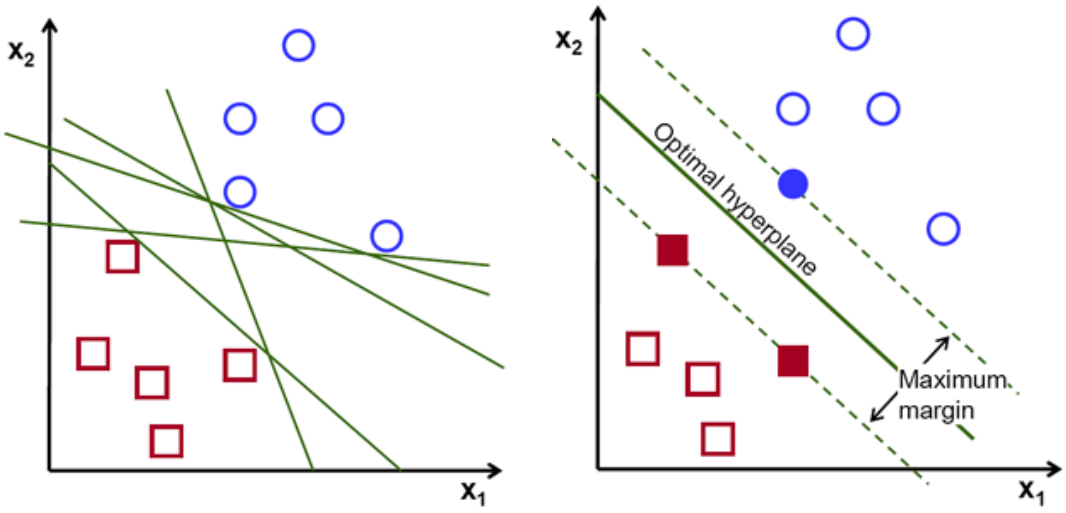


Figure 26: Support Vector Machine Algorithm

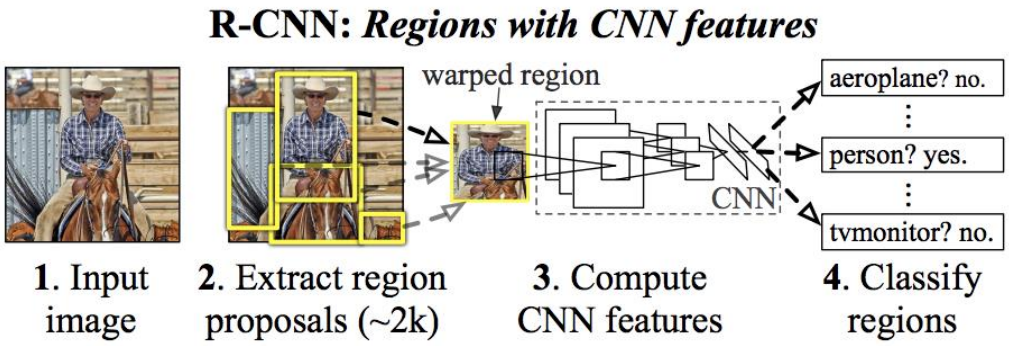


Figure 27: R-CNN pipeline

4.2 Fast R-CNN

4.2.1 General

After R-CNN network being released, some of its drawbacks were solved and so an advanced version of RCNN was produced, called Faster R-CNN [2]. This approach is similar to the one of R-CNN, however, there are some essential differences. Specifically, instead of feeding the 2000 proposed regions to the CNN, the entire input image is fed to a CNN to generate a convolutional feature map. Then, the region proposals are identified from the feature map, warped into squares and by using a Region of Interest (RoI) pooling layer they are reshaped into a fixed size. Following, they are fed to a fully connected layer. From the output RoI feature vector, a softmax layer is used to predict the class and the offset values of the bounding box.

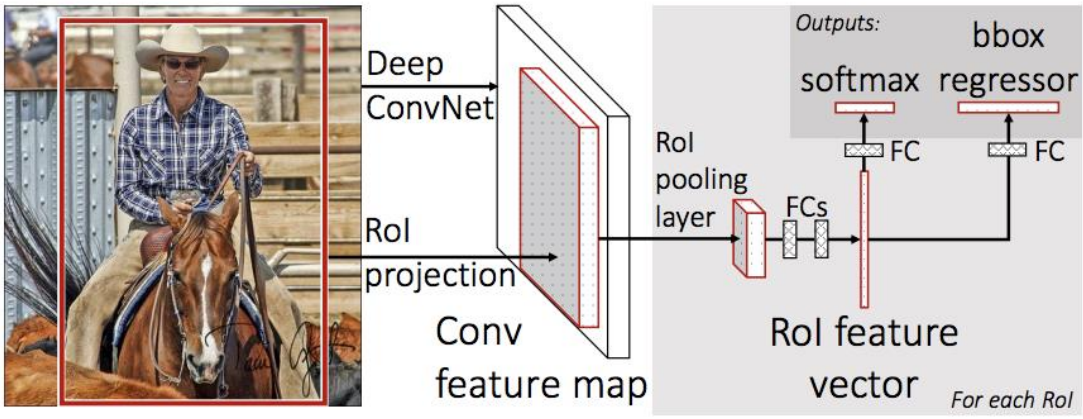


Figure 28: Fast R-CNN Pipeline

The main advantage of Fast-RCNN over R-CNN is that instead of applying the convolution operation to 2000 region proposals every time, the convolution is done only once per image and a feature map is generated by it. Therefore, it is significantly faster than R-CNN, both in terms of training and testing time. In the graph below the comparison between R-CNN, SPP-Net and Fast R-CNN is shown:

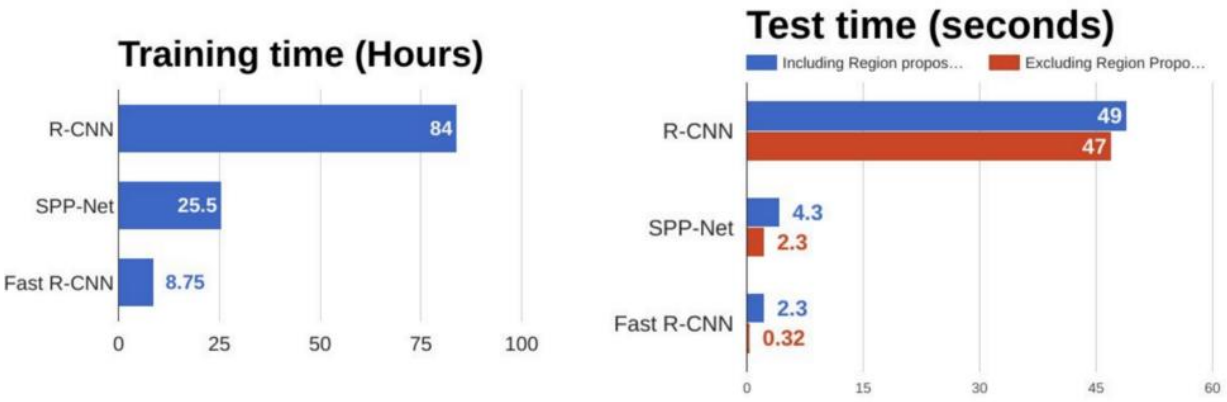


Figure 29: Comparison of R-CNN, SPP-Net and Fast R-CNN

4.3 Faster R-CNN

4.3.1 General

Although Fast R-CNN has significantly improved performance compared to R-CNN, both approaches use Selective Search algorithm to make the region proposals. However, this process is slow and time-consuming, which eventually affects the performance of the network. In order to counter this problem, Faster-CNN [3] was introduced, which is an algorithm based on Fast R-CNN that eliminates the Selective Search algorithm and lets the network learn the region proposals.

Similar to Fast R-CNN, the entire image is provided as an input to a convolutional neural network which outputs a convolutional feature map. However, instead of using the Selective Search algorithm to make the region proposals, a separate CNN-based network performs this task. The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and make the predictions for the bounding boxes. Therefore, the overall network has the following general pipeline:

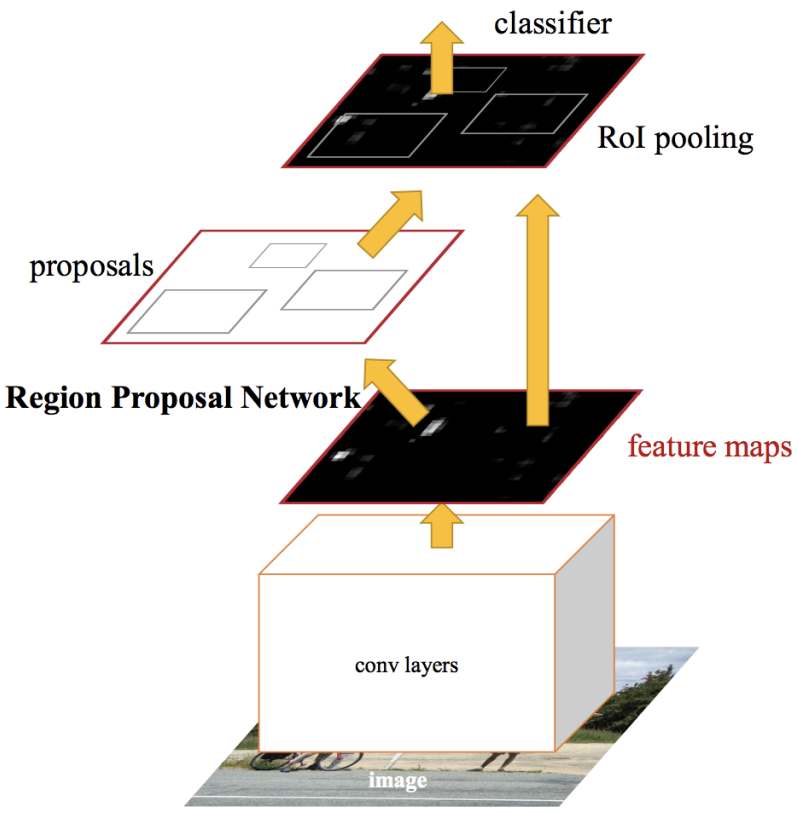


Figure 30: Faster R-CNN pipeline

The basic steps of the algorithm are the following:

1. Firstly, the image goes through convolutional laers and feature maps are extracted.

2. Then, a sliding window is used in the RPN (Region Proposal Network) for each location over the feature map.
3. For each location, 9 anchor boxes are used (3 scales of 128, 256 and 512, and 3 aspect ratios of 1:1, 1:2, 2:1) in order to generate the region proposals.

4. A box classification (cls) layer outputs 2k scores, whether there is an object or not in the k boxes.

5. A box regression layer (reg) outputs 4k coordinates (box center coordinates, width and height) of the k boxes.

6. For a feature map of size W x H, there are WHk anchors in total.

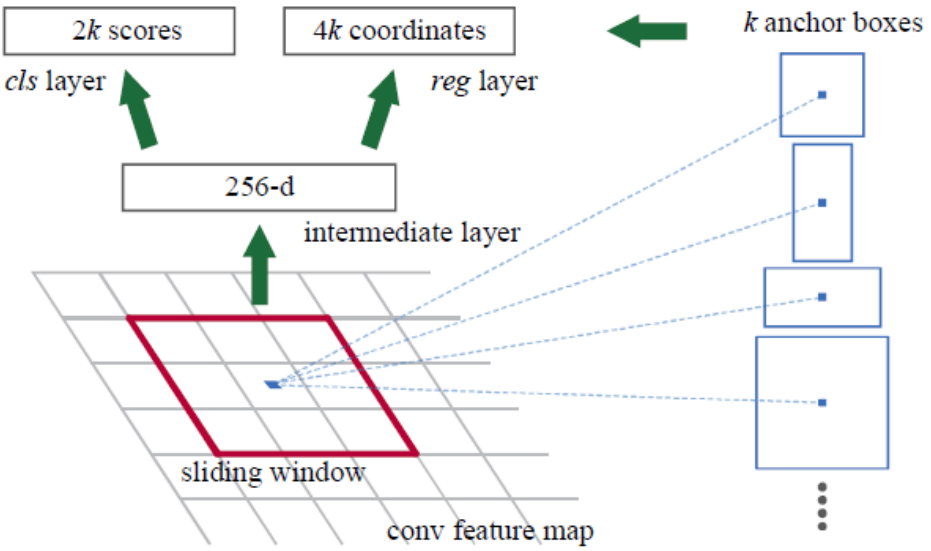


Figure 31: Region Proposal Network pipeline

The loss function of the Region Proposal Network is:

$$L(\{p_i\}\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{cls}(t_i, t_i^*)$$

The first term refers to the classification loss over classes, i.e. whether there is an object or not. The second term refers to the regression loss of bounding boxes only when there is an object ($p_i^* = 1$).

The λ parameter, is a value that has been proven to improve the efficiency of the network. It has been turned out that λ=10 achieves the best result in terms of mAP (%) score.

As region may have a high overlap with each other, non-maximum suppression is used to reduce the number of proposals.

Except the RPN, the remaining part of Faster R-CNN remains the same as Fast R-CNN. Firstly, RoI pooling is performed to the image, and then it is fed through a CNN and two fully connected layers.

From the below graph, it can be seen that Faster R-CNN is approximately 10 times faster than Fast R-CNN, since it does not use the time-consuming Selective Search algorithm.

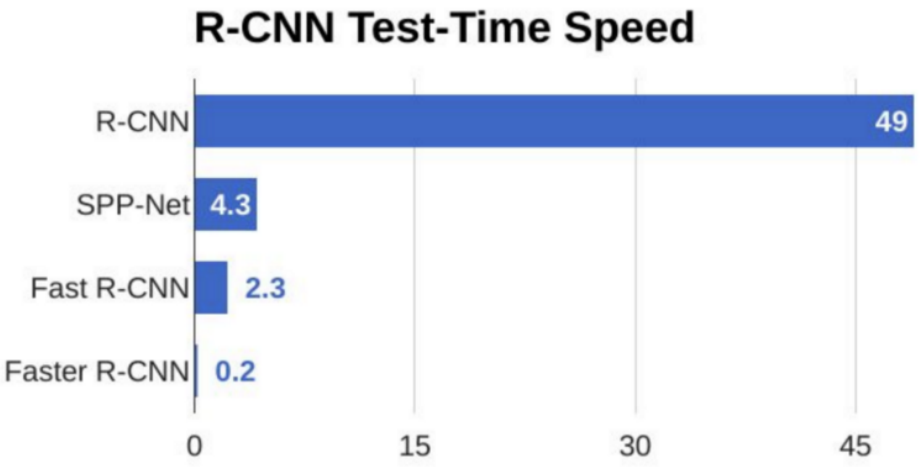


Figure 32: Comparison of state-of-the-art object detection algorithms

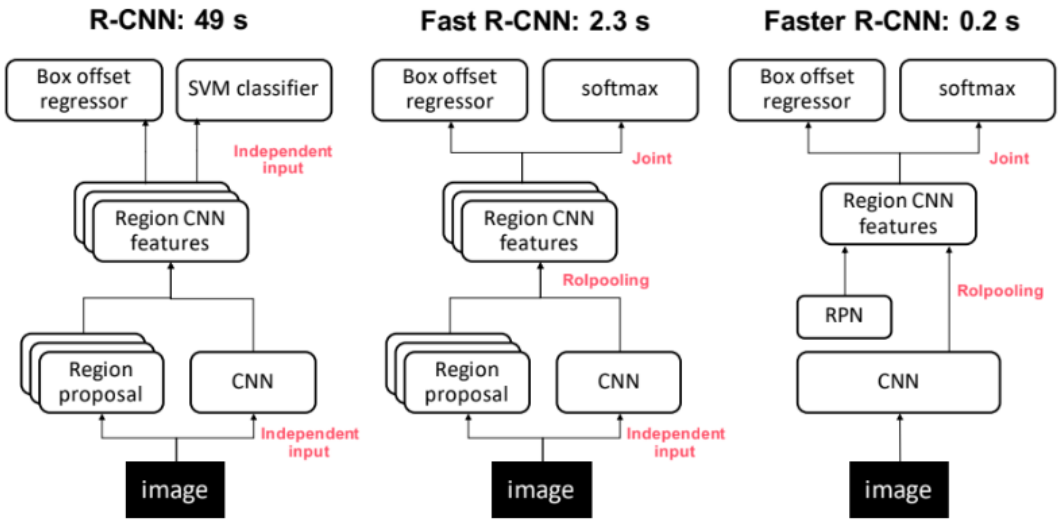


Figure 33: Illustration of the architecture of the R-CNN family and inference time speed of each model

4.4 You Only Look Once (YOLO) Algorithm

4.4.1 General

YOLO algorithm [6], which stands for "You Only Look Once", is a very efficient and fast object detection system. Its main and contemporary difference compared to prior work on the field is that detection is performed through predicting certain bounding boxes of objects while calculating a class probability of objects being included in that boundary. This is achieved through applying a single convolutional neural network on the image which outputs bounding boxes with the respective class probabilities without any other system interfering during the process. This makes YOLO a unique system, since it performs end-to-end detection through applying just one network to the input image.

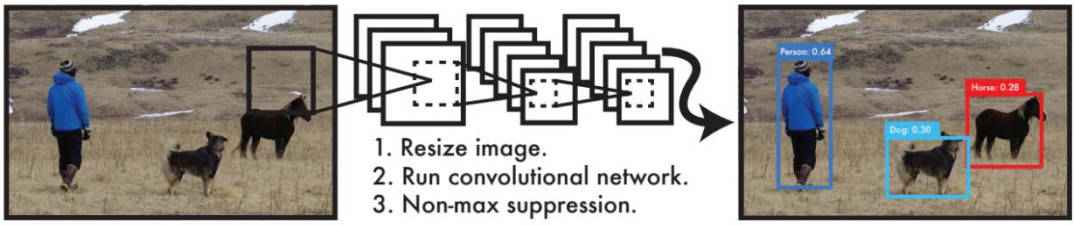


Figure 34: YOLO algorithm pipeline

YOLO can process images in real-time in 45 frames per second, outperforming many other state-of-the-art detection systems. Furthermore, its architecture enables it to learn generic representations and compared to other state-of-the-art systems it performs greatly on generalizing from general images to other domains such as artwork or visual environments.

Most systems up today use classifiers to perform detection by performing classification at various regions and scales in an image. For example, some networks use a sliding window approach were classification is implemented at even regions over the entire image. More recent and sophisticated systems use region proposal technique to generate some bounding boxes which are likely to include objects. Classifiers are then run over these proposed regions and detection is made. At a later stage, a process to eliminate duplicate detections and rescore the boxes based on other objects detected in the scene is made. Such systems are relatively slow and, unlike YOLO, are also very hard to optimize since they are composed of several different components, with each one requiring separate optimization.

More specifically, YOLO's benefits over other methods are:

1) YOLO is really fast, since its pipeline is really simple. A simple neural network is run through an image at about 45 frames per second, allowing YOLO to process video in real-time with only 0.25 milliseconds of latency.

2) YOLO, unlike other system sees the entire image when making predictions. This makes YOLO able to make detection considering information depending on context of the image that other networks do not process because they do not see the entirety of the image. Consequently, YOLO makes much less back-ground errors than traditional detection systems which may mistake background parts as objects.

3) A huge advantage of YOLO is that it can learn generalizable representations of objects. This makes YOLO able to perform detections in environments such as artwork, even if it has been trained on natural images.

4.4.2 Detection Process

YOLO takes the input image and divides it into an S x S grid. If the center of an object falls inside a grid cell, then that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes, which reflect how confident the model is that the box contains an object and how accurate it thinks the box is that it predicts. Confidence is defined by the following measure:

$$\Pr(Object) * IOU_{pred}^{truth}$$

, where IOU is the Intersection over Union between the predicted box and the ground truth box. Its value ranges between 0 and 1 and corresponds to the overlapping area between the predicted box and the ground-truth box. The higher the IoU, the better the predicted location of the bounding box for a given object is.

If no object exists inside that cell, the confidence score is zero.

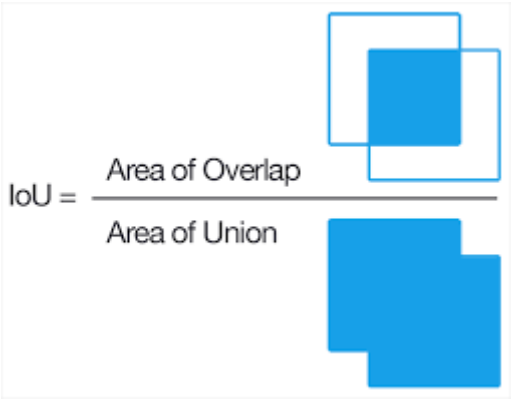


Figure 35: Intersection over Union

Each bounding box prediction consists of 5 parameters: *x, y, w, h* and confidence. The (x,y) coordinates represent the centre of the bounding box relative to the boundaries of the grid cell. The width and height of the box are relative to the whole image. Finally, the confidence parameter represents the IOU between the predicted box and the truth box.

Each grid cell also predicts C conditional class probabilities, $\Pr(\text{Class}_i|\text{Object})$, representing the probability that class_i contains an object.

The conditional class probability is multiplied by the individual box confidence score which gives the class-specific confidence score for each box. This reflects the probability of the box containing an object of a certain class and how well the predicted box fits the object.

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$$

This procedure is described in the image below:

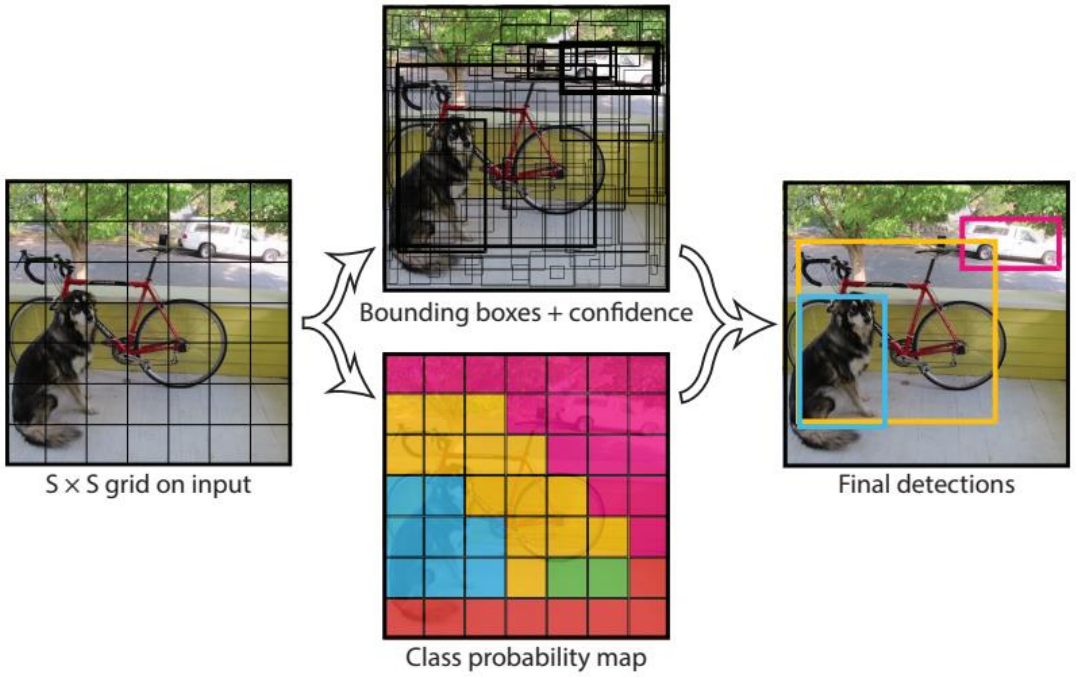


Figure 36: Image is divided into S x S grid cell and for each cell B bounding boxes are predicted. Predictions are encoded into an S x S x (B*5 + C) tensor

4.4.3 YOLO Architecture

YOLO model 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. (where S = 7 and C = 20).

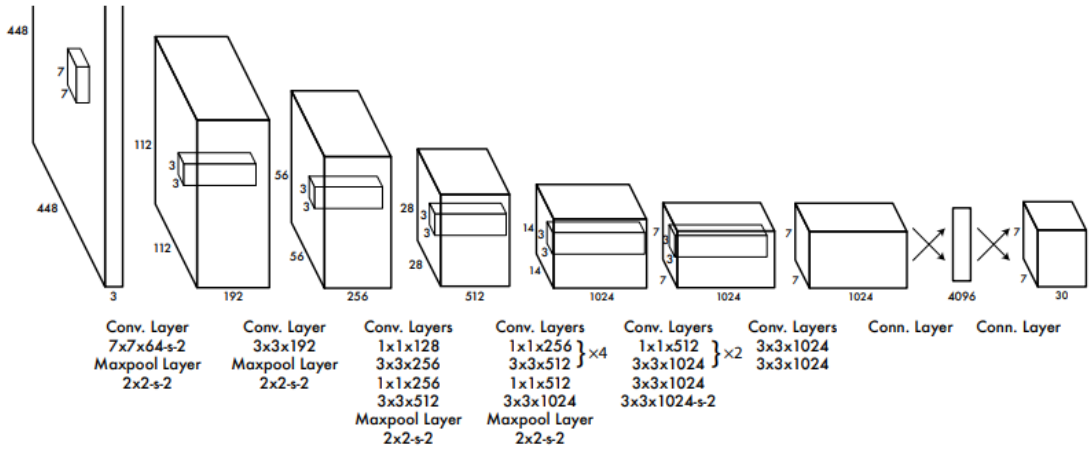


Figure 37: Network with 24 convolutional layers followed by 2 fully connected layers. Feature space is reduced from preceding layers

A linear activation function is used for the final layer and all other layers use the following LReLU (Leaky Rectified Linear Unit) function:

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

Sum-squared error is used in the output of the model. Although it is easy to optimize, sum-squared error weights localization error equally with classification error which is not ideal in terms of maximizing average precision. Furthermore, in an image a lot of grid cells do not contain any object, which leads confidence scores of those grid cell close to zero. This can cause overpower of gradient, which may lead to instability and divergence of the model in an early stage of the process.

For that reason, loss from bounding box coordinate predictions is increased, whereas loss from confidence predictions for boxes that contain no object is decreased. Therefore, two parameters $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$ are used.

Another imperfection of the sum-squared error is that error produced from large boxes is equally weighted as error produced from small ones. However, it is essential that small deviations in large boxes do not matter the same as in small boxes. To counter this problem, instead of predicting the height and width of the bounding boxes, the square root is used.

Moreover, during training, multiple bounding boxes per gird cells are predicted. However, only one bounding box should be responsible for each object. To determine which one will be responsible among the multiple boxes, the IOU with the ground truth is calculated and the one with the highest IOU is selected.

4.4.4 Loss Function

The loss function of the network is:

$$\begin{aligned} &\lambda_{coord} \sum_{i=0}^{S^2} \sum_{J=0}^{B} l_{ij}^{obj} [(x_i - \hat{x_\iota})^2 + (y_i - \hat{y_\iota})^2] \\ &\quad + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} l_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + \sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ &\quad + \sum_{i=0}^{S^2} \sum_{j=0}^{B} l_{ij}^{obj} (C_i - \hat{C}_i)^2 + l_{ij}^{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} l_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\ &\quad + \sum_{i=0}^{S^2} l_{ij}^{noobj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

where l_i^{obj} denotes if an object appears in cell I and l_{ij}^{obj} denotes that the j^{th} bounding box predicted in cell i is responsible for that prediction.

4.5 YOLOv3 Algorithm

4.5.1 General

YOLOv3 [8], which stands for YOLO version 3 is an updated version of YOLO algorithm with improved accuracy. The output of the network is a feature map with (B x (5 + C)) entries. B represents the number of bounding boxes each grid cell of the image can predict. Each of the bounding objects has (5 + C) attributes, which describe the centre coordinates, the dimensions (width and height), the objectness score, and C class confidences for each bounding box, where C is the number of classes.

The objectness score represents the probability that an object is contained inside a bounding box. It should be nearly 1 for grids that contain an object and almost 0 for grid cells that do not contain any object.

Class confidence represents the probability of the detected object belonging to a particular class (such as dog, car, fruit, etc.).

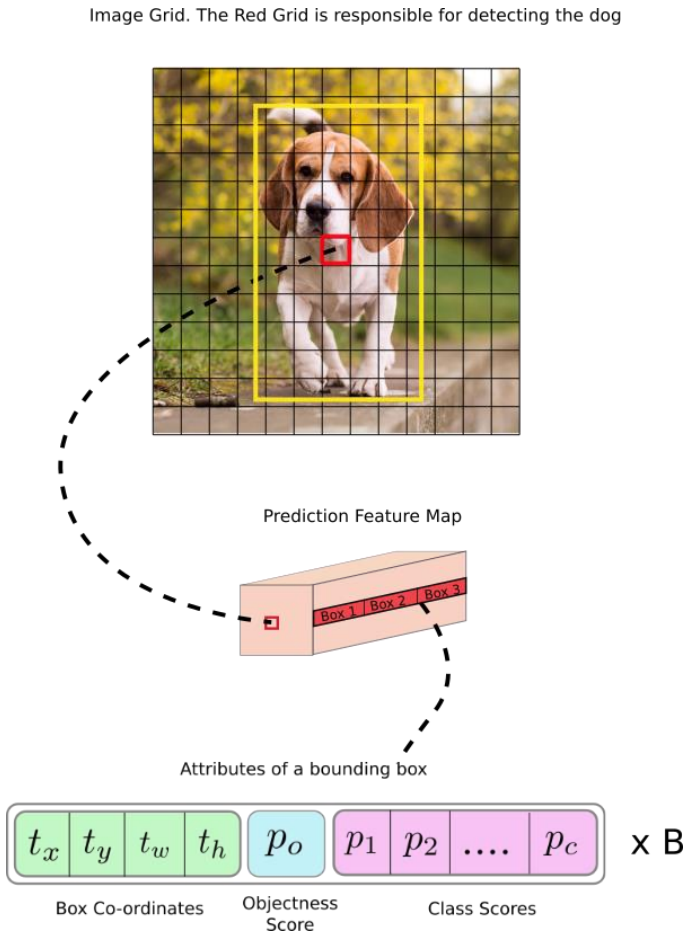


Figure 38: Image divided into grid cells. Prediction of an S x S x (B*5 + C) tensor

4.5.2 Bounding Box Prediction

As the older versions of YOLO, instead of predicting the width and the height of the bounding box, YOLOv3 predicts offsets to pre-defined default bounding boxes, called anchors. Then some transforms are applied to these anchors, in order to obtain the prediction. YOLOv3 has three anchors, therefore three bounding boxes per grid cell are predicted. The box with the highest IoU (Intersection of Union) will be the one responsible for the prediction. For each box, 4 coordinates are predicted t_x , t_y , t_w and t_h , according to the following formulas:

$$b_X = \sigma(t_X) + c_X$$

$$b_Y = \sigma(t_Y) + c_Y$$

$$b_W = p_W e^{tw}$$

$$b_h = p_h e^{th}$$

, where c_X , c_Y are the offsets of the grid cell from the top left corner of the image and p_W and p_h are the anchor width and height of the bounding box. Sigmoid function is used.

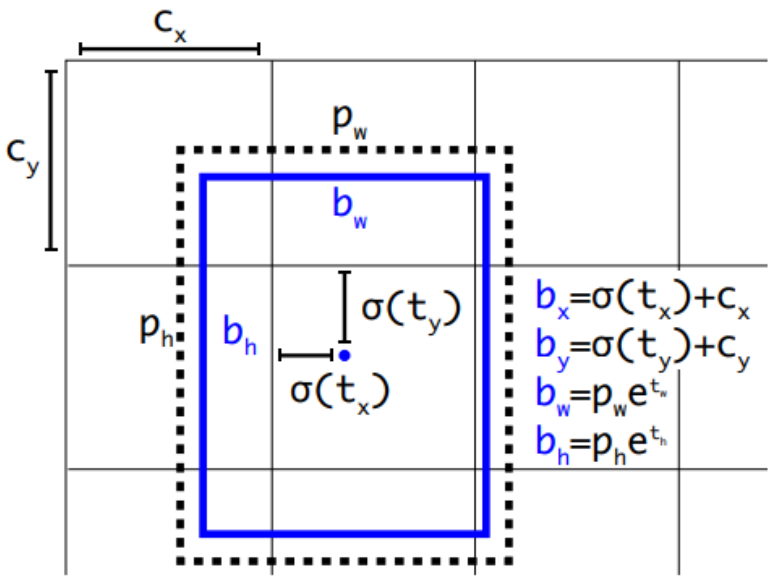


Figure 39: Bounding box prediction

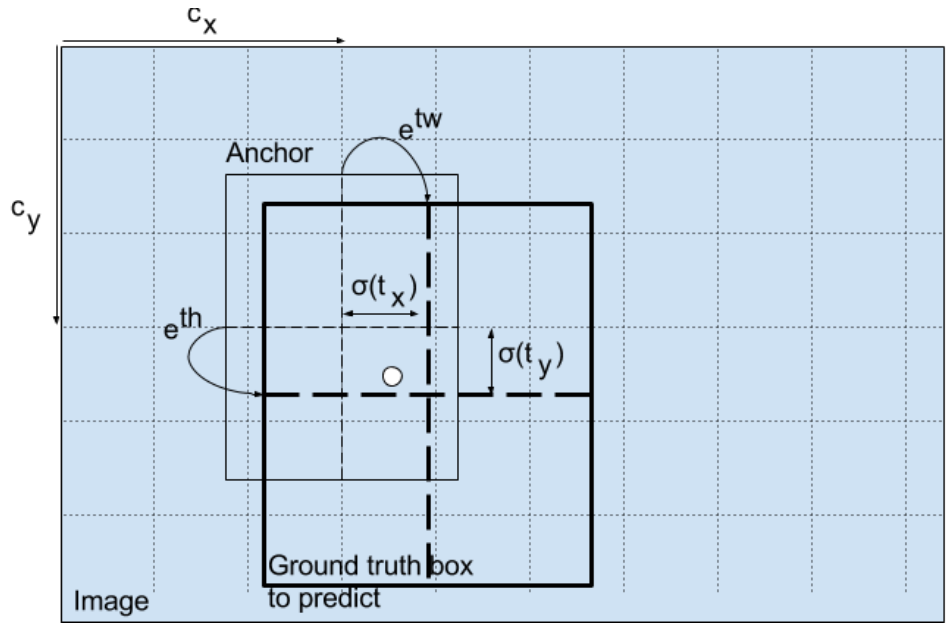


Figure 40: Transformation to give the final prediction of the bounding box

During training sum of squared error loss is used by subtracting the prediction for some coordinate from the ground truth coordinate: $\hat{t}_* - t_*$

YOLOv3 predicts a score for each bounding box using logistics regression. The score is assigned to 1, if the bounding box overlaps a ground truth object by more than any other bounding box prior. If the bounding box is not the best but does overlap a ground truth object by a threshold of 0.5, the prediction is ignored.

YOLOv3 predicts bounding boxes at three different scales from which it can extract features. The detection layer makes detection at feature maps of three different sizes, having strides of 32, 16 and 8 respectively. This means, that with an input of 416 x 416, for example, detections are made on scales of 13 x 13, 26 x 26 and 52 x 52. This advancement to the algorithm helps YOLOv3 to get better at detecting small objects.

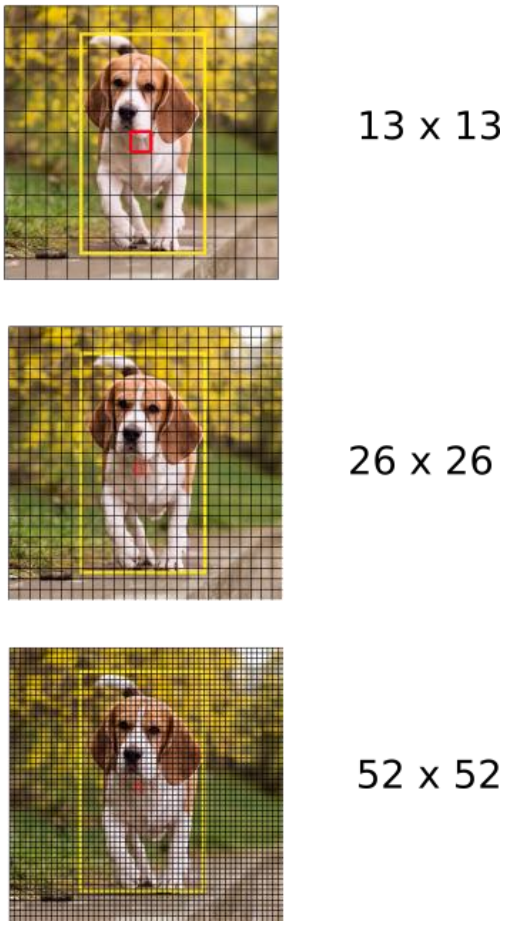


Figure 41: Feature maps at three different scales

For an image, say of size 416 x 416, YOLO predicts a number of ((52 x 52) + (26 x 26) + (13 x 13)) x 3 = 10647 bounding boxes, as each cell predicts 3 boxes. However, in the case of just one object being contained in the image, the detections are reduced from 10647 to 1.

Firstly, boxes with an objectness score below a particular threshold are ignored.

Furthermore, the process of Non-maximum Suppression is implemented. NMS intends to cure the problem of multiple detections of the same image.

4.5.3 Performance Metric

The commonly used metric used for object detection challenges is called the mean Average Precision (mAP). It simply is the mean value of the Average Precisions computed over all classes, as described by the following formula:

$$mAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q}$$

In the image below, the performance on the COCO dataset of YOLOv3 network compared to other state-of the art networks is presented:

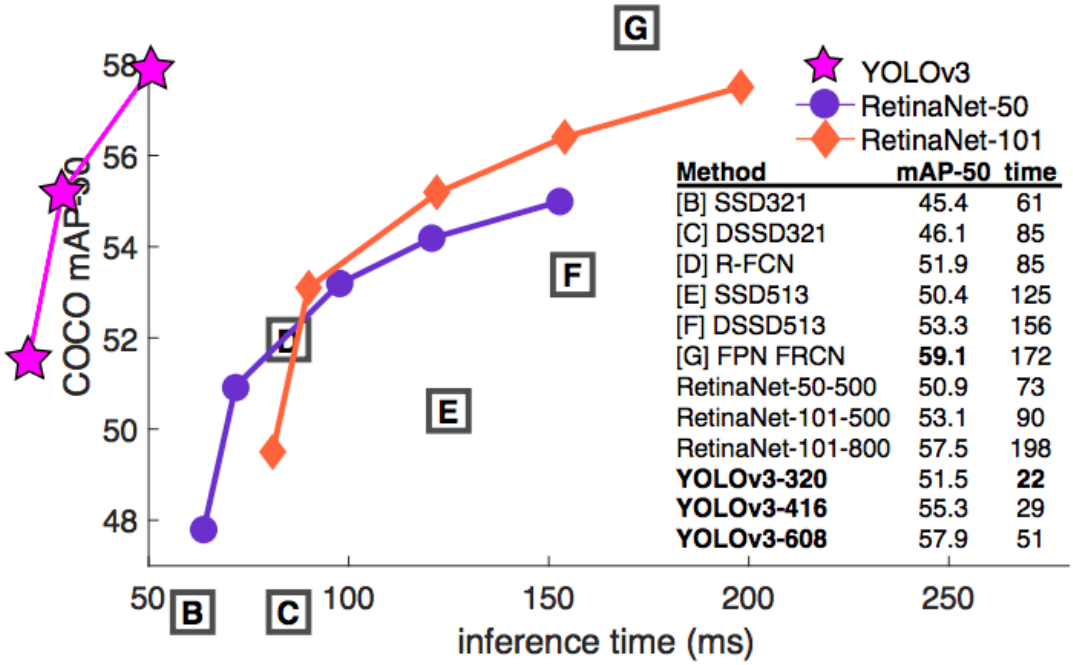


Figure 42: Performance of YOLOv3 compared to other state-of-the-art algorithms

5. Case Study

5.1 General Description – Goal

In the present study, the main goal was to make a first attempt at performing object recognition using machine learning methods in the field of Naval Architecture and Marine Engineering, in the effort of being able to describe accurately the content of an image of a ship's engine room. More specifically, the basic task was to perform recognition of the basic components of complex piping systems, such as flanges, straight pipes, elbow pipes and valves using a convolutional neural network-based approach. Furthermore, due to the fact that constructing a large enough dataset in order to train a CNN is a very time-consuming task that also requires manual intervention, an attempt was made to automate this procedure, as well as significantly minimizing the required time.

Specifically, generating a large enough and accurate dataset of particular objects in order to train a neural network is a difficult task, since it requires collecting numerous images of different scales and poses, visually inspecting them and also specifying the exact position of objects in the picture. This is done by adding to the dataset annotation files for every image, which indicate the relative position of bounding boxes, their dimensions, as well as the class of every object. This is, of course, a time-consuming task, since it can only be done through manually tagging objects by drawing bounding boxes around them. Moreover, as images get more complicated, containing several different objects of different classes, this task gets even more difficult. Although this has been the way which datasets have been generated until today, it is an inefficient process, as it is not only a time-consuming procedure, but also involves human error which can significantly affect training and consequently, the network's performance. For example, tagging an object at a wrong position, or mistakenly labelling an object to a different class may result in creating wrong training material.

Our proposed method can remarkably speed up the process of generating a dataset by creating as many training images as required, as well as creating the annotation files for every image which specify the position of objects and their class. Moreover, our current approach can ensure the validity of images included in the training dataset and can also generate images of objects at different scales and orientations, which are important for training a robust object recognition model. Furthermore, our method offers flexibility in terms of training a custom model for object detection, since it is able to generate a dataset for any object of our choice.

5.2 Used Tools

5.2.1 Darknet

The basic tool that was used for training our own custom object detection model is "Darknet" (https://github.com/pjreddie/darknet). Darknet is an open source neural network framework written in C and CUDA, officially launched in Linux system, which implements the aforementioned YOLO algorithm. It supports both CPU and GPU computation. This framework offers real-time object detection and classification by using some weights that were pre-trained in detecting some typical objects, such as cats, dogs, people, computers, etc.

Darknet also offers the option of training our own custom model on our own dataset, in order to detect objects of our choice. More specifically, to train a custom model, Darknet must be provided with a dataset, which comprises of the training images in .jpg format and some annotation files in. txt format that accompany each image and include information about the position of objects in the image by specifying the coordinates and dimensions of the bounding boxes.

Specifically, for every image in the dataset, the annotation file (.txt) must be saved in the same directory and with the same name and should have the following format:

```
<object-class> <x_center> <y_center> <width> <height>
```

, for each object on the image in a different line.

Where:

- `<object-class>` is an integer object number from 0 to (classes – 1) that indicates the class of the object
- `<x_center> <y_center>` are float values relative to the width and the height of the image, representing the coordinates of the centre of each bounding box and can be equal from 0.0 to 1.0 and are measure from top left corner.
- `<width> <height>` are float values relative to the width and height of the image, representing the width and height of the bounding box and can be equal from 0.0 to 1.0.

Note: `<height> = <absolute_height> / <image_height>`, where absolute height and image height are measured in pixels.

Figure 43: Sample image of 1 class (dog)

For example, for the above (1920 x 1080) image the annotation file (.txt) must have the following format:

```
0 0.682237 0.537409 0.382481 0.923356
```

, where 0 indicates the “dog” class and the rest numbers indicate the coordinates of the centre of the bounding box and its width and heights, relative to the image width and height measured from the top left corner.

5.2.2 How to use Darknet

To perform object detection with Darknet on Linux system, the following command must be used on the command line:

```
./darknet detector test ./cfg/coco.data ./cfg/yolov3-voc.cfg ./yolov3.weights
```

, where `coco.data` is a file that specifies the paths to some files necessary for training and has the following format:

```
classes= 80

train = data/coco/trainvalno5k.txt

valid = data/coco_testdev

#valid = data/coco_val_5k.list

names = data/coco.names

backup = backup/

eval=coco
```

, where:

- `classes` is the number of different classes in the dataset
- `train` specifies the images used for training
- `valid` specifies the images used for validation
- `names` contains the names of different classes (e.g. dog, cat, etc.)
- `backup` is the folder in which weights are backed up after a fixed number of iterations
- `eval` is the evaluation dataset

`yolov3-voc.cfg` is the configuration file. This configuration file contains all the information about the process of the image through the network and specifies the exact architecture of the neural network. Particularly, it defines all values that are going to be used in each convolutional, maxpooling and fully connected layer of the CNN. In our case, the yolov3-voc architecture was used, which is YOLOv3 trained on the VOC dataset. All the important parameters and hyper-parameters are stored in the configuration file, which has the following format:

```
[net]
# Testing
 batch=1
 subdivisions=1
# Training
# batch=64
# subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 50200
policy=steps
steps=40000,45000
scales=.1,.1



[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

```

Figure 44: Configuration file of YOLOv3 trained on VOC dataset

In this point, it is essential to make some clarifications regarding the main parameters of the system, defined in the configuration file:

- Batch: The batch parameter indicates the batch size used during training. Specifically, when the network processes the training images it is unnecessary in terms of time and computational power to process all images at every iteration. For that reason, just a subset of the training set is used to update the weights. This is subset is referred to as batch. A commonly used value for this parameter during training is 64, whereas during testing, batch should be assigned to 1.

- Subdivisions: Even though a batch size, say of 64 images is used, computational power might be limited, and therefore GPU may run out of memory. For that reason, Darknet provides the subdivisions parameter, that defines a fraction of the batch size that will be processed at a single iteration. Therefore, GPU will process a number of *batch/subdivisions* images at a time. During training, subdivisions parameter shall be assigned to multiples of 2 (e.g. 2, 4, 8, 16), whereas during training it should be 1.

- Width, Height and Channels: These parameters refer to the dimension (width x height) that the network resizes every image to, before processing it during training. A commonly used value is 416 x 416. Of course, results may improve if the resolution is increased (e.g. to 608 x 608) but, it would take longer to train. The channels parameter is set to 3 and indicates that an RGB image will be processed. During testing it is advisable to increase width and height (e.g. to 1024 x 1024).

- Momentum: In a CNN the weights of the neural network are updated based on a small batch of images in each iteration. For that reason, there is a relatively big fluctuation of the weights update. As mentioned in previous chapters, the momentum rate is used to penalize those huge fluctuations of weight updates between consecutive iterations. A commonly used value for the momentum rate is 0.9.

- Decay: Decay parameter is an additional term in the weight update rule that helps the network to prevent from overfitting. Specifically, overfit neural networks are often characterized by large weight values, so it is important to prevent weight values to get large in magnitude. To do so, it is advisable to regularize the loss function, as described below by adding a weight penalty term to the plain error term:

$$\tilde{E}(w) = E(w) + \frac{\lambda}{2}w^2$$

  By doing this, large weight values are penalized, since larger weights produce larger outputs. The regularization parameter λ, determines the penalty of the weights update. By applying gradient descent to the new cost function through differentiating it, the weights are updated as following:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \eta\lambda w_i$$

  The term $\eta * \lambda$ is the decay parameter which in our case was set to 0.0005.

- Learning Rate, Steps, Scales: As mentioned in previous chapters, the learning rate parameter controls how fast weights' values change. It is typically a number between 0.0001 and 0.1. At the beginning of the training process, where there is not a lot of information and the network does not fit the inputs well, the learning rate should be high. However, as the network processes more data and it is converging towards minimizing the loss function, weights should change less aggressively.

Therefore, learning rate must decrease over time. For that reason, the steps parameter is implemented, which indicates that the learning rate will remain constant for a number of iterations, and then it will be decreased. It is advised that steps should be 80% and 90% of max batches value, which means that after 0.8*max batches of iterations, learning rate will be decreased and after a total of 0.9 max batches of iterations, it will be further decreased. The scales parameter will be multiplied with the learning rate, and thus, it specifies by how much the learning rate will decrease. As mentioned above, steps and scales parameters may be multiple.

- Burn-in: Although learning rate should be high in the beginning and lower later on, as mentioned in the previous paragraph, it has been empirically proved that training speed may increase if a lower learning rate is used for a short period of time at the very beginning. This is controlled by the burn-in parameter, which is usually set to 100 – 1000 iterations.

- Angle, Saturation, Exposure, Hue: These parameters are used to create new data from the current inputs. This process is referred to as *data augmentation*. Specifically, an image containing a pipe rotated by 5 degrees will still be an image of a pipe. So, the angle parameter allows to randomly rotate the given image by +- angle. In a similar way, if the colours of the entire picture are transformed, the image is still a pipe. This transform is applied through the saturation, exposure and hue parameters.

- Max Batches: Max batches parameter refers to how many iterations the network will perform during training. For images with multiple classes, it is required to run for a greater number of batches. It is advised that for an n-class object detector, training should be run for at least 2000*n iterations.

- Anchors: Anchors parameters refers to the anchor boxes, which are a set of predefined bounding boxes of a certain height and width. These boxes are defined to capture the scale and aspect ratio of specific object classes based on object sizes on the training dataset. During detection, the anchor boxes are tiled across the image. The CNN predicts the probability, as well as some other attributes such as IoU value and some offsets for every anchor box. Therefore, the network does not directly predict bounding boxes but rather predicts the probabilities that correspond to each anchor box. The use of anchor box enables the network to detect multiple objects of different classes, object of various scales, as well as overlapping objects.

Furthermore, the configuration file contains all necessary information about every single layer of the neural network. Specifically, it specifies all parameters for the convolutional layers, which have been described in previous chapters, such as batch normalization, filters applied to the image, kernel stride and padding and what type of the activation function is used.

Moreover, the configuration file includes further information about the anchors, which define the proposed bounding boxes of the network, and the ignore and truth threshold which specify which bounding boxes will be predicted with respect to their confidence score.

Finally, `yolov3.weights` is the file that contains the weights of the neural network. When training a custom model, either pre-trained weights or the backed-up weights from previous trainings can be used.

5.2.3 How to train custom model in Darknet

On Linux, to train a custom model in Darkflow, the following command must be used in the Terminal:

```
./darknet detector train data/obj.data yolo-obj.cfg darknet53.conv.74
```

, where `obj.data` is the .data file as described in the above chapter

`yolo-obj.cfg` is the configuration file

`darknet53.conv.74` is the convolutional layer pre-trained weights file

5.2.4 How to run Darknet

On Linux, to perform object detection with Darknet, the following command must be used in the Terminal:

```
./darknet detector test ./cfg/coco.data ./cfg/yolov3.cfg ./yolov3.weights
```

5.2.5 Darkflow

The second tool that was used is Darkflow (https://github.com/thtrieu/darkflow), which is Darknet translated into Tensorflow, an open source machine learning platform and can also be used in Windows system. Like Darknet, Darkflow performs real-time object detection and classification.

Just as Darknet, Darkflow also offers the option of training a custom model. To perform custom object detection, it should be provided with a dataset comprising of the training images, as well as the annotation files in (.xml format), similarly to the (.txt) files in Darknet. These files include information about the exact position of objects in an image by specifying the coordinates of the upper left and lower right edges of the bounding boxes, as well as its dimensions (width and height). The annotation files have the following format:

```
<annotation>
  <folder>train/Images</folder>
  <filename>0001.jpg</filename>
  <segmented>0</segmented>
  <size>
    <width>600</width>
    <height>600</height>
    <depth>3</depth>
  </size>
  <object>
    <name>pipe</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>293</xmin>
      <ymin>217</ymin>
      <xmax>303</xmax>
      <ymax>382</ymax>
    </bndbox>
  </object>
</annotation>
```

Figure 45: Annotation file format

Aside from defining the position of bounding boxes in the image, the annotation file includes some further information:

- `Folder:` specifies folder containing image
- `Filename:` specifies image name

- `Segmented`: specifies if the object is segmented or not (either 0 or 1)
- `Width`: Width of image in pixels
- `Height`: Height of image in pixels
- `Depth`: Depth of picture (3 for RGB)
- `Name`: name of object's class
- `Truncated`: specifies if the object is truncated or not (either 0 or 1)
- `Difficult`: specifies if the object is difficult to detect or not or not (either 0 or 1)
- `(xmin, ymin)`: specifies the upper left edge of the bounding box in pixels, measured from top left corner of image
- `(xmax, ymax)`: specifies the lower right edge of the bounding box in pixels, measured from top left corner of image

5.2.6 How to train a custom model in Darkflow

To train a custom model in Darkflow, the following command must be used in the Command Prompt:

```
flow --model cfg/yolo-new.cfg --load bin/tiny-yolo.weights --train --gpu
1.0
```

5.2.7 How to run Darkflow

To perform object detection with Darkflow, the following command must be used in the Command Prompt:

```
flow --imgdir sample_img/ --model cfg/tiny-yolo.cfg --load bin/tiny-
yolo.weights --gpu 1.0
```

5.2.8 How to improve object detection

Before training a network, some basic steps regarding the training dataset should be followed in order to improve the detection accuracy:

- It is mandatory that each object should be labelled in the dataset – no object in the dataset should be left without label.

- For each object that is to be detected there must be at least 1 similar object in the training dataset with about the same shape, side of object, relative size, angle of rotation, tilt and illumination.

- It is desirable that the training dataset includes images with objects at different scales, rotations, lightings, from different sides, in different backgrounds. It is ideal that at least 2000 different images for each class are used.

- It is also desirable that the training dataset includes images with non-labelled object that are not to be detected. Non-labelled images should be as many as images with objects.

5.2.9 Dependencies

Image detection in Darkflow is executed through the Command Prompt and requires some dependencies to run, such as Python, OpenCV, Visual Studio and Tensorflow. Additionally, some further Python libraries are used.

5.3 Steps in Case Study

The main goal of our case study was to design an efficient custom object recognition model using the Darknet and Darkflow frameworks. We were mostly involved with detecting basic mechanical components of complex piping systems, such as pipes and flanges. This effort was divided into some basic gradual steps, in order to reach our final result. The basic steps that were followed are:

1. Object detection using Darkflow in Windows environment. Dataset was created using Google images
2. Object detection using Darkflow in Windows environment. Dataset was automatically created through importing models into a project designed in Unity which will later be further discussed.
3. Object detection using Darknet in Linux framework with the basic components of complex piping systems. Specifically, the basic components of piping systems (such as pipes and flanges) were separated in order to train a more efficient network.
4. Object detection with objects of different materials. Specifically, the network was trained considering objects of various different materials used in piping systems (such as aluminium or stainless steel) in order to simulate more accurately real conditions of pipes.
5. Object detection with additional components of piping systems added, such as valves, straight pipes and elbow pipes

The aforementioned steps were followed in order to have some comparative results and test the speed and accuracy of each method.

5.3.1 Object detection using Darkflow in Windows with Google training images

During the first stage of our case study, we attempted to train a custom model using Darkflow framework that can detect flanges in piping systems. To do so, we collected a dataset of approximately 300 Google images. A GitHub code (https://github.com/hardikvasa/google-images-download) was used to automatically download a number of google images using the key word "flange", in order to speed up the image collection process. However, a visual inspection was mandatory after downloading the dataset in order to detect faulty images, such as images not including flanges or images that were not in a .jpg format. The annotation files were created manually for every image through an editor by drawing bounding boxes around each flange. This was done by specifying the upper left and the lower right edge of the bounding box. Although the custom trained network performed well, the process of manually creating the annotation files was not ideal especially for cases were large datasets are required and there are multiple objects of different classes in each image, since it is a time-consuming task.

After creating the dataset, we trained our own system, using yolov3 configuration file with the following parameters:

Batch = 64

Subdivisions = 8

Height = 416

Width = 416

Learning rate = 0.001

Max batches = 4500

Decay = 0.0005

Backup iterations = 100

Threshold = 0.6

Classes = 1 (flange)

The duration of training was approximately 1 day, and the results were quite successful, as can be seen in the images below:

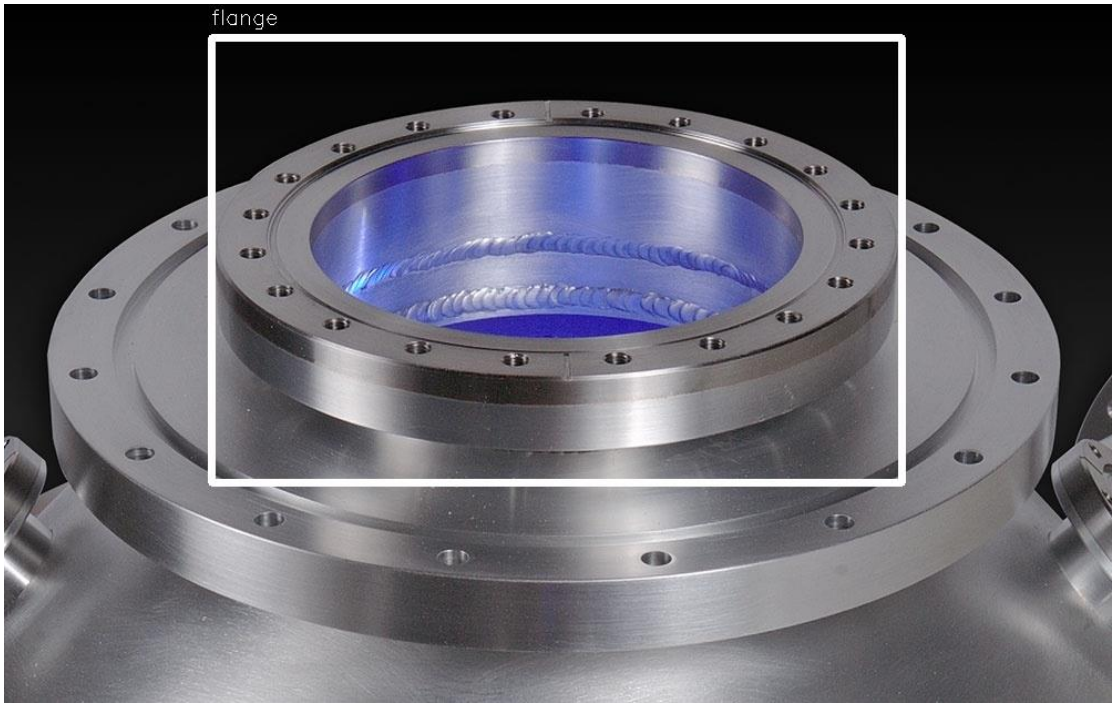


Figure 46*:* Flange successfully detected

Figure 47*:* Both flanges successfully detected

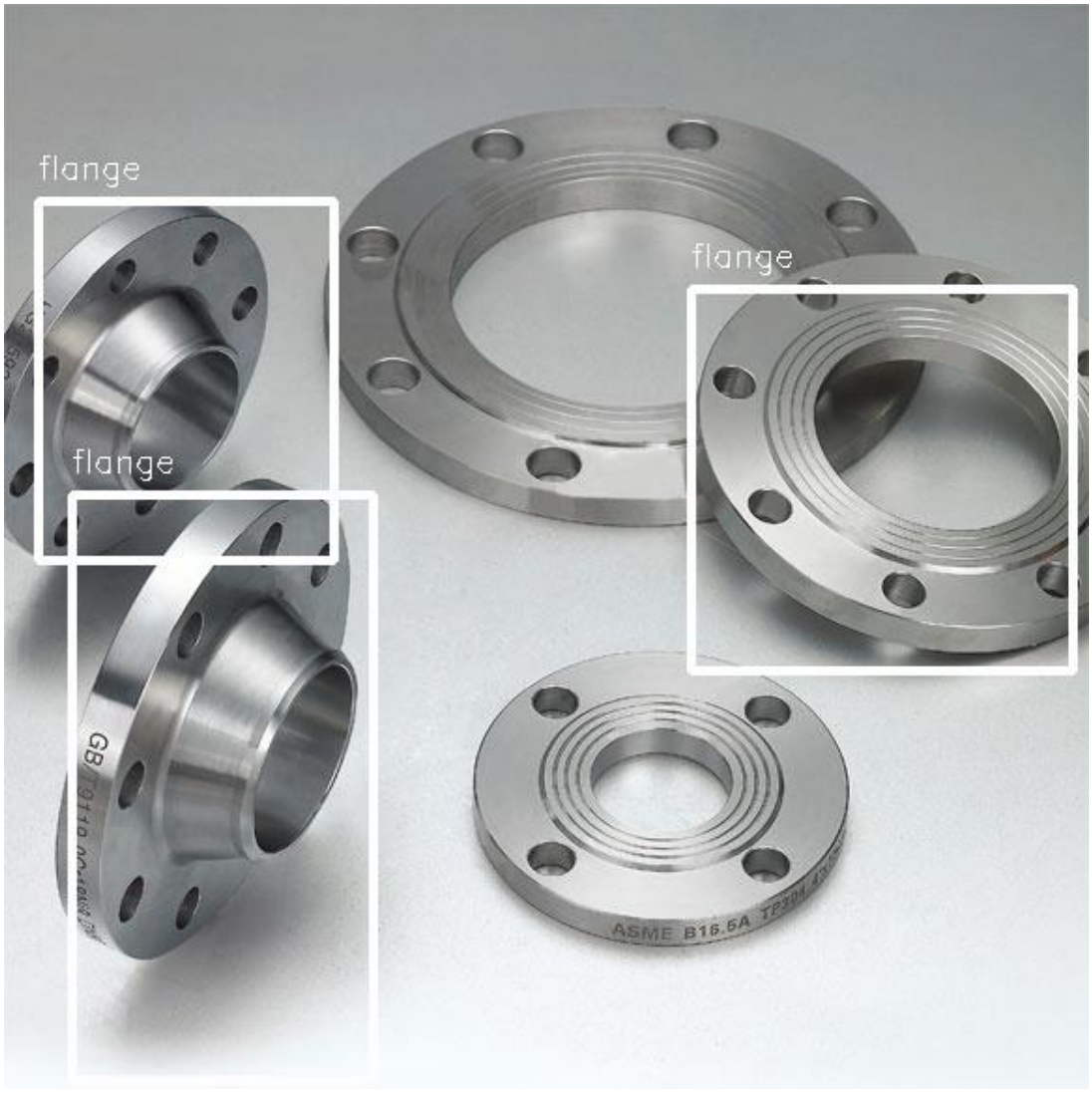


Figure 48*:* Most of flanges successfully detected

5.3.2 Object detection using Darkflow in Windows with automated training

At the second stage of our case study, an approach of automating and improving the training procedure was made. Although training with sample Google images has some fine results, this method is not ideal for tasks that require large datasets and high efficiency, since it requires manually collecting all images, visually inspecting them and then drawing bounding boxes around objects. Therefore, our goal was to non-manually create a large dataset that contains multiple images of objects of our choice in various different sizes and orientations. Furthermore, an automated way of generating annotation files for each training image was required in order to fully-automate the training procedure.

In order to meet these requirements, some tools were used. Firstly, we imported some simple 3D piping models into the environment of Unity. Our goal was to generate a large number of models' images, each one accompanied by its annotation file which specifies the exact position of an object into an image. In order to achieve this, a project was created in Unity, which also includes some codes written in C# which are responsible for the following tasks:

1. Develop a tool in Unity that allows random or manual rotation and scaling of any imported object around the 3 axes. Thus, we can have images with different orientations and sizes of the object. These actions can be performed either manually by simply pressing a single keyboard button, or automatically by applying a rotation after a fixed time step.

2. Create a bounding box around the imported 3D model by detecting its extreme edges. This bounding box should be following the geometry of the object at any rotation and scaling.

3. After every rotation and scaling of the model, automatically or manually capture a screenshot of the object and save it in a specified directory.

4. Along with capturing a screenshot, generate the annotation files, both in (.xml) and (.txt) format that contain information about the position and orientation of the objects in the image. Both captured images and annotation files can be saved to directories of the user's choice.

By performing the above steps, we were able to generate a dataset, as large as required, as well as create the annotation files required by Darknet or Darkflow in order to train a custom model. This way, we managed to significantly accelerate both the procedure of image collection as well as object tagging, which is one of the most challenging and time-consuming tasks when training a custom object detection model.

Furthermore, it was important for the proper functioning of the project that every imported model into Unity would be pinned to the center of the Unity scene. To do so, a program was developed into Grasshopper plug-in of Rhinoceros 3D that calculates the center of gravity of each object and then translates this point to the center of global axes. After this process, the model is exported into (.fbx) format, in order to be compatible with Unity.

The green “Geo” block takes a meshed geometry as an input in order to be edited by the program. The second “Volume” block takes the geometry as input and computes its volumetric centroid, which is depicted in the yellow box as 3 coordinates along the x, y, z axes. The output is then passed forward to the “Move” block which translates the volumetric center of the geometry to the global origin [0, 0, 0] by subtracting the center coordinates from every point of the object.

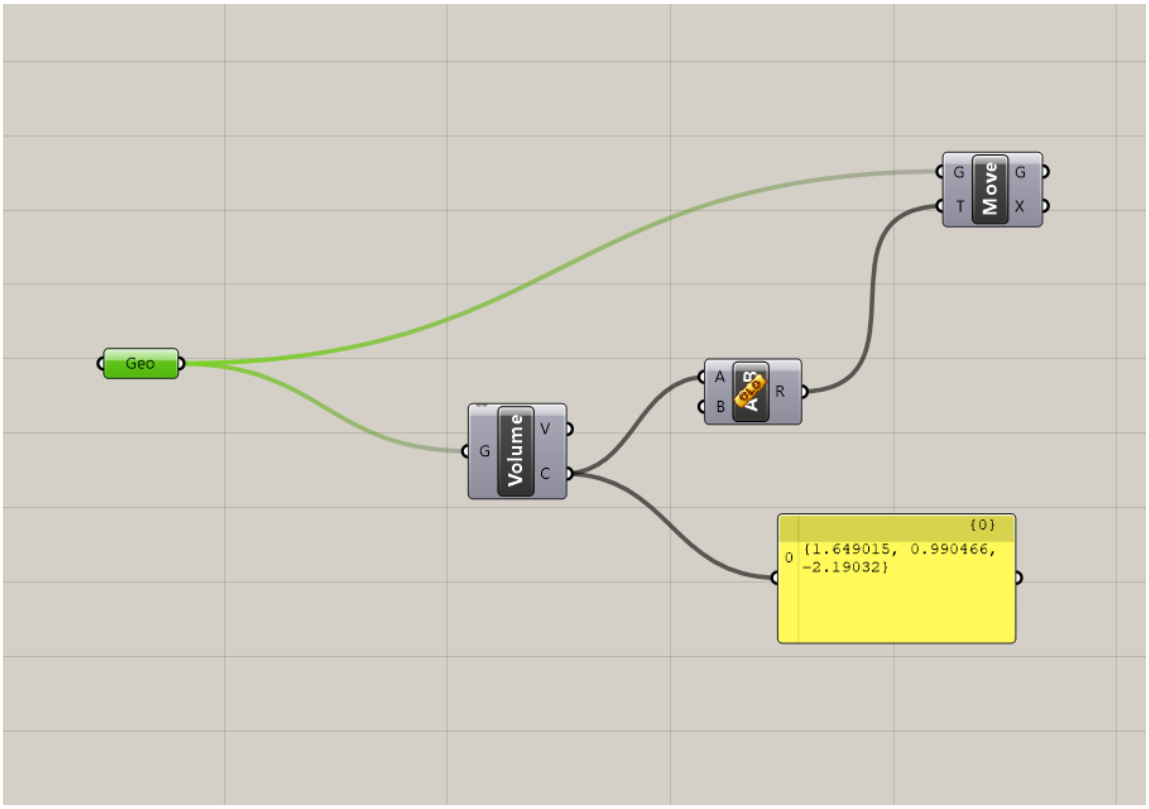


Figure 49: Computation of volumetric center in Grasshopper and translation of object to global origin

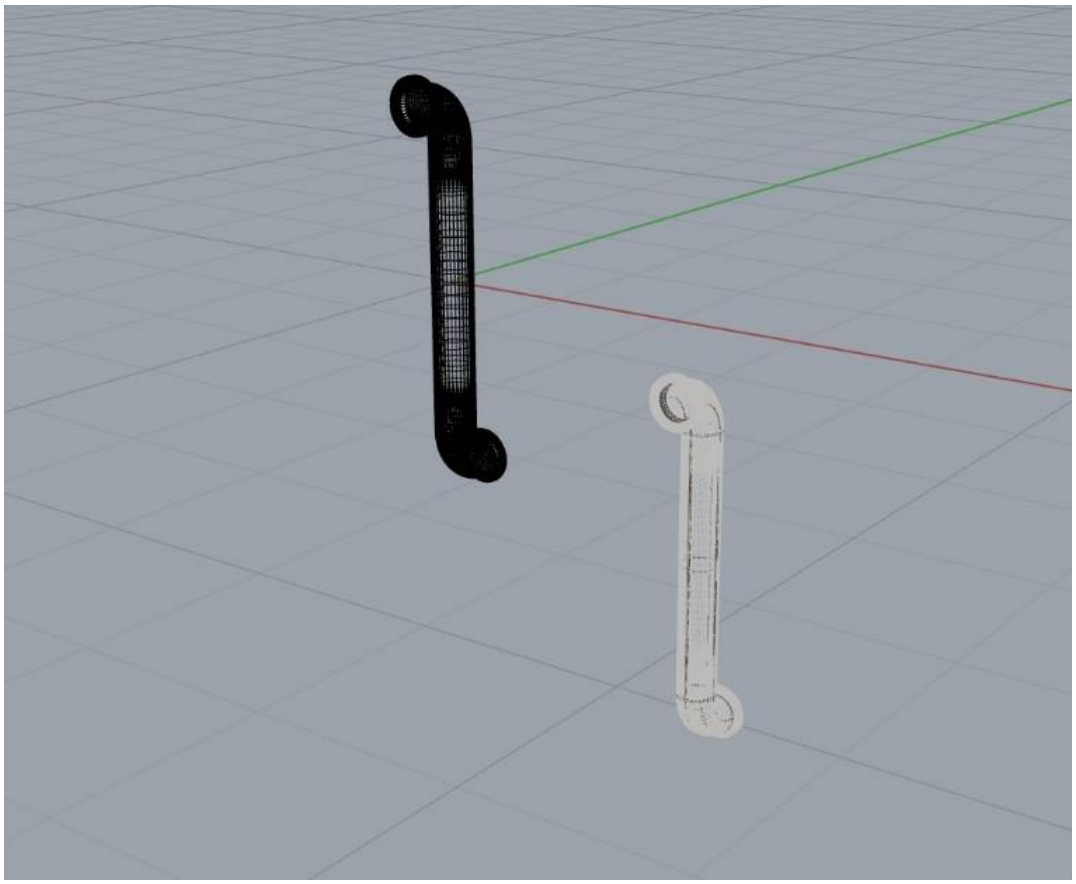


Figure 50*:* Meshed model is translated to the axes' origin

In the following image, the environment of the Unity project is presented. The imported model has been rotated around the three axes and scaled along them manually using the top left bars. The bounding box is automatically created around each object of the model and after capturing a screenshot, the coordinates of the top left and bottom right edges of the box, which specify the limits of the bounding box of the objects are exported to the annotaion file which will be used during training. The annotation file is automatically saved to a folder of the user's choice, specified in the "Folder name" field.

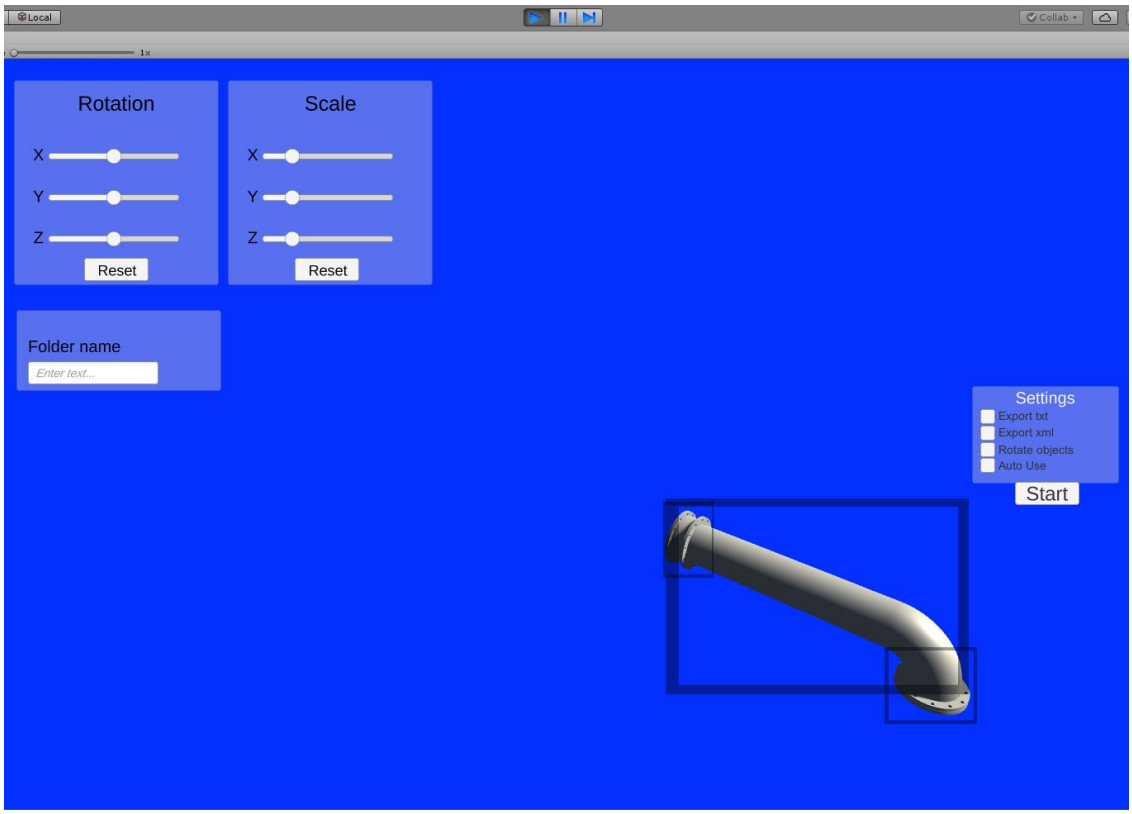


Figure 51: Environment of Unity project – Bounding box created around each component of the model

After implementing the above steps, a custom object detection model was trained through *Darkflow* using simple pipe images, such as straight, bent and circular pipes, generated by importing the models to the Unity project.

Below is presented the graph of the average loss of the trained network against iteration number. The average loss started from approximately 3500 and ended up to about 0.2. It was stopped at 1500th iteration, since the the model had already converged and we wanted to avoid creating an overfitted model that would not be able to generalize to objects that it had not processed. As it can be seen, after about 100 iterations, the average loss had already dropped below 5. Training lasted about 8 hours until it was stopped.

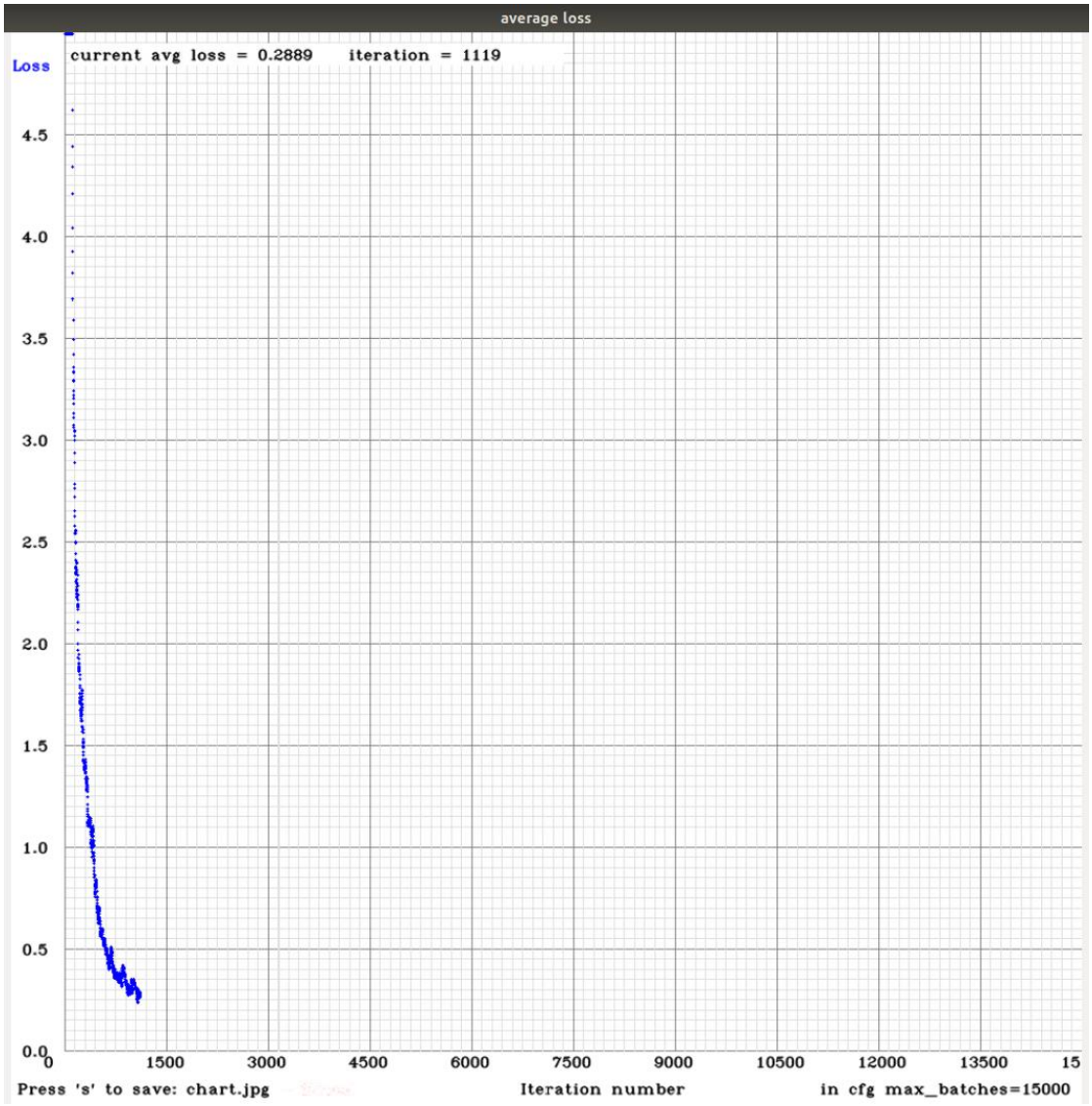


Figure 52*:* Average loss of network against iteration number

Below are some successful testing results of the trained model. The model was tested in detecting both simple pipes images which were also used in training, as well as detailed piping systems images. The trained model performed well in detecting basic components such as straight pipes but was not able to detect multiple components in a single image.

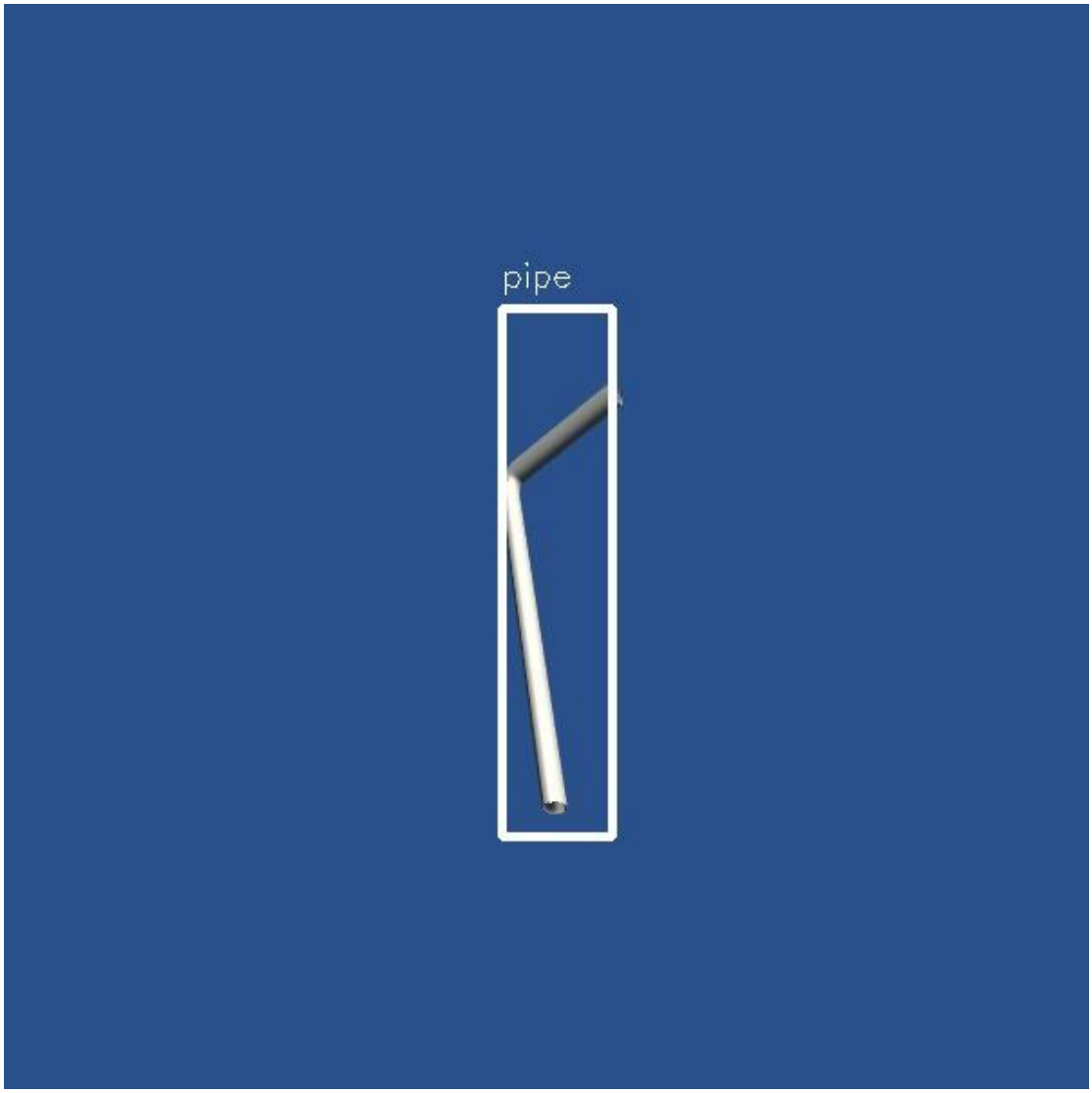


Figure 53*:* Trained model successfully recognizes a single bent pipe

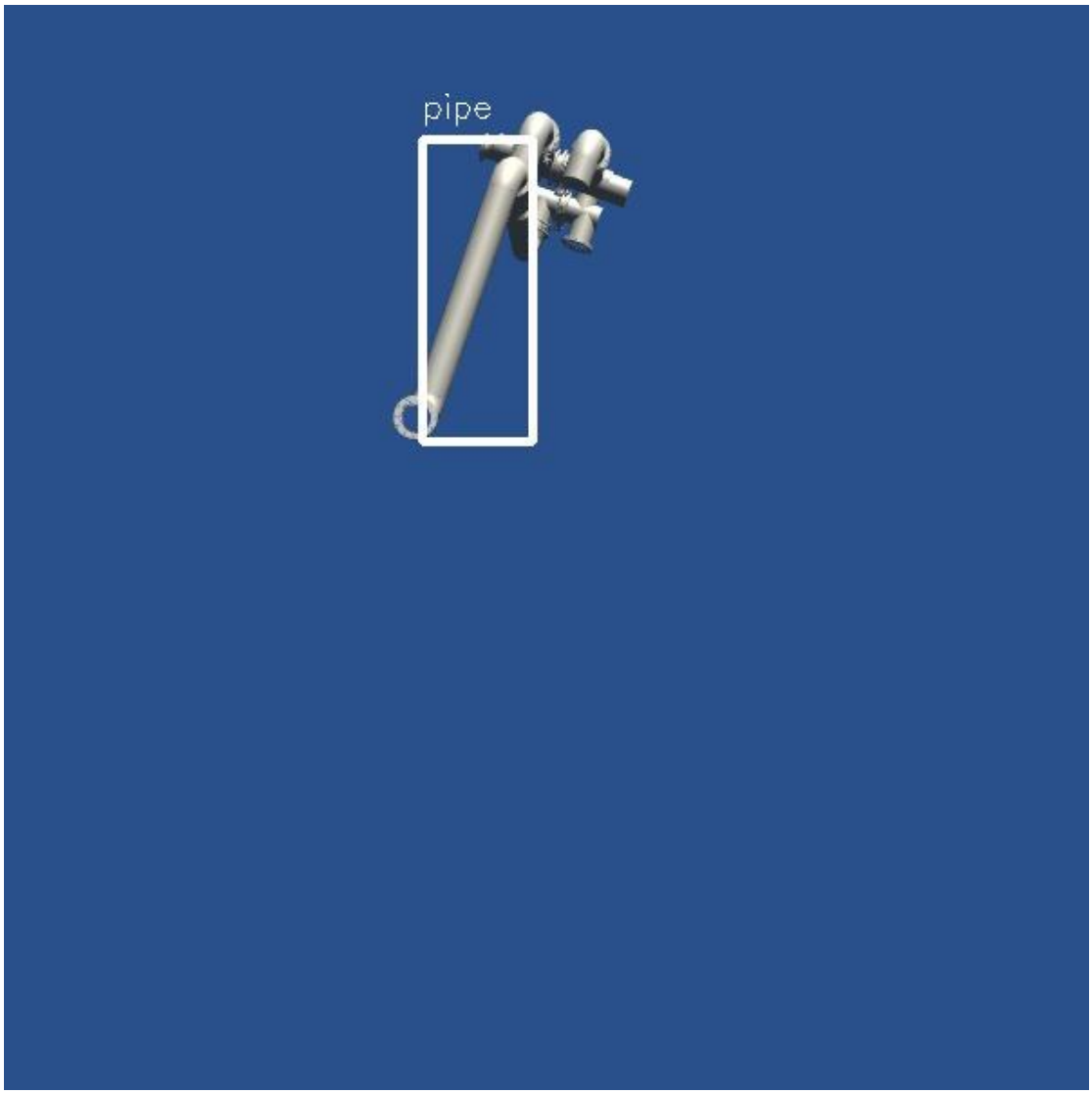


Figure 54*:* Trained model successfully recognizes a pipe in a rotated detailed piping system

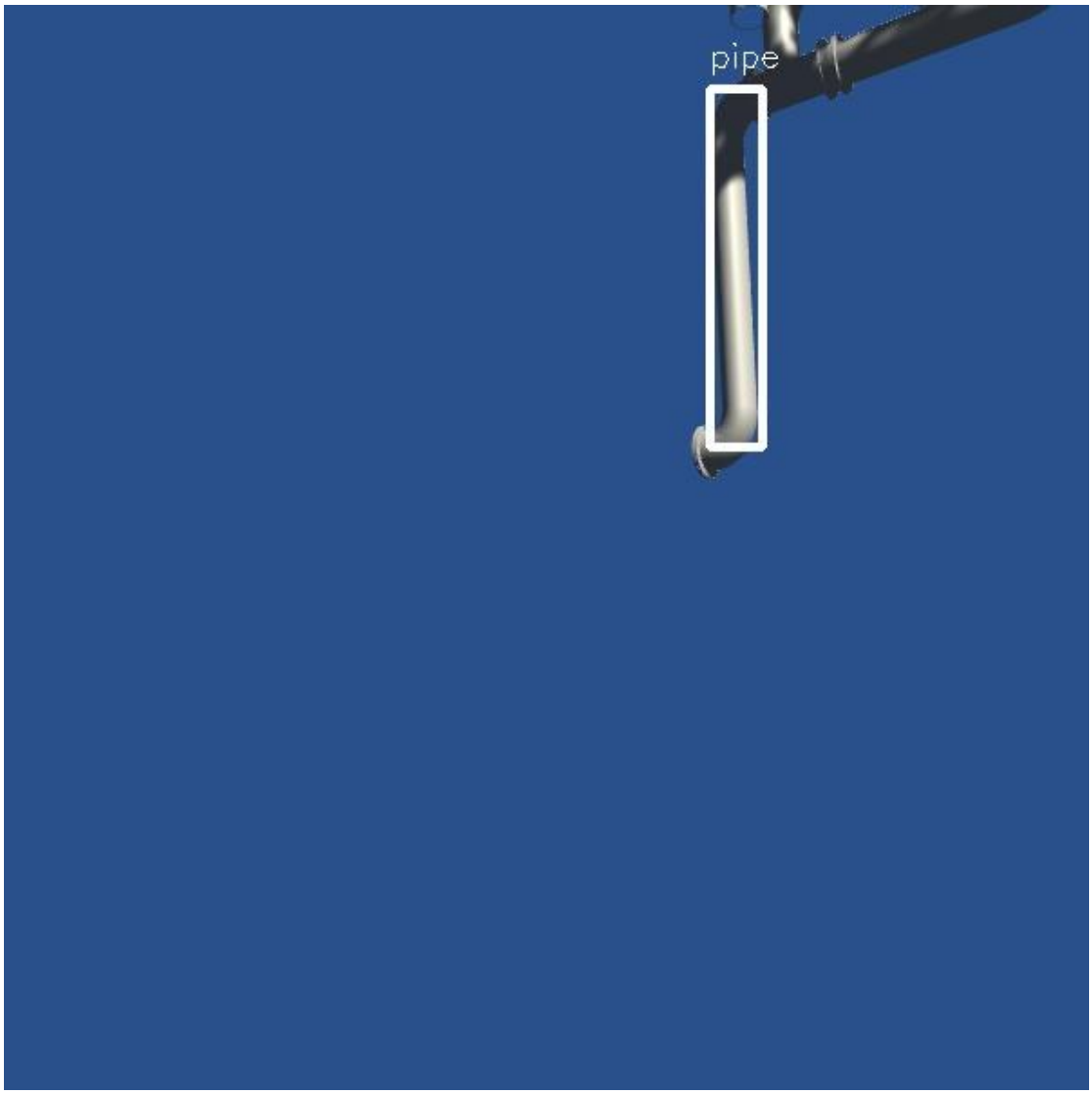


Figure 55*:* Training model successfully recognizes a pipe in a rotated and truncated piping system

5.3.3 Training with model components images

In the effort of being able to train a robust model that can detect multiple different components in a more detailed and complex piping model, it was necessary to make some advancements to the training set, as well as to the Unity project. Specifically, our goal was to generate a dataset that comprises of the basic components of a piping system in order to train the model to detect and recognize these parts in an image. To do so, we imported some detailed piping 3D models into the environment of Rhinoceros 3D and edited them. In particular, we isolated each component of the piping system, meshed it and exported it as an object (.obj format). At a later stage, the exported components were imported into the Unity project to capture screenshots of the models and create their annotation files for training, similarly to the procedure that was done before. Eventually, a dataset comprising of approximately 600 images was generated and was used to train another custom model.

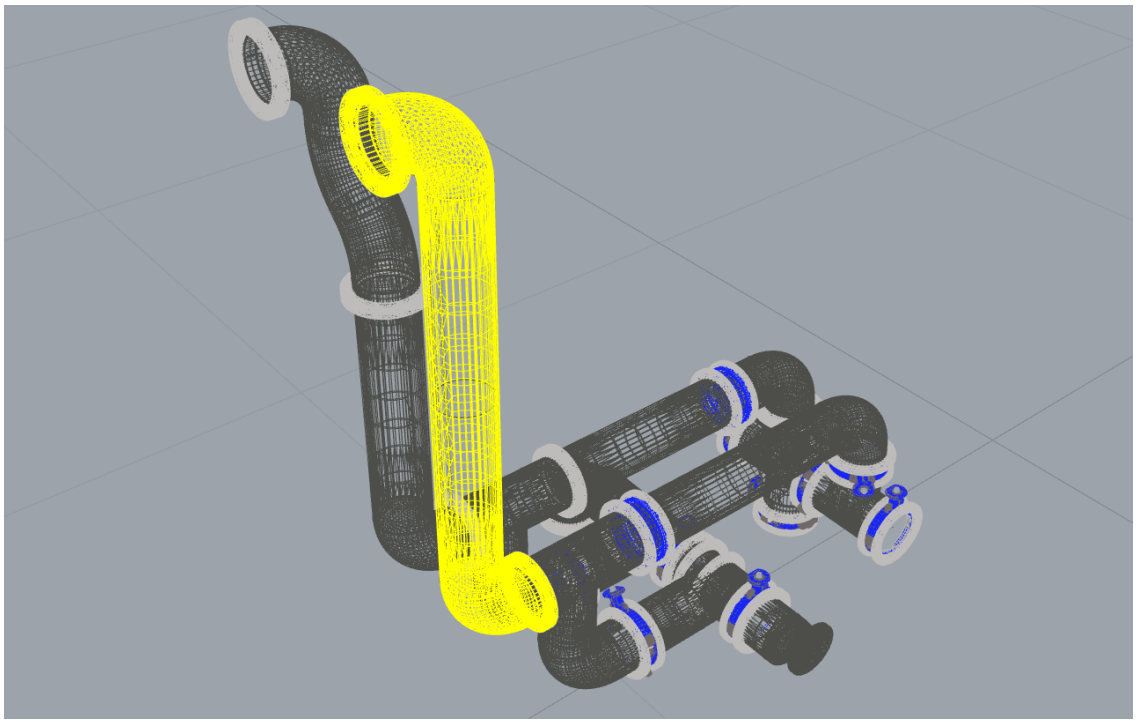


Figure 56*:* Meshed component of a detailed piping system

5.3.4 Training with multiple objects

In order to be able to detect multiple objects of various classes in a single image, some improvements were required to be applied on the dataset, as well as on the Unity project. More specifically, some changes were made so that each object in the Unity project was included to a class (either flange or pipe). When capturing screenshots and generating the annotation files (.txt) objects that were contained to the “flange” class would be assigned to 1 and objects that were in the “pipe” class were assigned to 0. Having changed the number of classes to 2 in the configuration file, when processing an image, YOLO can distinguish which objects is assigned to which class. For example, the annotation file has the following format:

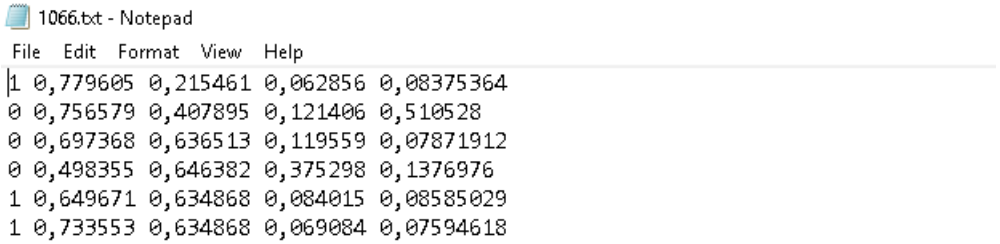


Figure 57: Annotation file (.txt) of image with multiple classes

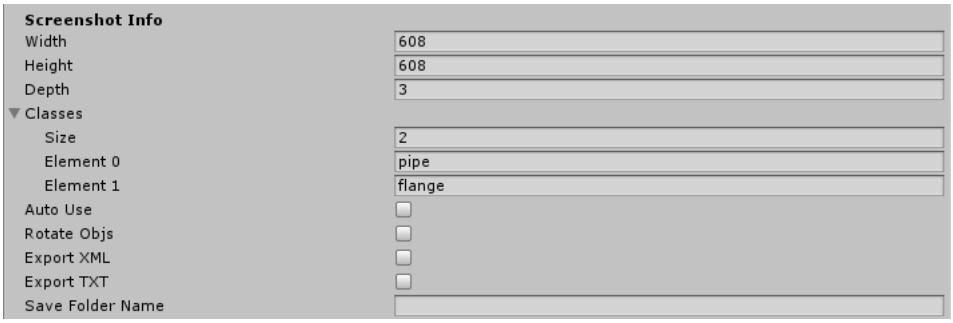


Figure 58: Screenshot information of image in the Unity Project. Width, Height and types of Classes are specified

In the above example, the training image *1066.jpg* contains objects of two different classes: 3 flanges (assigned to 1) and 3 pipes (assigned to 0). Each line specifies the object’s class, the coordinates of the centre of the bounding box and the width and height of the bounding box relative to the image width and height. All screenshots were captured in 608 x 608 resolution, which although is a relatively small value, it requires less computational power to be processed during training.

Moreover, some advances were applied to the codes, in order to change the background colour in a random way after capturing every screenshot in the Unity project. This was done, in order to prevent YOLO from linking certain objects to a particular background colour. By

applying such change, our trained system becomes more robust, since it can perform object detection regardless of the font colour.

The custom object detection model was trained with 800 images of simple piping systems, comprising of up to 6 six different components. The trained model was tested on images it had not processed before and had excellent performance in detecting all components in simple piping systems.

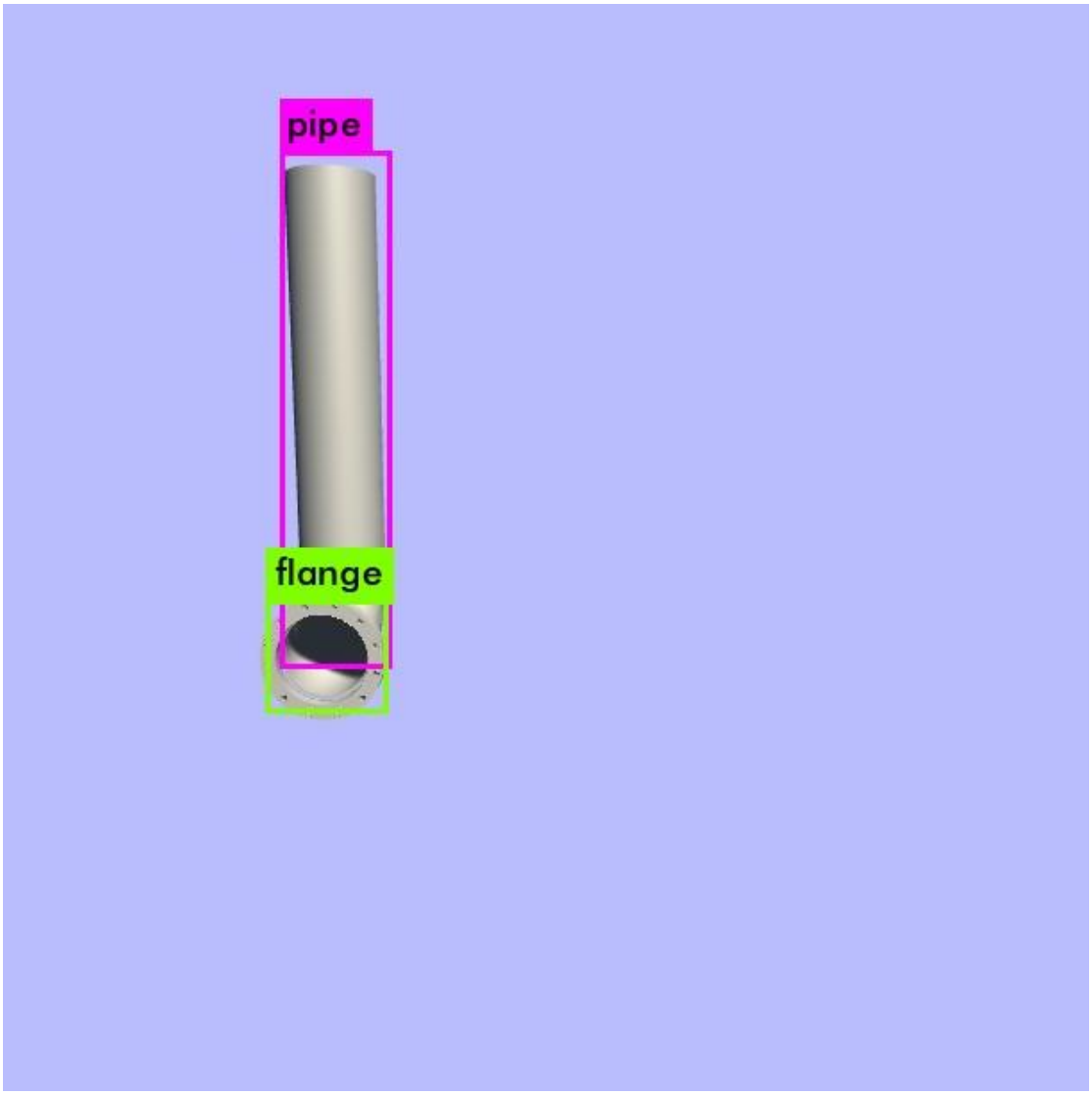


Figure 59*:* Custom model detects perfectly all components of a simple piping system (pipe & flange)

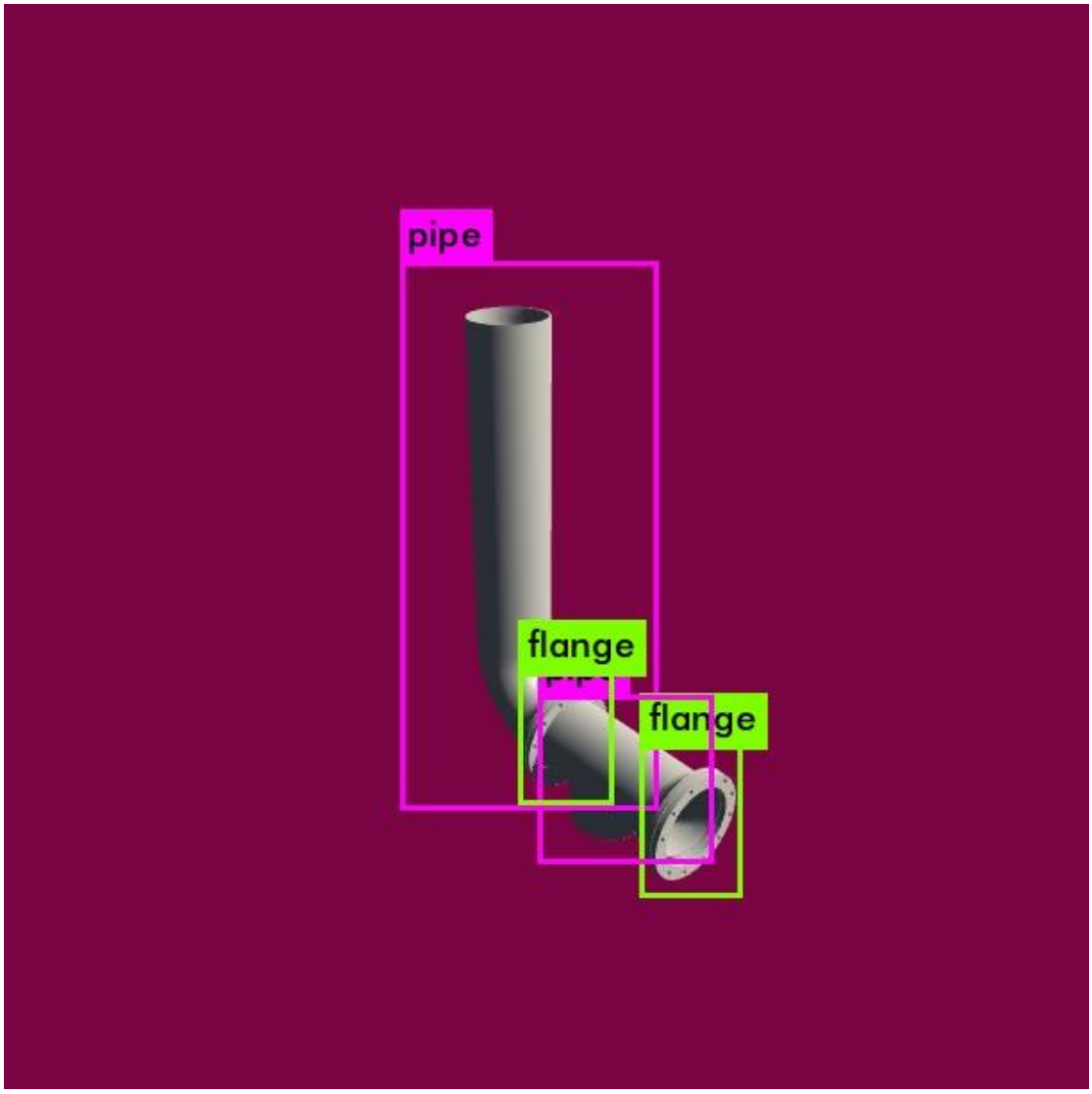


Figure 60*:* Custom model detects perfectly all components of the piping system (pipes & flanges)

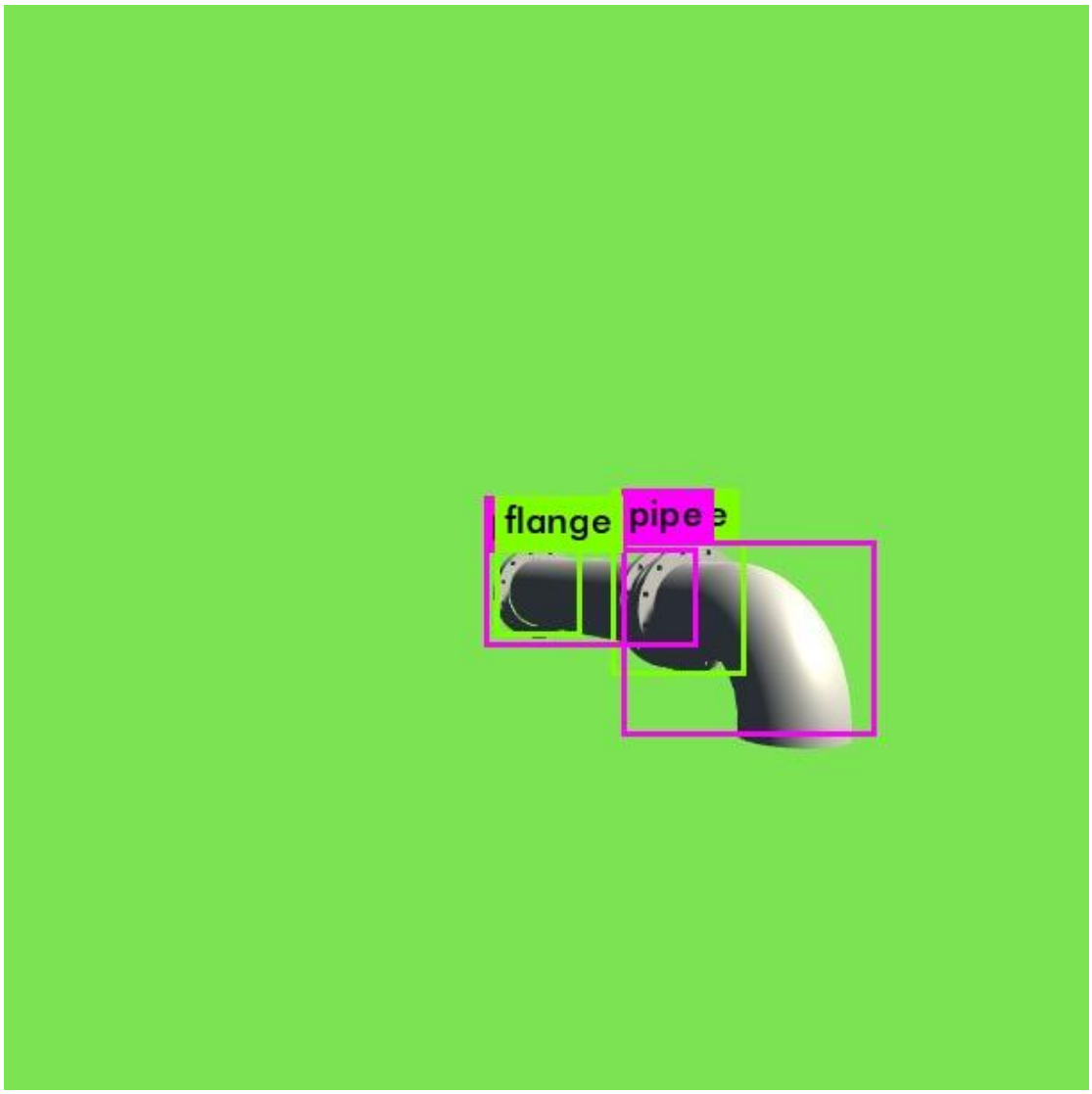


Figure 61*:* Custom model detects perfectly all components of the piping system (pipes & flanges)

At a later stage, our custom trained network which had been trained using simple components of piping systems, was tested on more complicated systems. As it can be shown, although it had not considered information about such complex systems, it performed really well, by detecting almost every component. This means that our trained model is able to generalize well and can perform detection to objects it has never processed before.

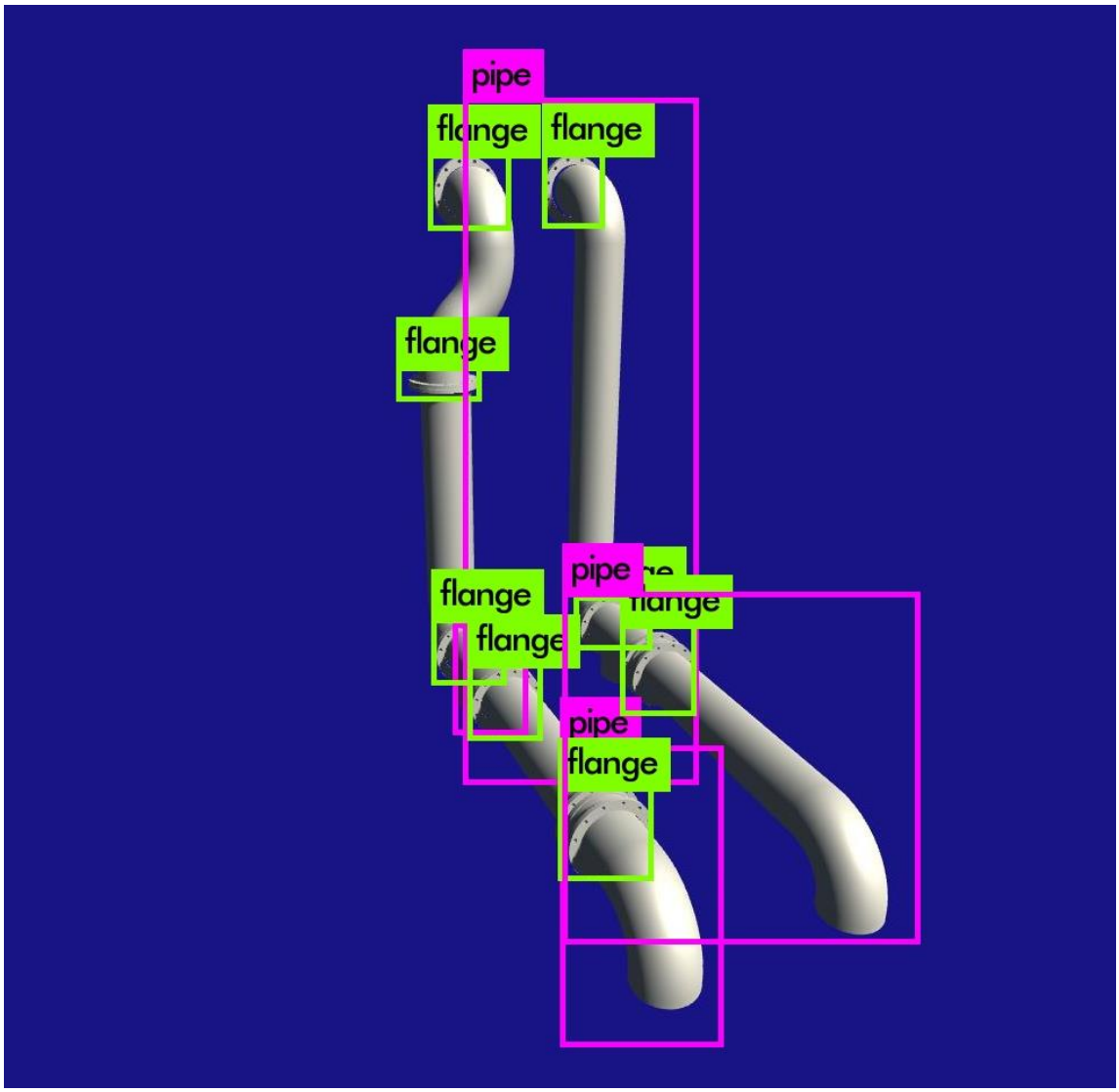


Figure 62*:* Custom model detects almost every component of a complex piping system (pipes & flanges)

5.3.5 Training with different materials

In order to train a more robust system that can perform well in real situtations, some imporvements were made to the training dataset. Specifically, based on the existing Unity project, some additional models were created which are intended to simulate real conditions of piping systems in a ship's engine room. To do this, we had to create some new materials and textures and apply them to the piping models. Particularly, materials such as "metallic", "stainless steel", "aluminium" and "rusted steel" were manually created. Furthermore, in order to replicate as accurately as possible real conditions of pipes in the engine room, different colours were applied to the models. Specifically, the applied colors were chosen in order to respond to the deafult pipeline colors on a ship. For example, green color for sea water pipes, blue for fresh water, black for fuel pipes and silver color for steam pipes. Additionaly, materials were created in a way that they had reflection of light in order to simulate the intense lighting inside a ship's engine room.

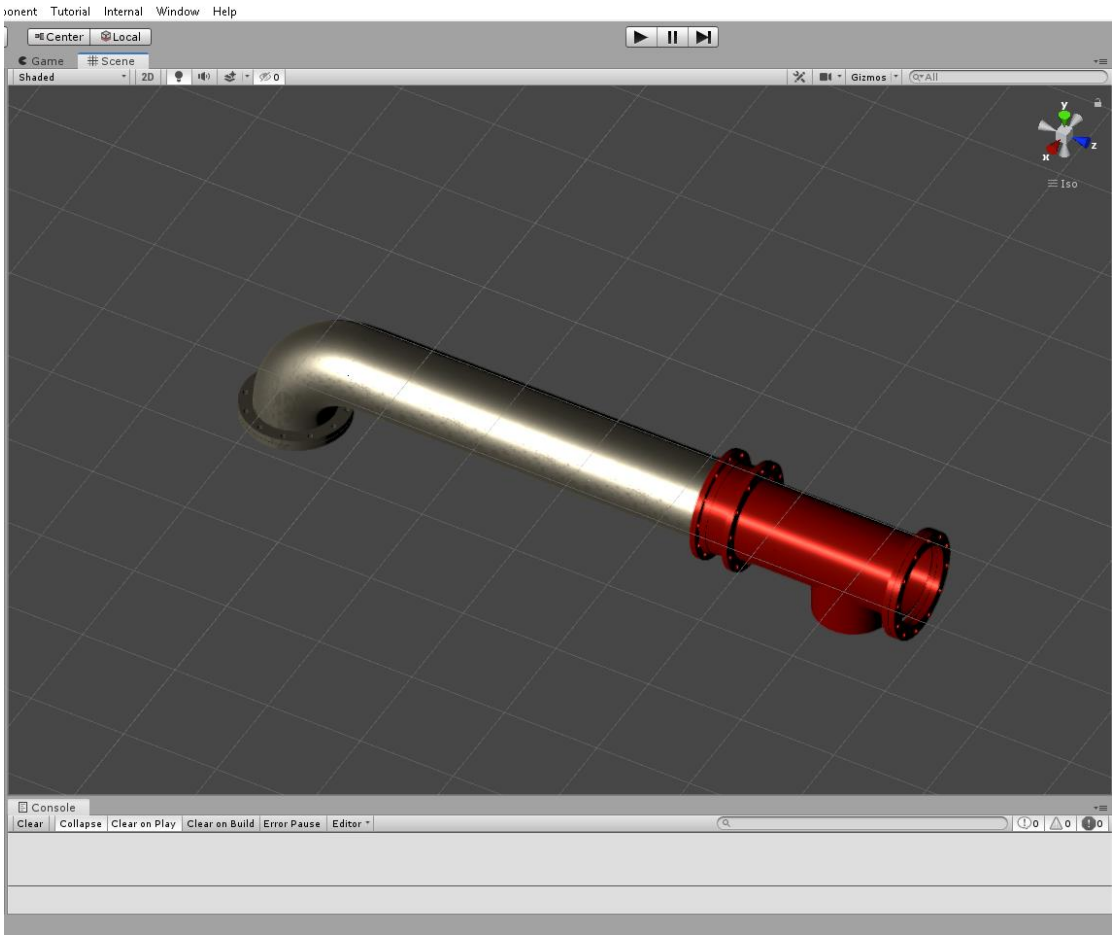


Figure 63: Model comprising of "stainless steel" and "aluminum" materials

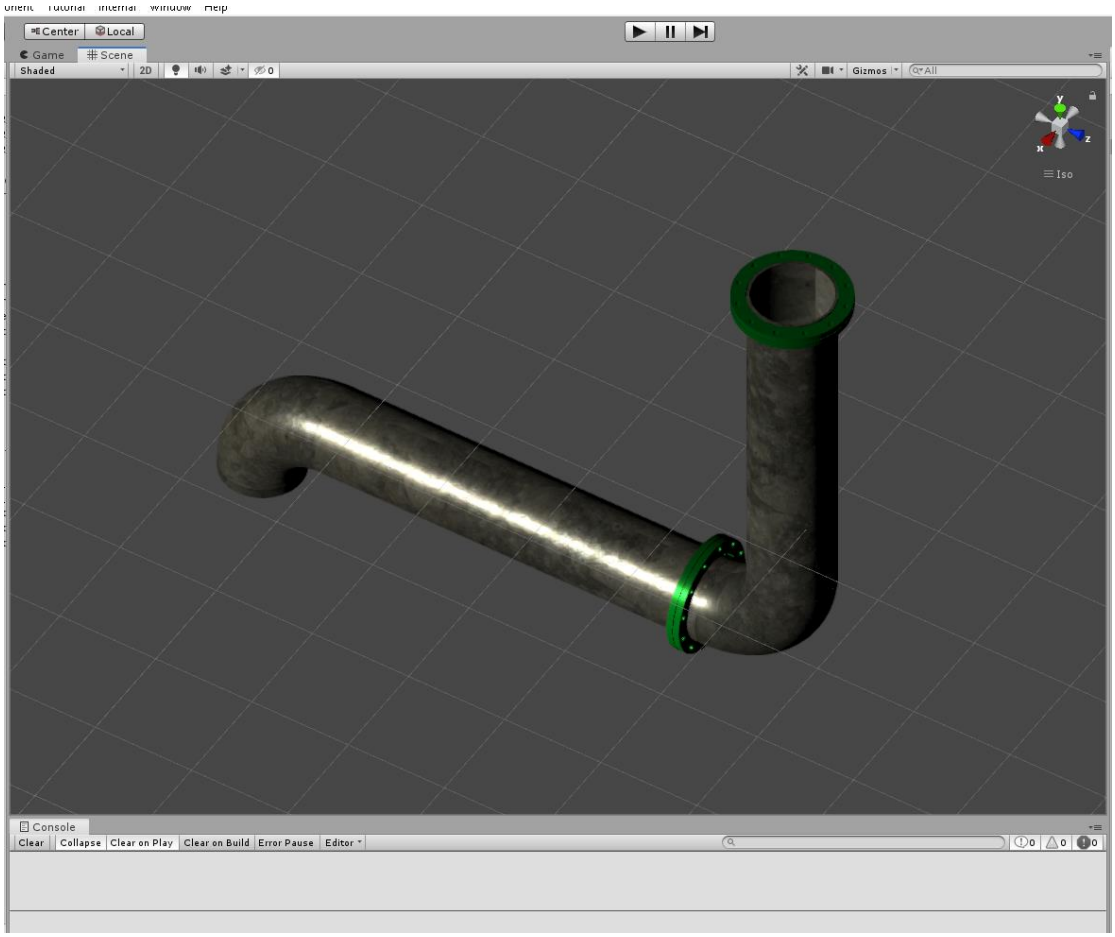


Figure 64*:* Model comprising of “metallic” and “aluminum” materials

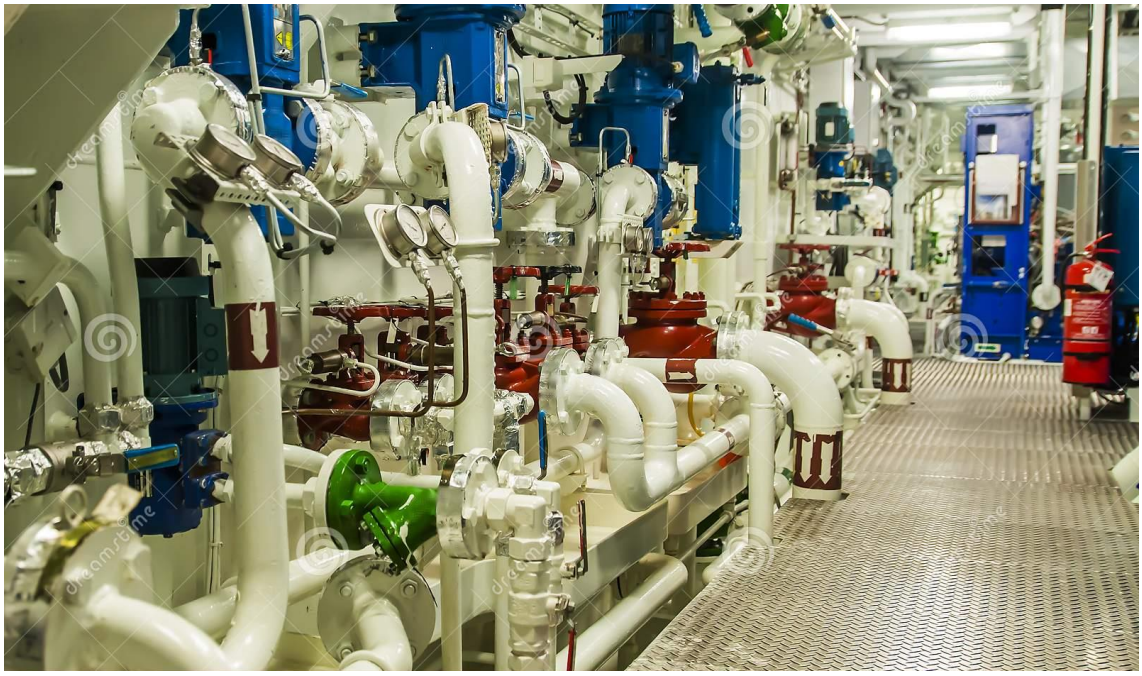


Figure 65*:* Piping system inside ship’s engine room

At this point, the trained network which was trained with images of pipes and flanges with the custom materials, was tested on images of real piping systems in order to test its accuracy and performance on real conditions. As it can be seen, our custom model has managed to detect and accurately recognize most of the components of a complex piping system. Of course, performing detection on images of a real environment is much more challenging, since it includes various components that the model has not been processed through its training. Furthermore, real conditions also contain noise, such as various small objects in front of pipes (e.g. cables, valves, other machinery equipment) and also defects. However, our trained model has performed quite well on real conditions.

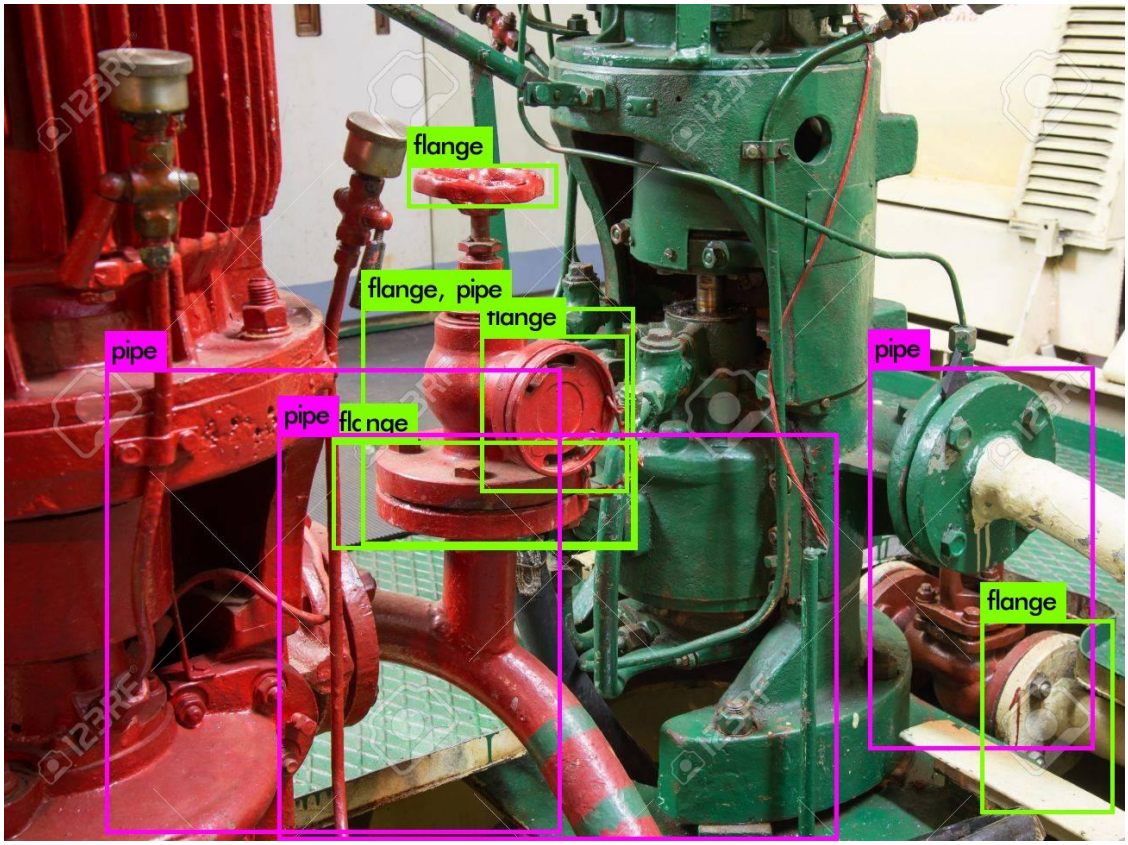


Figure 66: Custom model detects most of the components of a real complex piping system (pipes & flanges)

5.3.6 Training with additional components

In the last part of our case study we were involved with adding objects of further classes to our custom object detection model in order to be able to detect accurately all basic components of piping systems. More specifically, the network was trained in order to detect valves and also distinguish straight pipes from elbow pipes. Some results of the training are presented below:

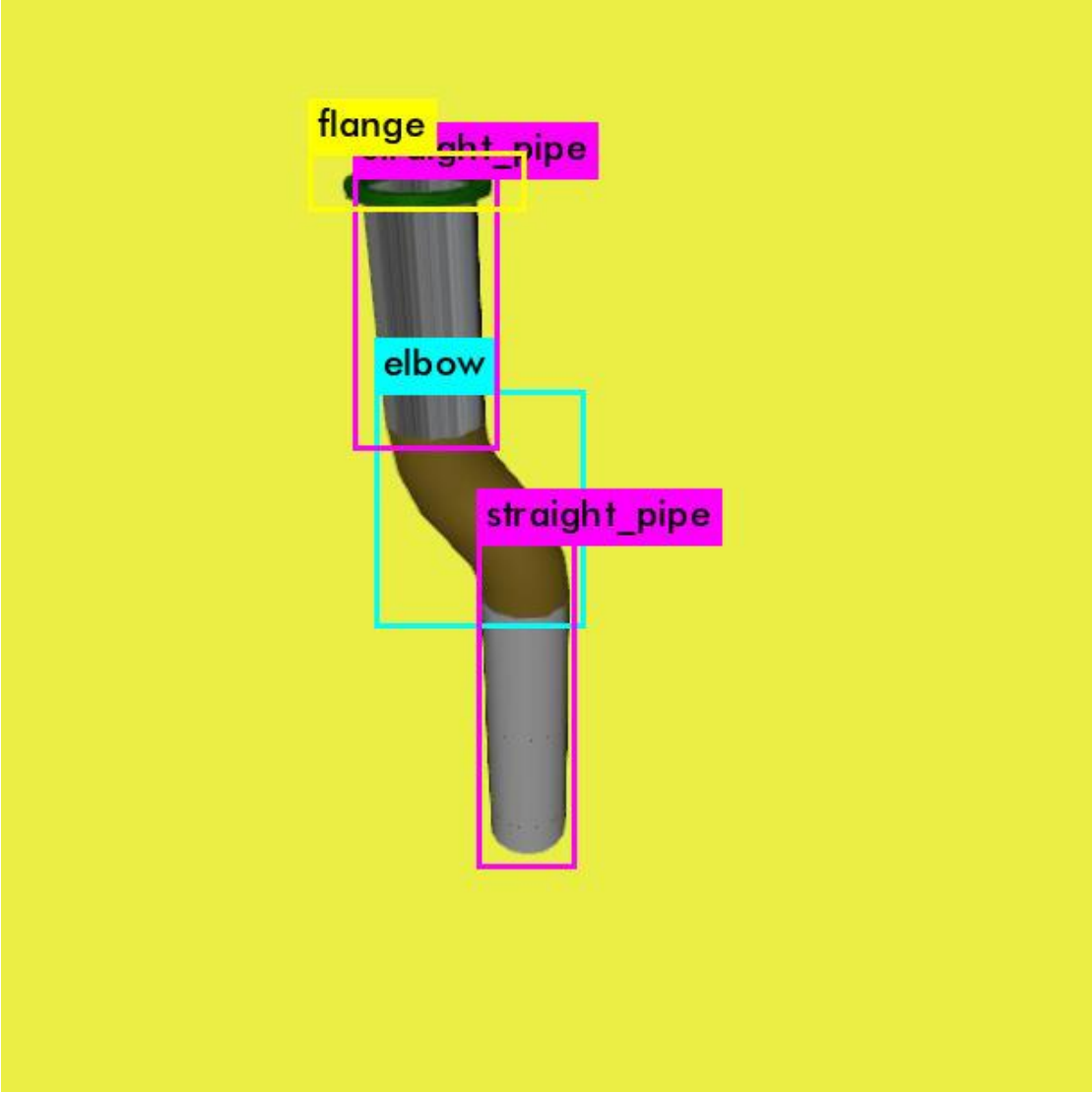


Figure 67: Custom model detects all components of piping system (flange, straight pipe, elbow)

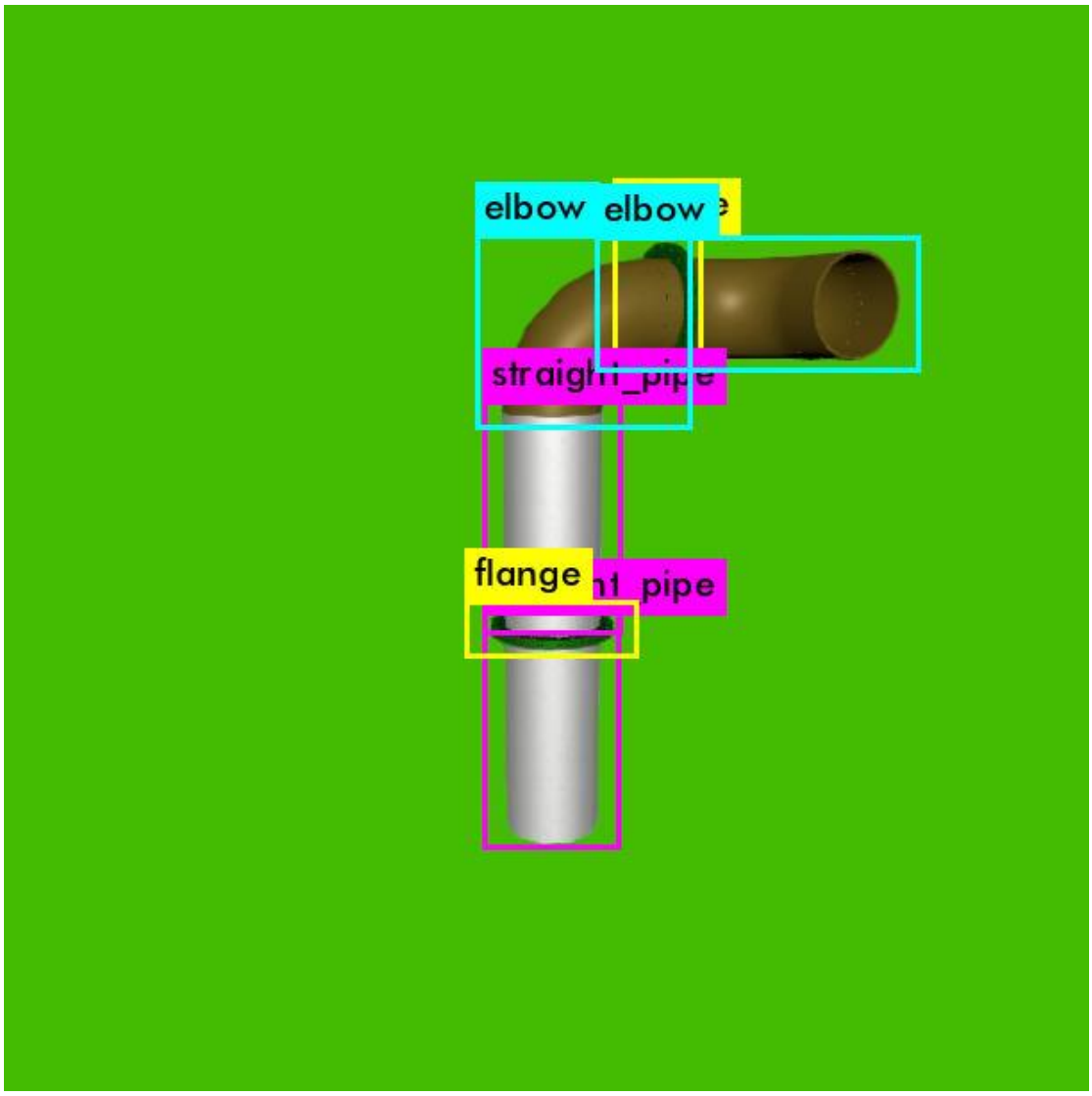


Figure 68: Custom model detects all components of piping system (flange, straight pipe, elbow)

Lastly, the custom object detection model was tested on images of real condition piping systems. Specifically, some images were captured from the Laboratory of Marine Engineering at the School of Naval Architecture and Marine Engineering of NTUA. Some of the results are shown below:

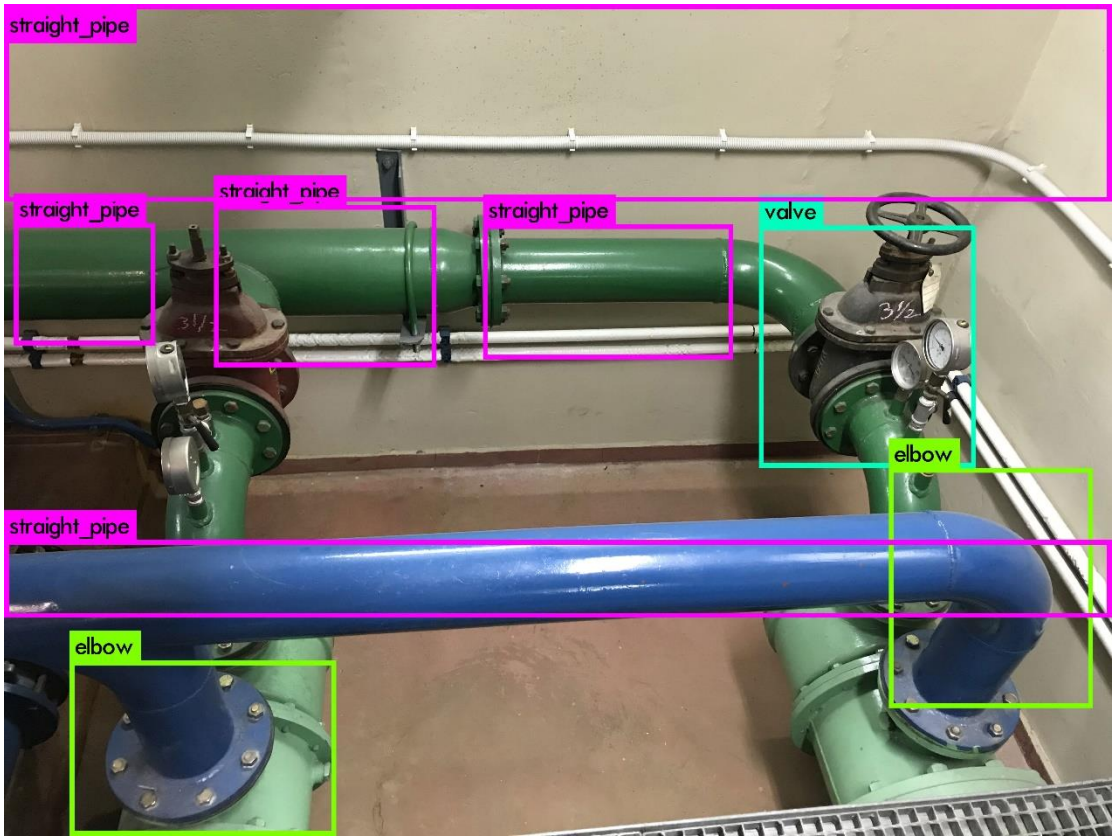


Figure 69: Custom model detects accurately almost every component of the real condition piping system

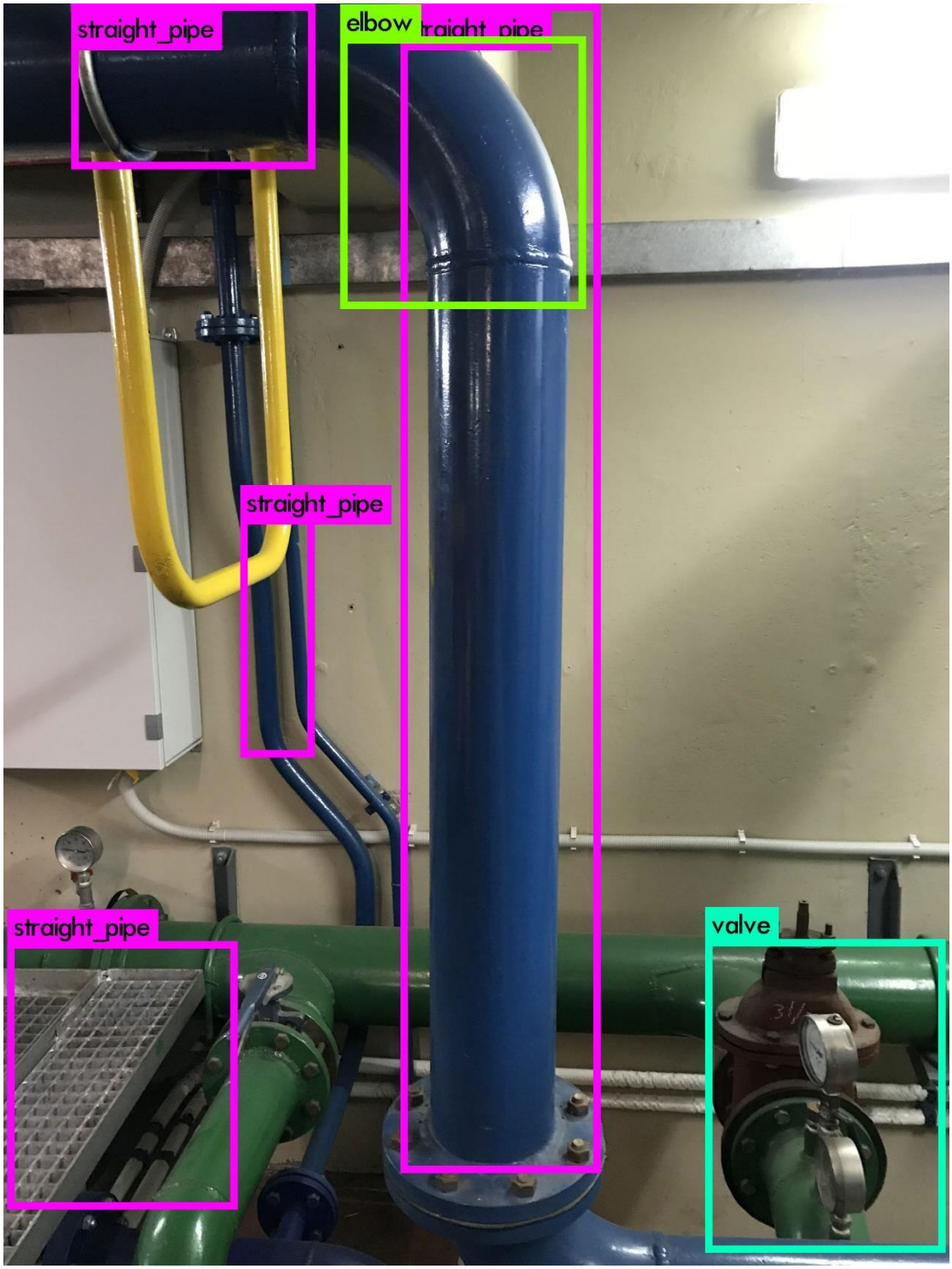


Figure 70: Custom model detects most of the basic component of real condition piping system

Although the results do not seem to be perfect, it is obvious that the custom model has successfully detected most of the basic components of the real condition piping systems in a noisy environment.

6. Conclusions – Future Work

6.1 Conclusions

In the present thesis, a robust object recognition framework has been developed to identify and localize mechanical elements of complex ship piping networks. The approach in the current problem has been based on object recognition in 2D images using convolutional neural networks. The presented results demonstrate that the current diploma thesis has achieved its goal which has been the proper training of machine learning algorithms and the recognition of piping elements. Specifically, it has succeeded on developing an efficient CNN-based object recognition framework in the area of the manufacturing industry capable of detecting accurately all basic components of complex piping systems. Furthermore, it has also succeeded on proposing a method of almost automating the training process of the CNN, as well as ensuring the validity and sufficiency of the training data, which until today has been the most time-consuming task of constructing a robust and efficient object recognition framework. This offers the option of building a custom model capable of detecting objects of multiple classes of one's choice with no limitations regarding the number, size, or complexity of objects.

6.2 Future Work

Of course, this proposed method is still a preliminary approach in the field of object recognition and there is huge room for improvement. However, through making some advancements and additions to the existing model, this can be significantly enhanced in order to potentially play an integral part in problems, such as exact identification of mechanical parts, 3D object recognition and faulty parts detection. Below some proposals for future work that can advance the findings of the present thesis are stated.

Firstly, aside from detecting components of complex piping systems, the current network can be trained to detect any kind of objects. Specifically, by importing into the proposed Unity project 3D models of different kind of objects (such as pumps, or engines), a large dataset can easily and accurately be generated in order to train a custom network. Therefore, a possible advancement to the current network would be to train a custom model that will be able to detect objects, such as, pipes, different kinds of engines (main engine, generators, etc), boilers and other machinery equipment. This improvement would help into being able to describe accurately the content of a ship's engine room.

Another impactful advancement to the existing model would be to further automate its basic steps. More specifically, some manual procedures required in the current network could be semi or fully automated in order to improve the speed and efficiency of the model. Particularly, the processes of generating the dataset and training the custom object detection model using Darknet can be automated. These tasks involve collecting all images and annotation files, saving them into specific directories, creating some additional files required for training, such as files that specify the labels of objects, or the configuration files. Of course, these are tasks that demand manual intervention. By automating all these

procedures, a flexible, fast and automatic network that can be created. This means that by providing an accurate dataset to the network it does not require any further manual processing and it can effectively train a custom model.

Last, at an advanced stage, the current network could be upgraded into performing 3-D object recognition in point clouds. This means training a CNN with 3-D data of a ship's engine room for example, in order to perform object detection on point clouds obtained by laser scans. Although this seems as a very complicated and challenging task, the custom object detection model proposed in the current thesis could possibly serve as a hybrid approach in this problem. Particularly, performing object recognition into 2-D images can help isolate a small region of a dense point cloud and try to perform recognition to the 3-D data in a limited 3D point cloud space. Having performed 3D object recognition on point clouds in a space such as a ship's engine room and for example a pipe is identified, assuming that the CAD model of the object is available, it can be registered in the 3D space. This capability would definitely be of great importance, since it would help to automatically generate a virtual 3D model of every desired space. That would have various applications, such as improving maintenance of machinery, path planning for retrofitting new components and diagnosing and preventing defects.

7. Literature – References

[1] Ross B. Girshick, Jeff Donahue, Trevor Drrell, Jitendra Malik. (2014). “Rich feature hierarchies for accurate object detection and semantic segmentation”. URL: https://arxiv.org/pdf/1311.2524.pdf

[2] Ross B. Girshick. (2015). “Fast R-CNN” URL: http://arxiv.org/abs/1504.08083.

[3] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. (2015). “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Net-works”. URL: http://arxiv.org/abs/1506.01497.

[4] Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick. (2018). “Mask-RCNN”. URL: https://arxiv.org/pdf/1703.06870.pdf

[5] Yue Wu, Yinpeng Chen, Lu Yuan, Zicheng Liu, Lijuan Wang, Hongzhi Li, Yun Fu. (2019). “Double-Head RCNN: Rethinking Classification and Localization for Object Detection. URL: https://arxiv.org/pdf/1904.06493.pdf

[6] Joseph Redmon , Santosh Divvala, Ross Girshick , Ali Farhadi. (2016). “You Only Look Once: Unified, Real-Time Object Detection”. URL: https://arxiv.org/pdf/1506.02640.pdf

[7] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger” URL: http://arxiv.org/abs/1612.08242.

[8] Joseph Redmon, Ali Farhadi. (2018). “YOLOv3: An Incremental Improvement”. URL: https://arxiv.org/pdf/1804.02767.pdf

[9] Ezeddin Al Hakim. (2018). “3D YOLO: End-to-End 3D Object Detection Using Point clouds”. URL: http://www.nada.kth.se/~ann/exjobb/ezeddin_alhakim.pdf

[10] Martin Simon, Karl Amende, Andrea Kraus, Jens Honer, Timo Samann, Hauke Kaulbersch, Stefan Milz, Horst Michael Gross. (2019). “Complexer-YOLO: Real-Time 3D Object Detection and Tracking on Semantic Point Clouds”. URL: https://arxiv.org/pdf/1904.07537.pdf

[11] Jiuxiang Gua, Zhenhua Wangb, Jason Kuenb, Lianyang Mab, Amir Shahroudyb, Bing Shuaib, Ting Liub, Xingxing Wangb, Li Wangb, Gang Wangb, Jianfei Caic, Tsuhan Chenc. (2017). “Recent Advances in Convolutional Neural Networks”. URL: https://arxiv.org/pdf/1512.07108.pdf

[12] Zhiang Chen. (2017). “DeeP-Learning Approaches to Object Recognition from 3D Data”. Master Thesis. URL: https://etd.ohiolink.edu/!etd.send_file?accession=case1496303868914492&disposition=inline

[13] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. (2016). "Why Should I Trust You? Explaining the Predictions of Any Classifier". URL: https://www.kdd.org/kdd2016/papers/files/rfp0573-ribeiroA.pdf

[14] David Stutz. (2014). "Understanding Convolutional Networks". URL: https://davidstutz.de/wordpress/wp-content/uploads/2014/07/seminar.pdf

[15] Weibo Liua , Zidong Wanga, Xiaohui Liua, Nianyin Zengb, Yurong Liuc, Fuad E. Alsaadi. "A survey of Deep Neural Network Architectures and Their Applications". URL: https://bura.brunel.ac.uk/bitstream/2438/14221/1/FullText.pdf

[16] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. (2012). "ImageNet Classification with Deep Convolutional Neural Networks". URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. (2014). "Dropout: A Simple Way to Prevent Neural Networks form Overfitting" URL: http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

[18] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, Xidong Wu. (2019). "Object Detection with Deep Learning: A Review". URL: https://arxiv.org/pdf/1807.05511.pdf

[19] Ian Goodfellowe, Yoshua Bengio, Aaron Courville. (2016). "Deep Learning (Adaptive Computation and Machine Learning series)"

[20] David Kriesel. (2005). "A Brief Introduction to Neural Networks"

[21] Simon Haykin. (2009). "Neural Networks and Learning Machines", Third Edition

[22] Kevin Gurney. (1997). "An Introduction to neural networks"

[23] Laurene Fausett (1994). "Fundamentals of Neural Networks – Architectures, Algorithms and Applications"

[24] Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola. (2019). "Dive into Deep Learning"

[25] Fei-Fei Li & Justin Johnson & Serena Yeung. (2017). Stanford University. Lectures 1-7