



NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF NAVAL ARCHITECTURE AND MARINE ENGINEERING
DEPARTMENT OF MARINE ENGINEERING

Propulsion Shafting Arrangement Modeling from Mechanical Drawings using Deep Learning and YOLOv8

DIPLOMA THESIS

Naoum Panteleimon

Examination Committee Members

Christos Papadopoulos, Associate Professor NTUA (Supervisor)

George Papalambrou, Associate Professor NTUA

Konstantinos Anyfantis, Assistant Professor NTUA

Athens, March 2023

Acknowledgements

As I conclude my undergraduate studies, I would like to express my heartfelt appreciation to all those who supported me during this journey. I am particularly grateful to Dr. Christos Papadopoulos, my supervisor, for giving me the opportunity to undertake this thesis under his guidance, and for equipping me with the resources to explore and broaden my knowledge in the field of Artificial Intelligence. This knowledge has proven to be invaluable when combined with my technical background in Marine Engineering. I would also like to extend my gratitude to George Rossopoulos, whose guidance and insightful suggestions were instrumental in the development of this thesis, resulting in a robust and precise outcome. Finally, I wish to thank my family for their unwavering support and encouragement, especially during the most challenging times.

Abstract

The modern maritime industry is heading towards a digitalized future which encompasses sustainable technological solutions which will automate operations in all aspects of the maritime sector. The role of the Marine Engineer as a professional encompasses the knowledge and the critical thinking required to solve problems and overcome technical difficulties. One of them could be the ability to read mechanical drawings.

The Marine Engineer is capable of identifying mechanical objects on a drawing, for example the position of the vessel's Main Engine on a General Arrangement Plan and matching the identified geometries with their respective dimensions. Thus, the engineer is able to extract the necessary data from a mechanical drawing which are needed to solve further issues and make optimal decisions based on them.

In the present thesis, we evaluate the feasibility of developing an Artificial Intelligence algorithm, which will employ the use of Deep Neural Networks, Object Detection and OCR Techniques in order to understand the Propulsion Shaft Arrangement of a vessel, identify and classify key objects in it, identify and extracting dimensions and correlate their positions with their respective dimensions.

The purpose of this algorithm is to extract the necessary data to generate a digital model of the vessel's shafting arrangement capable to undergo further processing such as being incorporated as a module in a Shaft Alignment software. Such an integration will potentially accelerate the Shaft Alignment process instead of manually entering the data into it and further lead a path in scale-up projects such as shaft alignments of multiple vessels in a fleet.

Multiple Object Detection algorithms and different OCR models have been proposed throughout computer vision applications such as R-CNN, SSD, RetinaNet, thus we need to employ the necessary metrics to evaluate the most accurate fitted to our application. On this particular thesis, the YOLOv8 (You Only Look Once – Version 8) has been employed, a cutting-edge, state-of-the art model (released at January 2023), which is an improvement of the previous YOLO versions introducing new features and improvements to further enhance performance and accuracy. Additionally, a large database of Shaft Arrangement drawings was generated by performing augmentations, emulating noise, cropping and minimal rotations of the image. We generated the necessary bounding boxes for the classes including several objects necessary to identify such as the Propeller, Flanges, the Stern Tube Bearings, the Shaft Bearing as well as the Propeller Shaft, the Intermediate Shaft and their dimensions. The chosen model was trained for 200 epochs and achieved an average of all classes Precision of 92.4%, Recall of 94.2% and mAp of 94%.

Contents

1	Introduction	1
1.1	Introduction to Artificial Intelligence	1
1.2	The role and ability of Marine Engineers	1
1.3	Our scope of work	2
2	Literature Review	3
2.1	Object Detection using SVMs & Edge Detection	3
2.2	Object Detection using Convolutional Neural Networks	4
2.2.1	Implementation of R-CNN Networks	4
2.2.2	Implementation of Single Shot Multibox Detectors (SSDs)	4
2.2.3	Implementation of YOLO algorithms in Object Detection	5
2.2.4	Regarding YOLOv8	5
3	Theoretical Background	6
3.1	Introduction to Computer Vision	6
3.2	Image Filtering Techniques	10
3.3	Machine Learning and Deep Neural Networks	16
3.3.1	Machine Learning	16
3.3.2	Overview on Neural Networks	25
3.3.3	Neural Network Architecture and Principles	29
3.3.4	Training Neural Networks and Backpropagation	31
3.3.5	Neural Network Performance Evaluation	34
3.3.6	Convolutional Neural Networks	38
3.3.7	Object Detection using CNNs	45
3.3.8	YOLO Versions and Improvements	50
3.3.9	YOLOv8 – A modern approach to Object Detection	54
4	Methodology	57
4.1	Shafting Arrangement Modeling Process	57
4.2	YOLOv8 and Transfer Learning	60
4.3	Object Detection Validation and Rules	61

4.4	Process Diagram for Shaft Modeling	63
4.5	Optical Character Recognition Techniques	65
5	Dataset Generation	66
5.1	Dataset Generation	66
5.2	Dataset Preprocessing Step	69
5.3	Dataset Inspection & Quality Evaluation	70
6	Training & Validation.....	76
6.1	Choosing YOLOv8 Model Variation	77
6.2	System Specifications for Training	77
6.3	YOLOv8 CLI and Python SDK	78
6.4	Roboflow API and Importing Training Dataset	78
6.5	YOLOv8 Training Hyperparameters	79
6.6	YOLOv8 Training Evaluation Metrics	81
6.6.1	Key Entities Detection Model Evaluation	81
6.6.2	Shafts Detection Model Evaluation	83
6.6.3	Dimensions Detection Model Evaluation	86
6.7	YOLOv8 Inference with Shaft Arrangement Samples	89
6.7.1	Key Entities Detection Model Inference	90
6.7.2	Shafts Detection Model Inference	92
7	Shaft Modeling Results	94
7.1	Details of the Application	94
7.2	Output Results for the Shaft Modeling	94
7.3	Additional Results worth noting	102
8	Conclusion & Future Work	103
8.1	Synopsis and Conclusion	103
8.2	Improvements using the developed software	104
8.3	Future work	105
9	Bibliography	106

List of Figures

Figure 3.1 Creation of a Digital Image	7
Figure 3.2 Grayscale intensity range	8
Figure 3.3 RGB Intensity Cube	8
Figure 3.4 Example of Shaft Arrangement Plan	9
Figure 3.5 Image Filtering Techniques	10
Figure 3.6 Color to Grayscale Conversion	12
Figure 3.7 Gaussian Distribution with $\mu=0$	12
Figure 3.8 Noise Generation using the Gaussian Model for $\sigma=0.5$	13
Figure 3.9 Sobel-Feldman Filter result	15
Figure 3.10 The Machine Learning structure	17
Figure 3.11 Example of Machine Learning	18
Figure 3.12 Linear Regression with errors	18
Figure 3.13 Implementation of Gradient Descent	19
Figure 3.14 Polynomial Example regression	21
Figure 3.15 Sigmoid Function for Classification Problems	21
Figure 3.16 Decision Boundary of Sigmoid Function	22
Figure 3.17 Classifiers in Multiclass Classification	24
Figure 3.18 Underfitting vs Overfitting Boundaries	24
Figure 3.19 A biological neuron	25
Figure 3.20 Perceptron Neuron	26
Figure 3.21 Step Activation Function	27
Figure 3.22 Sigmoid Activation Function	27
Figure 3.23 Tanh(x) with Sigmoid function comparison	28
Figure 3.24 ReLU Activation Function	29
Figure 3.25 A simple Neural Network	29
Figure 3.26 Deep Neural Network Architecture	30
Figure 3.27 Neural Network with 1 Hidden Layer	31
Figure 3.28 Precision - Recall Curve	35
Figure 3.29 F1 Score - Confidence Curve	36
Figure 3.30 Confusion Matrix Example	37
Figure 3.31 AUC-ROC Curve Example	37
Figure 3.32 Structure of a CNN	38
Figure 3.33 Image Convolution of Shaft Bearing Example	39

Figure 3.34 Step 1 of Convolution Example	40
Figure 3.35 Step 2 of Convolution Example	40
Figure 3.36 Convolution on a 3-channel image	41
Figure 3.37A Fully detailed Convolution Layer	41
Figure 3.38 Convolution Layer representation	42
Figure 3.39 Convolutional Neural Network example	42
Figure 3.40 Max Pooling and Avg Pooling	43
Figure 3.41 Pooling on Volume	43
Figure 3.42 The LeNet Architecture	44
Figure 3.43 AlexNet Architecture	44
Figure 3.44 Classification and Localization on Object Detection	45
Figure 3.45 Definition of IoU	46
Figure 3.46 One-Stage Methods compared to Two-Stage Methods	47
Figure 3.47 R-CNN Steps and Architecture	48
Figure 3.48 Faster R-CNN Steps and Architecture	48

1 Introduction

1.1 Introduction to Artificial Intelligence

Artificial Intelligence (AI) is the replication of human intelligence in machines, designed to mimic human thinking and behavior. AI systems can analyze datasets, detect patterns, and make informed decisions based on available information. Alan Turing's paper, "Computing Machinery and Intelligence," paved the way for theoretical computing and AI, as he explored the possibility of computers exhibiting human-like intelligent behavior. AI has the potential to revolutionize the work of marine engineers by enabling them to make data-driven decisions, improve accuracy and efficiency, and foster innovation and creativity. By automating repetitive tasks, AI software can reduce the risk of human error, thereby enhancing the safety of maritime operations and marine engineering projects.

1.2 The role and ability of Marine Engineers

Marine engineers have a crucial role in the design and maintenance of merchant vessels. They are responsible ensuring that the marine systems are well maintained and functional for the vessels' lifetime. One of the key skills that sets marine engineers apart is their ability to read and understand mechanical design drawings. These drawings provide detailed information about the layout and functionality of several marine systems such as E/R arrangements: (Main Engine, Pumps, Filtering Systems, BWTS). Marine engineers use these drawings to understand the inner workings of the arrangements depicted in them and be able to use the data from them to operate, troubleshoot and maintain the ship's systems. Such an ability to read and interpret complex drawings is a critical aspect of their field of work and ultimately it is essential to ensure the safety of the maritime operations.

Our present thesis will evaluate the feasibility of designing an AI software which is ultimately capable of thinking like an engineer and understanding mechanical design drawings, identifying crucial objects and correlating data from the objects detected to extract digital twin of the drawing.

1.3 Our scope of work

Our focus will be on a designing an AI for modeling a particular drawing: The Shafting Arrangement Plan. Specifically, the Shafting Arrangement Plan is a critical drawing of the design and construction of a vessel. It is a detailed illustration that shows the placement, the positions and dimensions of the vessel's shafts, bearings and the components which are part of the vessel's propulsion system. The Shafting Arrangement Plan provides crucial information to the marine engineer by helping them understand the principles of the vessel's propulsion. It includes the necessary data which ensure that the various components of the propulsion system are aligned and spaced properly in order for the vessel to operate efficiently with minimal vibration. The AI should be capable of understanding the structure and the positions of the various mechanical components of the shafting arrangement plan and be able to extract a digital twin of the shafting arrangement, with inputs able to be imported to a Shaft Alignment software.

Using an accurate object detection model is really important for the development of the AI because even the small errors in the shaft arrangement plan can lead to significant problems in the shaft alignment process such as increased wear, tear on components and even complete system failure. Therefore, it is necessary to apply validation criteria based on the engineers' knowledge and critical thinking which will be able to handle the digitalization of the shaft with care and attention to detail.

In the following chapters, the overview of the literature is presented. Basic theoretical concepts regarding machine learning, computer vision, object detection will be discussed on great detail. The basic foundation of the shaft arrangement should be introduced along with the key components of the shaft arrangement which will ultimately be our objects of detection.

2 Literature Review

Studies regarding Object Detection and Data Extraction from Mechanical Drawings reveal the great significance and interest of researchers in the fields of computer vision and engineering. Progress has been made throughout the years as both the hardware of the machines and the detection models improve throughout the years. In the field of object detection at mechanical drawings, most of the applications are based on broader object detection studies using SVMs and Edge Detection as well as their improved Deep Learning CNN algorithms such as SSDs, (Single Shot Detector), R-CNNs (Region-based Convolutional Neural Network) and YOLO (You Only Look Once) models.

2.1 Object Detection using SVMs & Edge Detection

In one of the initial studies in this field, P. Dosch, K. Tombre, C. Ah-Soon, and G. Massini (2000) employed a combination of edge detection and support vector machine (SVM) algorithms for detecting objects in architectural drawings. SVM is a machine learning algorithm utilized for classification and regression analysis. In object detection, SVMs can recognize and locate objects in drawings and images by identifying the hyperplane that separates the objects from the background. The algorithm identifies the most suitable hyperplane that maximizes the margin between objects and the background, offering a reliable technique for identifying objects in noisy and cluttered drawing images. The aforementioned researchers utilized their developed SVM to detect objects in architectural drawings by training it on a set of labeled images with manual annotations. The algorithm learned to recognize objects and applied its knowledge to detect the objects in new, unseen images. To enhance the accuracy of detection, their SVM was combined with an Edge Detection algorithm that could identify object and background boundaries. This provided additional information to the SVM algorithm, enhancing its ability to make accurate digital modeling decisions. SVMs are useful for object detection in mechanical drawings due to their robust ability to differentiate objects from the background, and their versatility in handling various types of image features, such as edges and textures.

2.2 Object Detection using Convolutional Neural Networks

2.2.1 Implementation of R-CNN Networks

Amongst modern researchers a deep-learning based approach is more spread, especially using Convolutional Neural Networks. Girshick R. Donahue J. Darrel T. and Malik J. (2013) developed the Regional Convolutional Neural Network (R-CNN) which was the first model to implement convolutional neural networks in object detection. In the R-CNN method, the objects' region proposals are generated via a traditional object detection method, such as selective search and then they are passed through a CNN to extract features from the proposed regions. These features are then used to make object detection decisions using a classifier such as an SVM. The combination of CNNs and SVMs enabled a much more computational-efficient way to identify objects inside images than using an SVM alone. R-CNN was widely used since then for computer vision applications, and it was improved by its successor Faster-RCNN which was used in object detection at mechanical drawings such as Hu, H., Zhang, C. & Liang, Y. (2021) who used a Faster-RCNN to detect roughness regions on drawings. R-CNN models are continued to be used as solution and they are used for their efficiency and fast response in detection results.

2.2.2 Implementation of Single Shot Multibox Detectors (SSDs)

Wei Liu, Dragomir Anguelov, et al. (2016) developed an SSD (Single Shot Multibox Detector) which used Convolutional Neural Networks which could be trained end-to-end to make object detection decisions directly from the input image. Their proposed method showed a promising method in the future of object detection by outperforming traditional computer vision models such as SVMs. The SSDs ability to make object detection decisions directly from the feature maps allows SSD algorithms to be faster, more precise and efficient than SVMs, which typically require multiple steps such as region proposal generation and feature extraction. Gimenez, L. Robert, S. Suard, F. Zreik, K. (2016) incorporated SSDs in an attempt to reconstruct 3D Building Models from scanned 2D floor plans. Using SSD algorithms over SVMs enabled the researchers to handle wide range object sizes and aspect ratios. SVM algorithms typically require fixed-sized feature vectors as inputs which can be a limitation in object detection where objects can have a wide range of shapes and sizes. On the contrary, SSD algorithms are designed to handle a variety of object sizes using anchor boxes which are predefined bounding boxes which are used to generate region proposals of for object

detection. According to Z. Li and F. Zhou, the implementation of Convolutional Neural Networks was considered a breakthrough in object detection enabling a robust and powerful method to provide a robust and fast object detection whilst being able to handle objects with different sizes and aspect ratios.

2.2.3 Implementation of YOLO algorithms in Object Detection

Following the previous Object Detection algorithms, Ali Farhadi (2016) developed the YOLO (You Only Look Once) model which found much application in object detection solutions. YOLO uses a single CNN to perform object detection without the need for regional proposals or multiple stages of processing. This results in a much faster object detection than R-CNN and SSD making it suitable for real-time applications. Additionally, YOLO is able to make object detections based on a full image rather than individual regions, thus it allows to make predictions for objects which are not fully contained within one region, a limitation which appeared on R-CNN. This enabled researcher such as Nurminen, J.K., et al. to implement a YOLO model for Object Detections in Design Diagrams even when there is noise or part of the object is missing from the drawing. Since 2016 there have been multiple versions of YOLO algorithms which are improved in both accuracy and prediction speed. The YOLO models have been evolved vastly than their predecessors making them the most frequent choice of object detection amongst researchers.

2.2.4 Regarding YOLOv8

YOLOv8 (Version 8) was released at January 10th, 2023, and it is the latest version of YOLO boasting improvements in its accuracy and robustness. Thus, little to none research papers have been done yet, considering this particular thesis one of the first examining a YOLO v8 Object Detection model. According to J. Solawetz (2023), representing Ultralytics, the leading company in developing the YOLOv8 there are several reasons why to implement it in the new object detection models:

- 1) YOLOv8 has the highest accuracy amongst its predecessors, achieving a mAp (50-95) of 54.5% on the COCO dataset.
- 2) YOLOv8 has a well-structured CLI and Python commands, suitable for easier training and application deployment.
- 3) Last but not least, YOLOv8 comes with bug improvements and a well-maintained community with experts on the field of computer vision.

3

Theoretical Background

As explained by Wang and Lu (2015), Artificial Intelligence (AI) involves developing machines and algorithms that can perform tasks requiring human-like intelligence, including recognizing patterns, learning from data, and making decisions based on their training. The field of Object Detection has seen a significant impact from AI in recent years, which involves identifying instances of specific objects in images or videos.

With advances in computational power, AI algorithms can now perform this task with great accuracy, even with complex and cluttered inputs. Convolutional Neural Networks (CNNs) have emerged as the leading machine learning technique for object detection. These networks use large pre-labeled image datasets and employ multiple processing layers to learn increasingly complex features and object representations. However, unlike humans, machines lack the ability to understand context and knowledge expertise and can only recognize objects based on learned patterns.

To enhance AI's cognitive abilities for object detection in mechanical design drawings, it is recommended to develop a set of validation rules that can emulate the decision-making process of an intelligent marine engineer.

3.1 Introduction to Computer Vision

Digital Imaging and Conversion

The field of computer science that concentrates on enabling machines to interpret, understand and analyze information is known as Computer Vision. It encompasses the development of algorithms and models capable of analyzing images in the same way humans do, including recognizing objects and detecting patterns.

In computer vision, the image itself is one of the key mathematical representations. It is defined as a 2-dimensional array of pixel values and mathematically represented as a function. function $g: R \rightarrow R^2$:

$$z = g(x, y) \quad , \quad x, y, z \in R$$

where x, y are the spatial coordinates of each pixel and the value of the function $z = g(x, y)$ represents the intensity of the pixel.

Although machines are able to only comprehend digital images which are discrete representations of analog images. This can be achieved in 2 processes:

Digital Sampling: Involves the dividing of the image into a 2D grid of pixels and measuring the intensity of light/color at each cell. Mathematically it is represented by a mapping function which maps a continuous 2D image to a discrete grid:

$$f: R^2 \rightarrow Z^2$$

Quantization: The intensity values are quantized; they are rounded to the nearest integer as follows:

$$f[i, j] = \text{round}\left(\frac{g(x, y)}{q}\right) \cdot q$$

Where $f \in Z^2$ is the quantized intensity value at the coordinate (x, y) , $g(x, y)$ is the continuous intensity value at the same coordinate, q is the quantization step and $\text{round}(X)$ is the rounding function. The quantization step q determines the number of levels of intensity that are used to represent the image. The smaller the q , the finer the image representation is, although it comes with a trade-off of a larger amount of image data to be processed.

This results in a matrix of discrete values which represent the image as seen in the image below:

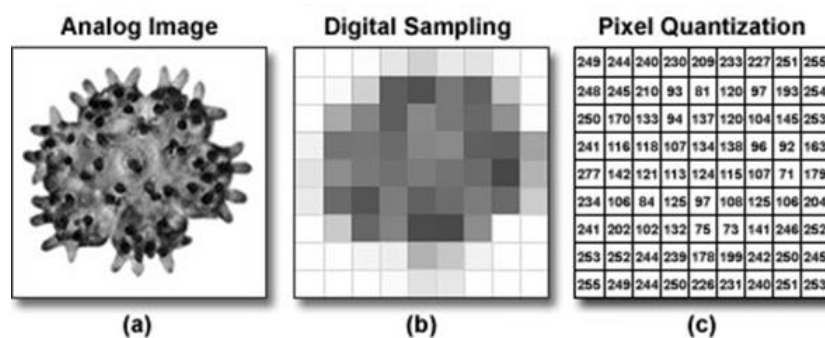


Figure 3.1 Creation of a Digital Image

Grayscale and Color Images

We have 2 different types of digital image models: The Grayscale and the Colored Image:

Grayscale Image: It is an image consisting only of shades of gray with no color (monochromatic image), just like the one defined above. In a grayscale image each pixel is represented by a single intensity value ranging from 0 (black) to 255 (white). The intensity value of a pixel

determines its shade of gray with the lower values being darker and the higher values being lighter.

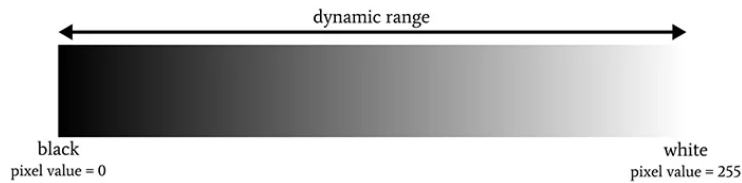


Figure 3.2 Grayscale intensity range

Colored Image: On the contrary, a colored image is an image which includes multiple color channels, typically Red, Green and Blue (RGB). In a colored image, each pixel is represented by three values (R, G, B) which represent the intensity of each color channel. The additive combination of these three intensity values results in a specific color for each pixel. The figure below represents a comparison between the grayscale intensity range and the colored range which is illustrated as a cube, thus describing the digital colored pixel as a 3D vector:

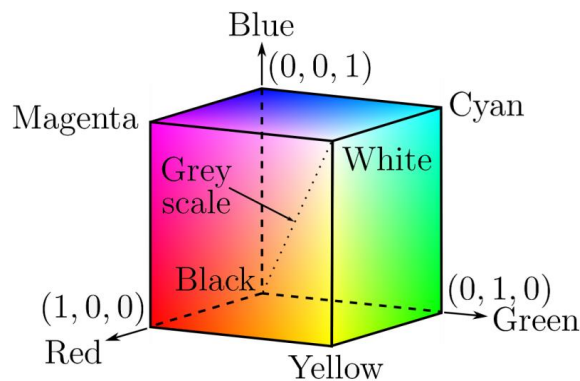


Figure 3.3 RGB Intensity Cube

Comparison between Grayscale and Colored Images

Colored images provide a more natural representation of the world with color being an important aspect of our perception. Although, comparing grayscale images to colored images, the implementation of grayscale images is much preferred, when possible, in computer vision models for the following reasons:

- A) *Simplicity*: Grayscale images are simpler to process and analyze than colored images. They contain only one channel of information representing the intensity of each pixel, making it easier to detect edges, shapes and textures.

- B) *Reduced computational complexity:* Grayscale images require fewer computations than grayscale images, as there is less information to process, thus making grayscale images faster and more efficient to work with.
- C) *Reduced noise:* Color images tend to have more noise and variations in intensity than grayscale images, which can pose difficulties in identifying accurately features and detecting objects in them.
- D) *Increased Robustness:* Grayscale images are often more robust to changes in lighting and exposure conditions than colored images, where color information can be severely affected.
- E) *Texture & Shape oriented:* A plethora of computer vision and image processing applications focus on detecting the shape and texture of objects rather than their color. Grayscale images provide a much straight-forward representation of shape and texture making them optimal to identify such features.

In our case scenario, Naval Architectural plans do not have a color-coded information on them as the lines are black colored and the drawing background is white. Additionally, the texture of the lines (width and linetype) and their shape are the most important features to identify on our target geometrical entities of a mechanical drawing. Due to the factors above, it is only reasonable to proceed our study by using grayscale digital input images.

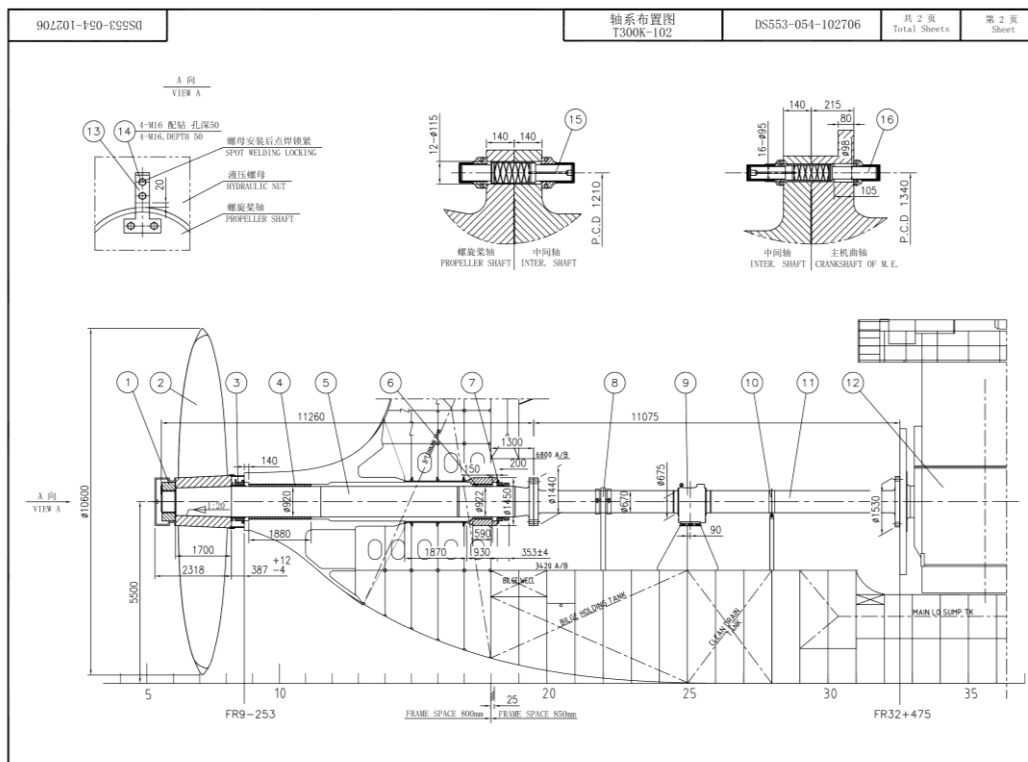


Figure 3.4 Example of Shaft Arrangement Plan

3.2 Image Filtering Techniques

The necessity to import grayscale images to improve our object detection modeling in accurately extracting features from mechanical design drawings, arises importance of introducing a color-to-grayscale image filter.

Image filtering is a pre-processing step of great importance in computer vision which involves transforming a digital image into a simplified or enhanced version of itself. Image Filtering can be used to remove noise, removing unwanted features or highlighting certain features in order to transform the image in an input suited for further processing an analysis.

There are different categories of image filtering techniques as shown in the graph below:

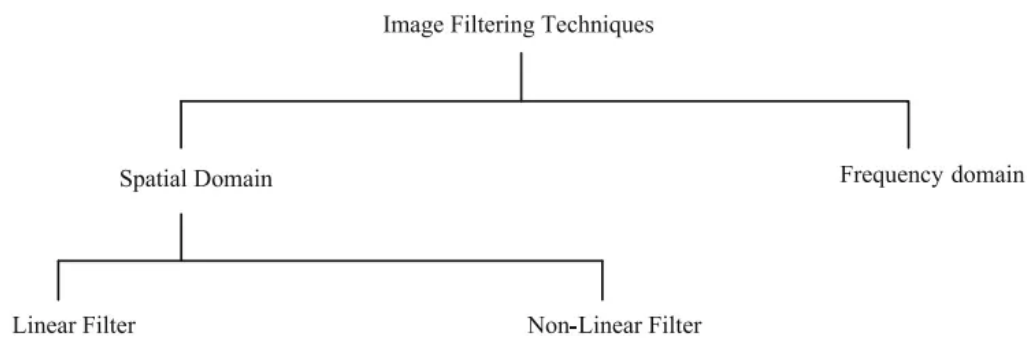


Figure 3.5 Image Filtering Techniques

Spatial Filtering: Refers to filters which operate on the pixel values of the image in their spatial locations. They operate using convolution between the filter and the image:

Linear Filters perform linear transformation on the pixel intensity values with a set of weights to produce a new image. This can be expressed using the convolution operator. Assuming $f(i, j) \in Z^2$ is an image and the filter or kernel is h which includes the weights $h(k, l)$ then, their convolutional output image is denoted as:

$$g = f \otimes h$$

$$g(i, j) = \sum_{k, l}^N f(i - k, j - l) \cdot h(k, l) = \sum_{k, l}^N f(k, l) \cdot h(i - k, j - l)$$

Depending on the weights (filter coefficients) of the kernel h we may have different contributions of each input pixel in f , generating a variety of linear filters designed to operate various image processing tasks such as smoothing, sharpening and edge detection. Such linear filter examples include:

- A) Gaussian filter: Which smoothens the image by weighting the neighboring pixels based on a Gaussian distribution.
- B) Sliding Average filter (Box filter): Replaces each pixel with the mean value of its neighboring pixels, which helps to reduce noise.
- C) Sobel filter: Detects edges in an image by computing the gradient of the image's pixel intensity.

Non-Linear Filters perform non-linear operations on the image intensities. considering the kernel as a statistical estimator. Their objective is to estimate the intensity values of the pixel when the pixels are in presence of noise. Commonly used non-linear filters include:

- A) Median filter: It estimates the value of a pixel in an image by selecting the median value as the pixel intensity value from a set of neighboring pixel values.
- B) Bilateral filter: It considers the local intensity of a pixel and its spatial neighbors in order to calculate their weighted average in a manner to preserve edge information and reduce the image's blurring.

Non-Linear filters are ideal to preserve image details but at the cost of higher computational time due to their operations being more complex. As a general thumb of rule, fast linear filters are effective for simple image processing tasks (such as smoothing and sharpening), however when preserving image details is critical, a non-linear approach is much more applicable.

Grayscale Operation

The Grayscale Operation refers to a linear filter that transforms a color image into a grayscale one. This operation involves reducing the richness of the color image from its RGB channels, which are Red, Green, and Blue, to a single intensity channel or brightness. This grayscale transformation is achieved by calculating a weighted average of the red, green, and blue color channels to obtain a single grayscale intensity value for each pixel:

$$f(i, j) = w_1 \cdot R(i, j) + w_2 \cdot G(i, j) + w_3 \cdot B(i, j)$$

The most commonly used weighting combination is the luminance-based weighting, where the red, green and blue channels are weighted as:

$$w_1 = 0.2989 \quad w_2 = 0.5870 \quad w_3 = 0.1140$$

After the weighted average is computed for each pixel, the resulting single intensity value represents the grayscale version of that pixel. The final grayscale image is a 2D matrix of intensity values as it can be seen in the figure below:

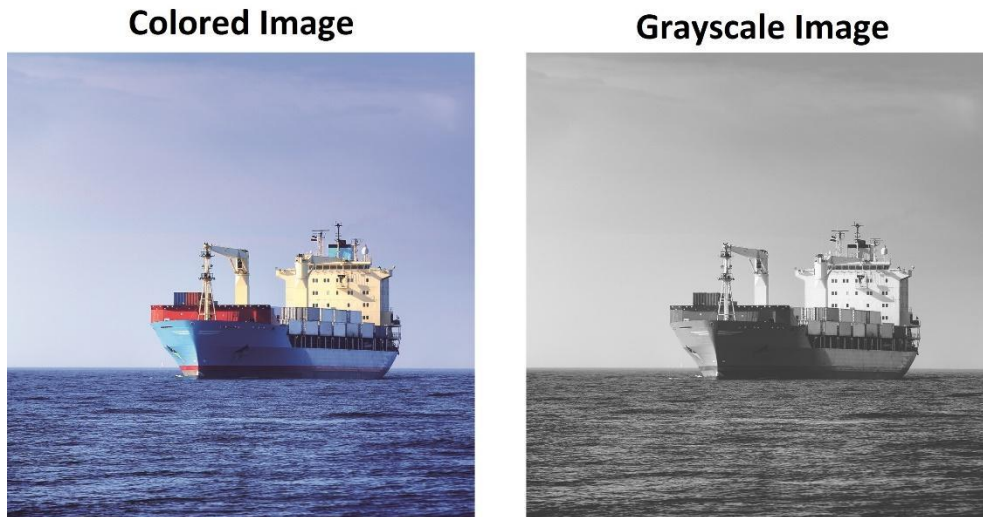


Figure 3.6 Color to Grayscale Conversion

Preprocessing using Image Filters

Prior training a computer vision model it is necessary to preprocess the data in a manner which improves the model’s detection accuracy. Regarding scanned images, it is suggested to generate a dataset which will include noise and minimal rotations (± 1 deg.)

Noise Generation using Gaussian Distribution

Noise generation using the Gaussian distribution is a commonly used technique to simulate real world image acquisition scenarios with poor light conditions and sensor noise present. A Gaussian Distribution is defined as:

$$G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \quad \mu = 0 \rightarrow \quad N_z(0, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{z^2}{2\sigma^2}}$$

where μ is mean and σ the standard deviation. During noise generation, the $\mu = 0$ ensure that the noise will have a net effect of zero on the overall pixel intensities of the image.

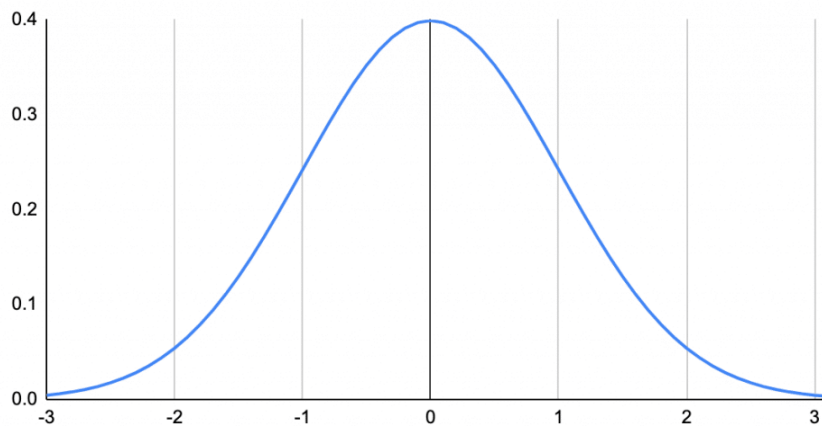


Figure 3.7 Gaussian Distribution with $\mu=0$

Using the Gaussian distribution, we are able to sample random values with random probabilities which will be added on the pixel intensities, a linear operation such as:

$$I_{noisy} = I + N(0, \sigma^2)$$

The higher the standard deviation is $\sigma \uparrow$, the result image will have more intense noise and vice versa. For $\sigma = 0.5$ the result is shown in the figure below:



Figure 3.8 Noise Generation using the Gaussian Model for $\sigma=0.5$

Image Rotation

Image rotation by a certain angle around its center despite not being a filter, it's a necessary transformation for increasing the accuracy of our model. The rotation is performed by transforming the 2D coordinates of each pixel by multiplying the matrix with the rotation transformation matrix. For an image of a given height h and width w (in px) the transformation of the image around its center $(w/2, h/2)$ for any angle θ is:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & \frac{w}{2}(1 - \cos\theta) + \frac{h}{2}\sin\theta \\ \sin\theta & \cos\theta & -\frac{w}{2}\sin\theta + \frac{h}{2}(1 - \cos\theta) \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The first 2 rows perform the rotation and translation at the same time to keep the image in the same initial position, while the last row is a homogenous transformation to keep the third dimension unchanged.

Preprocessing Methods

Regarding To facilitate the thesis case, it is imperative to utilize an OCR (Optical Character Recognition) algorithm to read the dimension numbers from the Shaft Arrangement Plan. Various types of image processing filters can be employed to improve the accuracy of the character recognition process in OCR, such as Edge Detection Filters (e.g. Sobel or Canny filters) and Noise Reduction Filters (e.g. Median or Gaussian filters).

As per C. Patel et al (2012), these filters can augment the visibility of the numbers, making them more distinguishable for the OCR model to precisely recognize them.

In addition, as per A. Singh (2012), some systems may utilize morphological filters or binarization techniques, which depending on the image scenario, could escalate the recognition accuracy by up to 30%.

Sobel Filter Edge Filter

As it is previously mentioned, the Sobel filter operated by convolving the image with 2 3x3 kernels, one for detecting the horizontal edges and another for the vertical edges. The Sobel-Feldman kernels operate by calculating the gradient of the pixel intensity values. Since we are interested in detecting areas where the intensity changes drastically, as these are likely to be the boundaries between objects, it is reasonable to use the gradient as a measure of the rate of change in pixel intensity values. Thus, the Sobel filter calculates the gradient in both horizontal and vertical direction and the gradient magnitude is considered a measure of the edge strength. The Sobel kernels are:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

And the gradient magnitude as well as the angle of orientation are

$$\nabla f = |G| = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$
$$\theta = \left(\frac{G_y}{G_x} \right)$$

As W. Gao and X. Zhang (2010) suggest, the Sobel filter is computationally efficient, produces clear distinct edges in the images and it appears robust on noisy images. Although the Sobel filter is sensitive to intensity changes which can result in false edges being detected, it may miss small or thin edges and it is not ideal for curved edges. With a plethora of edge detection filters such as Canny, LoG and Hough filters, depending on the task the best filter is to be chosen.

An example of performing the Sobel Edge Filter is shown in the figure below:

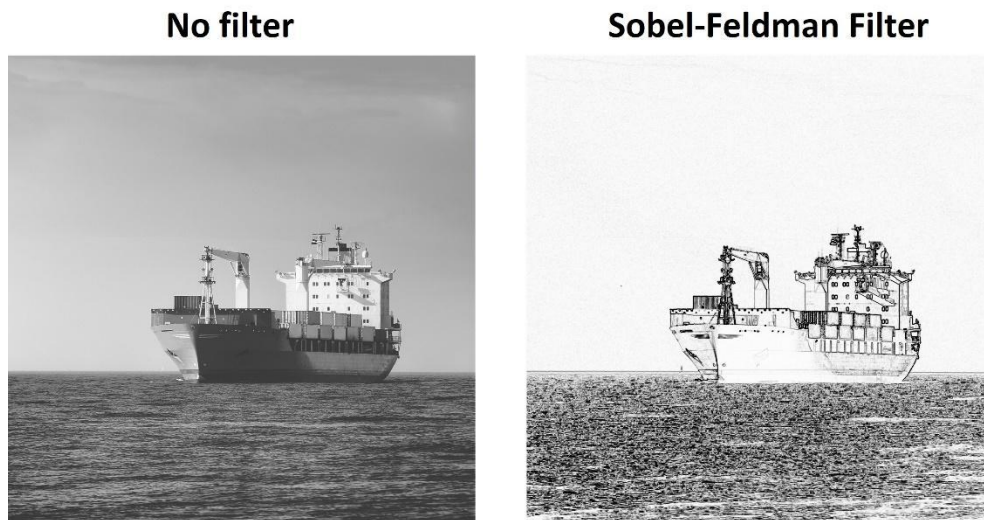


Figure 3.9 Sobel-Feldman Filter result

Open CV as a Computer Vision Library

The OpenCV software library is an open-source computer vision and machine learning library that offers pre-trained algorithms and functions for computer vision applications, such as object detection, video analysis, and image processing. The library is written in C++ and can be used in several programming languages, including Python, Java, and C#.

According to G. Bradsky, the founder of OpenCV, it is useful in computer vision projects for image processing because it provides many algorithms for image processing, such as filtering, morphological operations, edge detection, and thresholding. These algorithms can be utilized for tasks like image segmentation, image de-noising, and object recognition. OpenCV also has pre-trained deep learning models for object detection that can be fine-tuned for specific use cases.

Furthermore, Python's native PIL (Python Imaging Library) can also be used to open, manipulate, and save images and has essential modules for image processing.

For our thesis object detection model, we can employ OpenCV and PIL libraries to access the filters for image processing by integrating them as functions in our thesis' shaft arrangement object detection model.

3.3 Machine Learning and Deep Neural Networks

3.3.1 Machine Learning

Machine learning, as described by Arthur Samuel, refers to the ability of computers to learn without being explicitly programmed. However, a more contemporary definition was proposed by Tom Mitchell, which states that a computer program learns from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E . For example, in our object detection model for mechanical design entities in mechanical drawing images, E refers to the experience gained from multiple iterations of learning what objects need to be detected, T refers to the task of detecting all available mechanical entities, and P refers to the accuracy of predictions based on the engineer's knowledge. Machine learning problems can generally be classified as either supervised or unsupervised learning. In supervised learning, the computer is given a dataset and already knows what the correct output should be. Supervised learning problems can be further classified as regression or classification problems. Regression problems involve predicting results within a continuous output, while classification problems involve predicting results in a discrete output, meaning that input variables are mapped into discrete categories.

A) Supervised Learning

The field of Supervised Learning involves using a dataset to learn a relationship between input and output data. This approach includes two types of problems, namely "Regression" and "Classification." A regression problem involves predicting continuous output values based on input variables, while a classification problem involves predicting discrete output values by assigning inputs to specific categories.

In our current thesis the problem of detecting certain objects inside an image input is a broadened classification problem, since we have the digital image data as an input and as output we have multiple discrete classes whether the objects we detect are for example "bearings", "flanges" or "propellers".

B) Unsupervised Learning

Unsupervised Learning enables us to approach problems with minimal knowledge of what our result output would be although we can derive structure from data where we don't necessarily know the effect of the variables. We derive such structure by clustering the data based on relationships among the variables in the data with a lack of feedback on the prediction results. A common unsupervised learning problem in business and marketing, is

given a dataset of customer data including the age, income, and location of customers an AI can apply clustering algorithms to group customers into similar segments such as “young with low income” or “elderly with high income”, which can be deemed useful in targeted marketing and predictive advertisement applications.

Machine Learning Model Representation

To establish our notation for future reference we use $x^{(i)}$ to denote the “input” variables (features) and $y^{(i)}$ to denote the “output” (or target) variable, which the AI is attempting to predict. A pair $(x^{(i)}, y^{(i)})$ is called a training example and the dataset we will be using to learn is a list of m training examples $(x^{(i)}, y^{(i)})$: $i = 1, 2, \dots, m$ is defined as a training set. Note that the superscript (i) in notation is an index into the training set and not an exponentiation. Additionally, we denote as X the space of input values and Y the space of output values. In an object classification problem, the $X \in f(x\Delta, y\Delta) \exists Z$ the image data in a matrix and $Y \in Z$ which are the classes being detected.

To describe the supervised learning problem slightly more formally, the goal is, given a training set, to learn a function $h : X \rightarrow Y$ so that $h(x)$ is an accurate predictor for the corresponding value of y , with $h(x)$ referred as a hypothesis function:

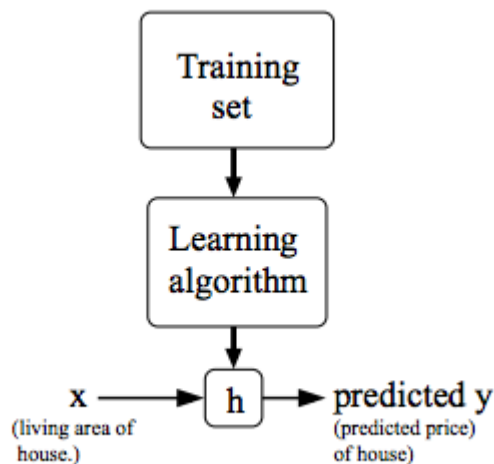


Figure 3.10 The Machine Learning structure

In a regression problem, the $y \in R$ since it’s a continuous prediction but in a classification problem, the $y \in Z$ since we predict discrete classes $Y = \{0, 1, 2, \dots\}$. A brief example would be illustrative in our case explaining how to train a simple machine learning algorithm:

Assume we have a set of features x and outputs y which comprise a training set: $(x^{(i)}, y^{(i)})$

Feature Variables x	Output Variables y
2104	460
1416	232
1534	315
852	178
... (m inputs) (m outputs)...

Figure 3.11 Example of Machine Learning

Our goal is to learn a function h which can accurately predict the y given the x . We assume the hypothesis function is linear:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

In order to achieve this goal, we are tasked to choose the parameters θ_0 and θ_1 in order for the line to fit the points:

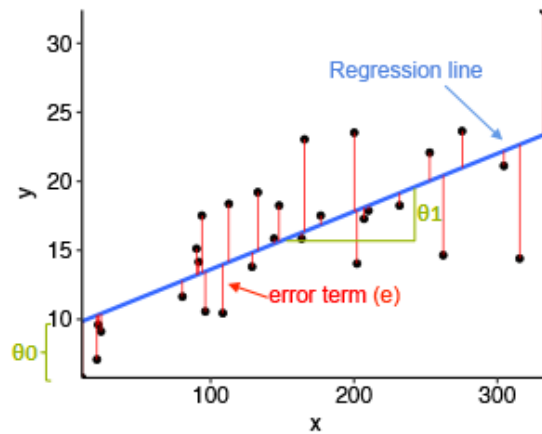


Figure 3.12 Linear Regression with errors

For point $x^{(i)}$ with output $y^{(i)}$ its projection on the line is $h_{\theta}(x^{(i)})$. Observing the problem above is only natural to conclude that a well fit of the line derives with parameters θ_0, θ_1 which minimize the average vertical distance between the points and the line itself:

$$\frac{1}{2m} \sum_{i=1}^m [h_{\theta}(x^{(i)}) - y^{(i)}]^2$$

The metric used above is defined as a loss function, more specifically the Mean Squared Error loss function (the $(1/2)$ halved mean is used for computational convenience):

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m [h_{\theta}(x^{(i)}) - y^{(i)}]^2 = \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

Therefore, our problem is rewritten as:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Which means that we seek the possible best line, such so as the MSE of the vertical distances of the scattered data points from the line will be the least, ideally passing through every point. A visualization between the parameters and the J error and the parameters will help perceive problem of the MSE minimization:

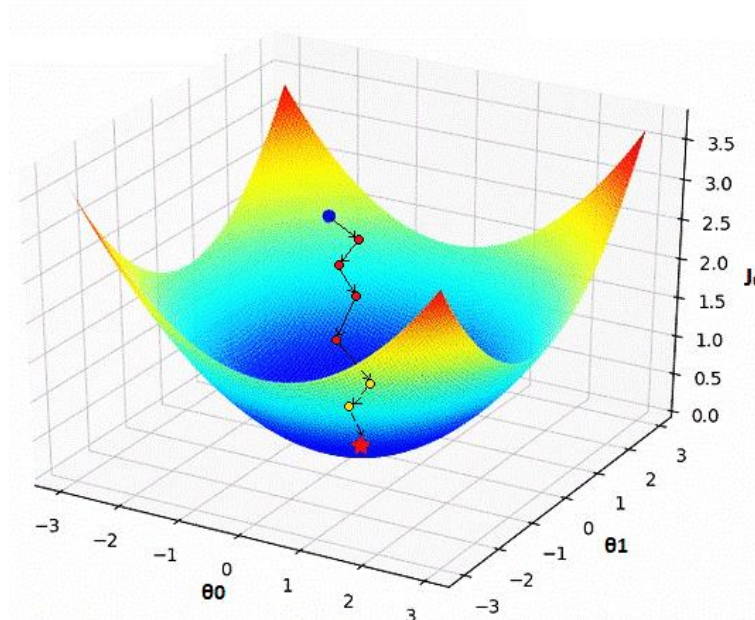


Figure 3.13 Implementation of Gradient Descent

As seen in the figure above we will successful if we consider a path of the parameters (θ_0, θ_1) which leads us to the very bottom of the graph where the value of the error is the minimum. A solution to this is by taking the derivative (the tangent) of the loss function as a direction to move forward with the steepest descent step by step. The size of each step is determined by a parameter a which is defined as the *learning rate*.

Thus, the Gradient Descent algorithm is defined as an optimizer to be repeated until convergence:

$$\theta_j := \theta_j - a \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \rightarrow \quad \theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^m [h_{\theta}(x^{(i)}) - y^{(i)}]^2$$

The learning rate a is an important parameter to our learning process which ensures that the gradient descent algorithm converges in a reasonable time. If a is too small the gradient descent is slower, but if a is large, the gradient descent may overshoot causing a possibility to diverge from the minimum MSE. It is important to try multiple values of learning rates prior to training any machine learning algorithm, although there are applications where the a can be variable depending on the convergence to the minimum value of the MSE.

Machine learning with Multiple Features

When multiple features are introduced, we may have multiple number of parameters at our hypothesis function. Introducing the new notation:

$x_j^{(i)}$ = Value of the feature j in the i -th training example (number)

$x^{(i)}$ = The features of the i -th training example (vector of features of the i -th example)

m = The number of training examples $i = \{1, 2, \dots, m\}$

n = The number of features $j = \{1, 2, \dots, n\}$

Thus, the hypothesis function accommodating those multiple features can be expressed as:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

$$h_{\theta}(x) = [\theta_0 \dots \theta_n][1 \dots x_n]^{-1} = \theta^T x$$

The Gradient Descent can be rewritten for $j = \{0, 1, \dots, n\}$:

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^m [h_{\theta}(x^{(i)}) - y^{(i)}] \cdot x_j^{(i)}$$

Two techniques that can potentially speed up the learning process are feature scaling and mean normalization. Feature scaling involves dividing the input values by the range (e.g. the max value minus the minimum value) of the input variable resulting in a new range of just 1 and mean normalization involves subtracting the average value for an input variable from the values, resulting in a new average of just zero. To implement both of these techniques we adjust the input values as seen below:

$$x_i := \frac{x_i - \mu_i}{s_i}$$

Where μ_i is the average of the values for the and $s_i = \max - \min$

There is further room to improve our features and form our hypothesis function by combining multiple features into one, such as creating new features $x_3 = x_1 x_2$ Such can create new hypothesis functions such as

Polynomial Hypothesis:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_3^3 \quad \text{with } x_2 = x_1^2, \quad x_3 = x_1^2$$

Square Root Hypothesis:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1} \quad \text{with } x_2 = \sqrt{x_1}$$

Choosing an optimal hypothesis function may be a result of several trial-and-error efforts, so it is recommended to try different hypothesis functions depending the input data and choosing the one which gives the least error.

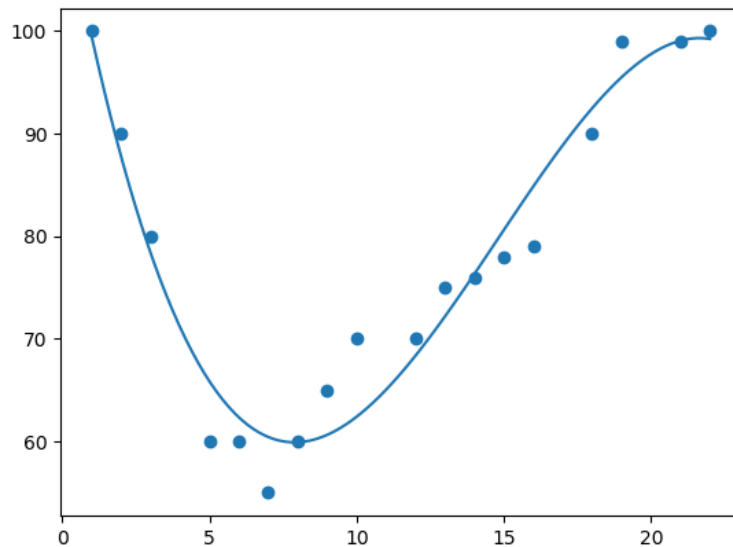


Figure 3.14 Polynomial Example regression

Classification Problems

Having a good overview of basic regression problems, we may proceed to classification problems. A novice attempt classification one simple method is to employ linear regression and map all predictions greater than 0.5 as 1 and all less 0.5 as 0, although such attempt would not always work.

The classification problem is similar to the regression, except that the values we want to predict take discrete values. Our initial focus will be on the binary classification problem at which the $y = 0$ or 1 . For instance, if we are attempting to build a classifier for shaft bearings from images, then $x^{(i)}$ may be the pixel intensities of the image representing the features of the image and y may be 1 if it is a shaft bearing (positive class) and 0 if it's not (negative class). The $y^{(i)} \in \{0,1\}$ is also defined as the label for the training example.

For a classification problem, where our hypothesis values cannot be greater than 1, it is reasonable to alter the form of the hypothesis function $h_{\theta}(x)$ to satisfy $0 \leq h_{\theta} \leq 1$.

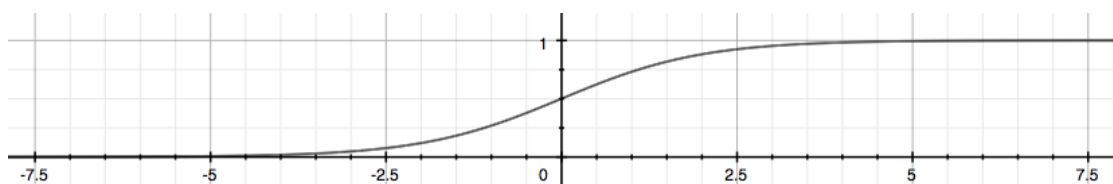


Figure 3.15 Sigmoid Function for Classification Problems

Our new form of hypothesis uses the “Sigmoid Function” (Logistic Function) such that:

$$h_{\theta}(x) = g(\theta^T x)$$

If $z = \theta^T x$:

$$g(z) = \frac{1}{1 + e^{-z}}$$

The function $g(z)$ maps any real number to the (0,1) interval, making it suitable for transforming an arbitrary valued function into a function suited for classification.

Now, $h_{\theta}(x)$ represents the *probability* that our output is 1. For example, $h_{\theta}(x) = 0.7$ represents a probability of 70% that our output is 1. Obviously, the probability that our prediction is 0 is the complement of the probability of being 1, in that case 30%:

$$h_{\theta}(x) = P(y = 1|x; \theta) = 1 - P(y = 0|x; \theta)$$

$$P(x; \theta) + P(y = 1|x; \theta) = 1$$

In order to output a discrete 0 or 1 classification we may translate the output of the hypothesis function as follows:

$h_{\theta}(x) \geq 0.5 \rightarrow y = 1$ If the probability to be 1 is above 50% - Its 1

$h_{\theta}(x) < 0.5 \rightarrow y = 0$ If the probability to be 1 is below 50% - Its 0

Since the sigmoid function $g(z) \geq 0.5$ when $z \geq 0$ for our input of $\theta^T x$ its safely perceived that when $\theta^T x \geq 0$ the $h_{\theta}(x) = g(\theta^T x) \geq 0.5$, hence:

when $\theta^T x \geq 0 \rightarrow y = 1$ and when $\theta^T x < 0 \rightarrow y = 0$ such may define the decision boundary which separates the area where $y=0$ and $y=1$. Objects above the decision boundary are classified as 1 and objects below the decision boundary are classified as 0.

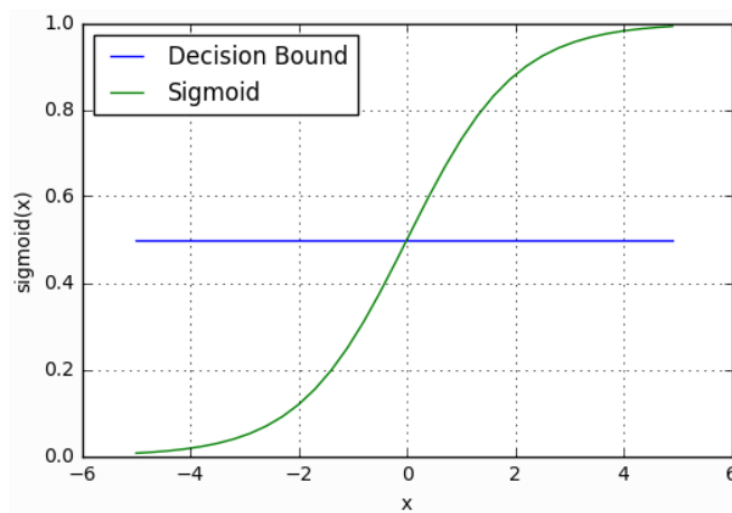


Figure 3.16 Decision Boundary of Sigmoid Function

Additionally, we can't continue to use the same MSE loss function for linear regression, because the sigmoid would not be a convex, hence for the logistic regression the loss function would be represented as:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

With:

$$\text{Cost}(h_{\theta}(x^{(i)}), y^{(i)}) = -y \log \log (h_{\theta}(x)) - (1 - y) \log \log (1 - h_{\theta}(x))$$

Thus, the expression above can be written as:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log \log (h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log \log (1 - h_{\theta}(x^{(i)})) \right]$$

A vectorized implementation would be:

$$h = g(X\theta)$$

The Gradient Descent is still considered a valid optimizer, although there are faster methods to optimize the θ parameters such as ADAM, BFGS and L-BFGS.

Multiclass Classification Problems

Having a solid foundation of binary classification, we proceed to multiclass classification where instead of $y = \{0,1\}$ we may have $y = \{0,1, \dots, n\}$. We will approach this multiclass by dividing the problem to $n+1$ binary classification problems, where in each one we predict the probability that y is one of our target classes against the rest of them. Thus the hypothesis:

$$y \in \{0,1, \dots, n\}$$

$$h_{\theta}^{(0)}(x) = P(y = 0|x; \theta) \quad \text{Probability of our class being 0 amongst the rest of them}$$

$$h_{\theta}^{(1)}(x) = P(y = 1|x; \theta) \quad \text{Probability of our class being 1 amongst the rest of them}$$

....

$$h_{\theta}^{(n)}(x) = P(y = n|x; \theta) \quad \text{Probability of our class being n amongst the rest of them}$$

This approach would be visualized by defining appropriate decision boundaries to separate the items of the target class with the items of the other classes. For example, given a dataset with multiple classes, such as mechanical entities from shaft arrangement drawings, we define decision boundaries as classifiers where at each case anything above it would be the target object and anything below it would not be (one-vs-all principle). This decision boundary will fit the classifier which will estimate the probability of y being a certain i -class:

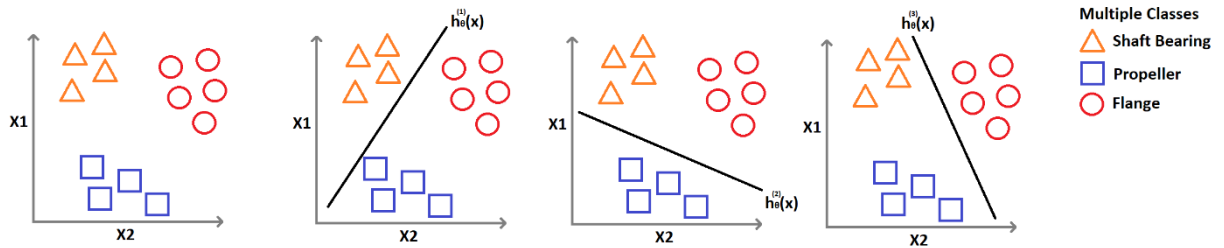


Figure 3.17 Classifiers in Multiclass Classification

Hence, we will be able to detect more than a single class using multiple binary classification problems.

Overfitting and Underfitting

Considering we want to optimize the classifiers we may attempt to use boundaries with fit our data with higher accuracy. Such could be added by adding extra features on the predictor line, however when adding more features there's a great chance to overfit the dataset:

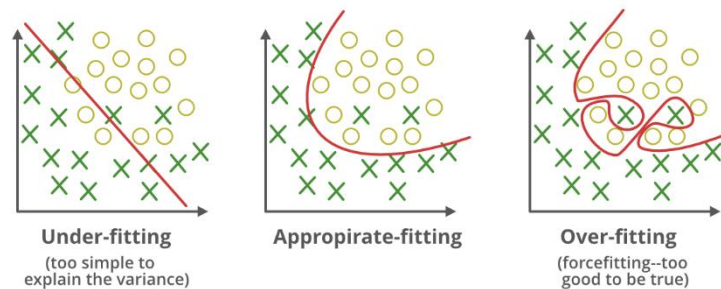


Figure 3.18 Underfitting vs Overfitting Boundaries

Underfitting: Our classifier is Underfitting when it is unable to capture the underlying trend on the data, such that it may perform well on training data but poorly on test data, resulting in poor prediction accuracy.

Overfitting: Our classifier is Overfitting when the classifier is generating localized adjustments to fit the dataset but fails to generalize the underlying trend on them failing to detect new data.

In order to prevent Overfitting and Underfitting we may employ techniques such as:

- A) Regularization, which adds a penalty term to the loss function to prevent overfitting and underfitting accordingly.
- B) Cross-Validation: Dividing the data into training – test – and validation sets where the model is being trained on the training set but validated for overfitting in a validation set.
- C) Early-Stopping: Which can stop the training optimizer when the performance of the validation starts to degrade.
- D) Adding noise: In order to make the model more robust to small variations of the input.

3.3.2 Overview on Neural Networks

The concept of neural networks has its roots in the 1940s and 1950, when researchers such as Warren McCulloch and Walter Pitts published a seminar paper titled *“A logical calculus of the ideas immanent to nervous activity”*, (1943) proposing the idea of using mathematical models to simulate the function of the human brain. This paper laid the foundation for the field of artificial intelligence and the development of neural networks. In the following decades researchers such as Hebb (1949) and Rosenblatt (1958) made important contributions on understanding the behavior and training of artificial neural networks. Hebb proposed the concept of *“Hebbian learning”* proposing that learning occurs when connections between neurons are strengthened through repeated use, a principle documented as *“neurons that fire together, wire together”*. This idea formed the basis for many early artificial neural network models and it is still an important concept in modern machine learning.

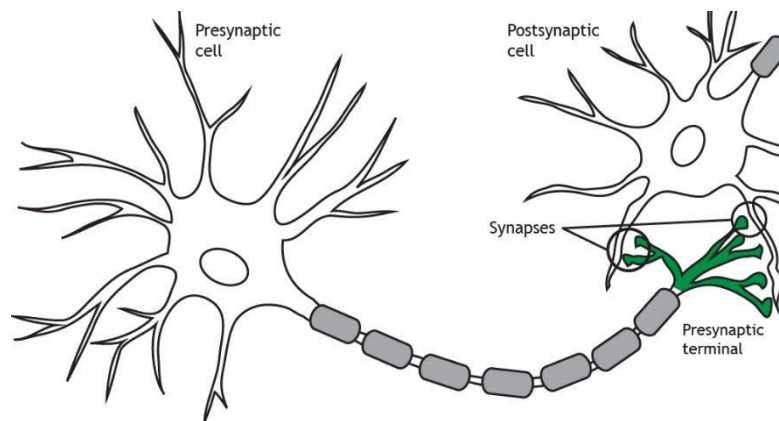


Figure 3.19 A biological neuron

The main concept of artificial neural networks is to model the behavior of biological by using simple mathematical functions to simulate the key aspects of the neural behavior. This includes the ability to receive inputs, process those inputs and then produce an output along with the ability to learn from that experience by adjusting the connections between neurons based on past inputs. From a biological perspective, biological neurons are the building blocks of the human brain and they are responsible for processing and transmitting information as electrical signals. They receive inputs from other neurons, integrate the signals and produce an output signal which is transmitted to other neurons.

This basic function inspired Rosenblatt (1958) by introducing the perceptron, the most fundamental building block of artificial neural networks. It consists of a single artificial neuron

that receives inputs, performs a weighted sum of the inputs and produces an output based on a threshold function as it can be seen by the figure below:

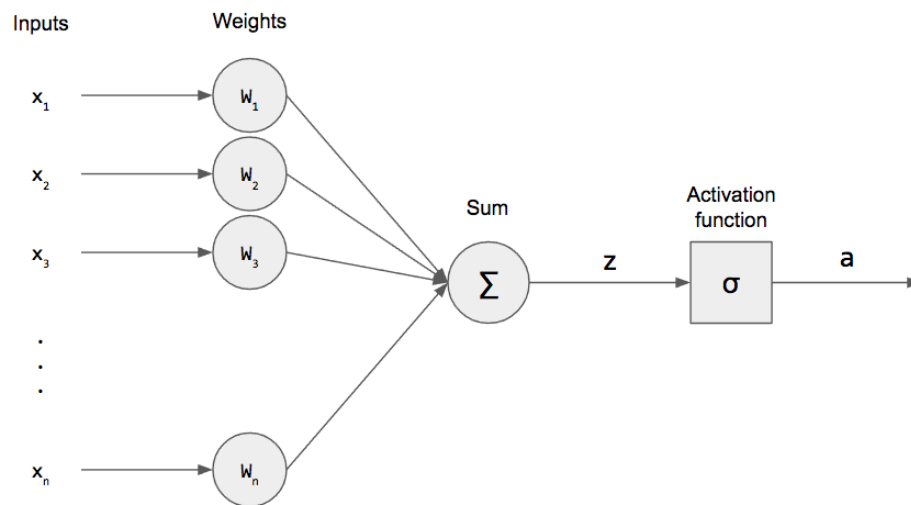


Figure 3.20 Perceptron Neuron

The perceptron consists of:

- Synapses which accept Inputs: $x_1, x_2, x_3 \dots x_n$ in numerical values. Those are considered the stimuli of the artificial neuron
- Weights, which are assigned to each input and determine the influence on the perceptron's output.
- The summing junction, which adds all the signals with their weight products, creating the sum denoted as z :

$$z = \sum_{i=1}^n w_i x_i$$

- The activation function, which acts as a threshold producing a binary output whose values above the desired threshold are set to 1 and below the threshold is set to 0

$$y = \sigma(z) = \sigma\left(\sum_{i=1}^n w_i x_i\right)$$

The threshold function and the perceptron's weights are being adjusted during its learning process, enabling the model above to learn from experience and improve its performance over time. Some researchers suggest adding a bias b_k in the perceptron model in order to shift the decision boundary of the perceptron, thus providing an additional degree of freedom to the network's output.

The perceptron output model could be rewritten as:

$$y_k = \sigma \left(\sum_{i=1}^n w_{ki} x_i + b_k \right)$$

As it is seen, by the perceptron model, the activation function poses a crucial role in thresholding the weighted sum (with the bias). It's binary decision (zero or one) is the factor which represent the decision made by the perceptron. The first preceptor used a step activation function which takes the value of 1 above a certain value and zero if it's not reaching the required threshold:

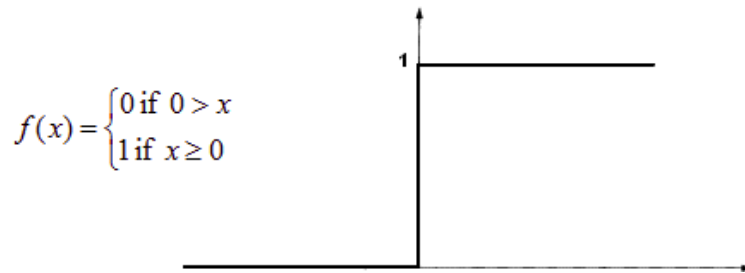


Figure 3.21 Step Activation Function

Depending on the application there are several types of activation functions which may provide non-linearity to our model, such as but not limited to:

A) Sigmoid Activation Function

The sigmoid function is one of the most widely used activation functions which provide a balance between linear and non-linear behavior. It has a characteristic "S" shaped curve whose output approaches 1 as $x \rightarrow +\infty$ and 0 as $x \rightarrow -\infty$. Such can be described by the function below:

$$\sigma(z) = \frac{1}{1 + e^{-\lambda z}} \quad \sigma : R \rightarrow [0,1]$$

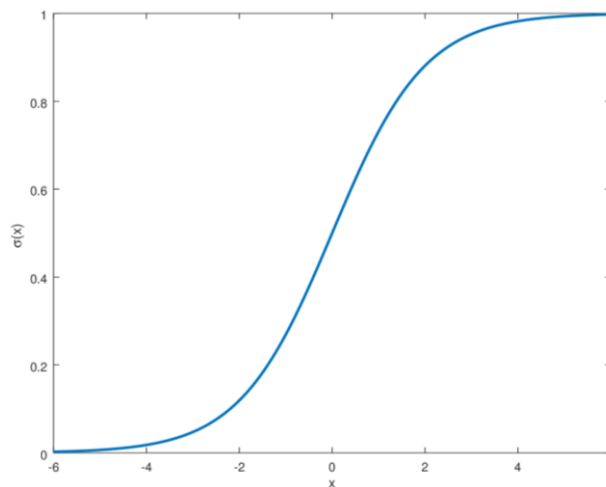


Figure 3.22 Sigmoid Activation Function

The $\lambda \in R$ is the steepness of the sigmoid function for fine-tuning the model. The sigmoid function appears to be effective in several neural network applications, although it sometimes outputs not so well calibrated results because near the edges of the function, the gradients tend to minimize and changes in the y axis do not respond to changes in the x axis. This stops the network from learning, an issue known as “Vanishing Gradient”.

B) Hyperbolic Tangent Function

The hyperbolic tangent activation function is similar to the sigmoid function (regarding its “S” shape), but it maps its inputs at a range of $[-1,1]$ instead of $[0,1]$

$$\sigma(z) = \frac{1}{1 + e^{-2z}} \quad \sigma : R \rightarrow [-1,1]$$

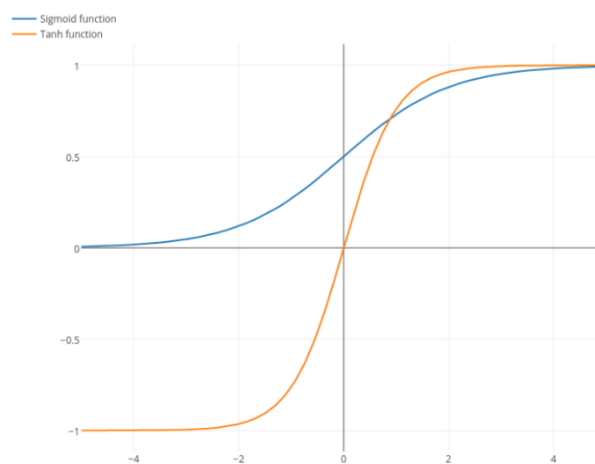


Figure 3.23 Tanh(x) with Sigmoid function comparison

Between the Sigmoid and the Tanh activation functions, both of them merge linearity with non-linear properties, but due to $\tanh(x)$ symmetry around the origin it can prevent the “Vanishing Gradient” issue and thus can lead to better convergence. Although it is more complex and depending the project, the sigmoid can produce better results.

C) ReLU (Rectified Linear Unit)

Amongst the most popular activation functions is the ReLU (Rectified Linear Unit) which will be also used in our current thesis. It is defined as:

$$\sigma(z) = (0, z) \quad \sigma : R \rightarrow [0, \infty)$$

Its key advantage is its computational efficiency since its activated only for positive inputs (z). Although, being zero at negative values the issue of “dying ReLU” arises, which may consistently outputs 0, resulting in a deactivated unit with poor performance. To address this issue there are different activation functions in the bibliography such as Leaky ReLU, Scaled ELU (SELU) and Parametric ReLU (PELU).

The figure of the ReLU Activation Function is shown below:

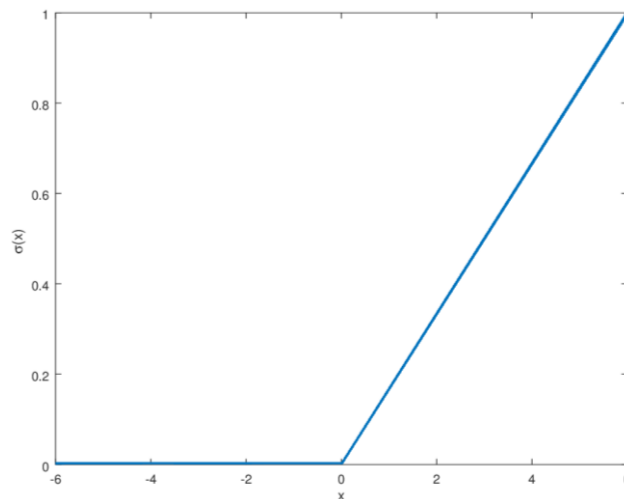


Figure 3.24 ReLU Activation Function

It is really important to choose a correct activation function as such will allow the network to learn complex relationships between the inputs and the outputs. A well-chosen activation function will improve the performance of the neural network and will reduce the risk of overfitting, speed the convergence of the network and allow it to train with high accuracy.

3.3.3 Neural Network Architecture and Principles

Having an adequate understanding of the Artificial Neural Networks, the early researchers attempted to model neural networks, although they faced limitations particular in computing power, thus the field stagnated for decades. Up until 1980-1990 with the introduction of GPUs for faster training, the first complete Neural Networks were developed. Neural Networks consist of multiple Perceptrons which are organized in layers as shown in the graph below.

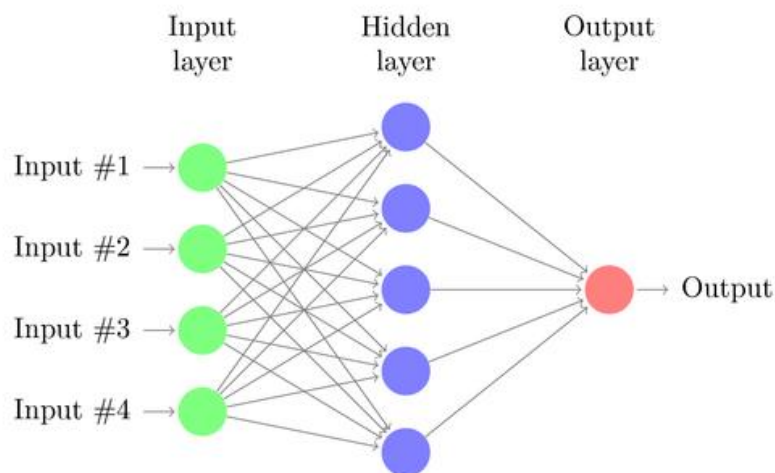


Figure 3.25 A simple Neural Network

There are 3 different types of Layers:

- Input Layer: Which is composed of neurons which receive the input data and pass them to the next Layer. The number of nodes is determined by the dimensionality of our model (for example, an object detection neural network would have as many input nodes as the image's pixels).
- Hidden Layers: Which are positioned between the input and the output layers. They are called "hidden" because their activations are not directly observable. Their purpose is to help the network learn more complex features passing the result to the next layer.
- Output Layer: Which is the final layer responsible for outputting the neural networks predictions. The number of output nodes depends of the dimensionality of our model (for example an object detection neural network would have as many input nodes as the number of classes for detection, with their value being the accuracy of the prediction).

When adding multiple hidden layers, we construct a more complex model, a Deep Neural Network (DNN). Those hidden layers allow the DNN to detect increasingly complex patterns in the input data thus increasing the accuracy of the predictions. Assuming a DNN of multiple Hidden Layers denoted $h_1, h_2, h_3, \dots, h_n$ the DNN can be presented by the following graph:

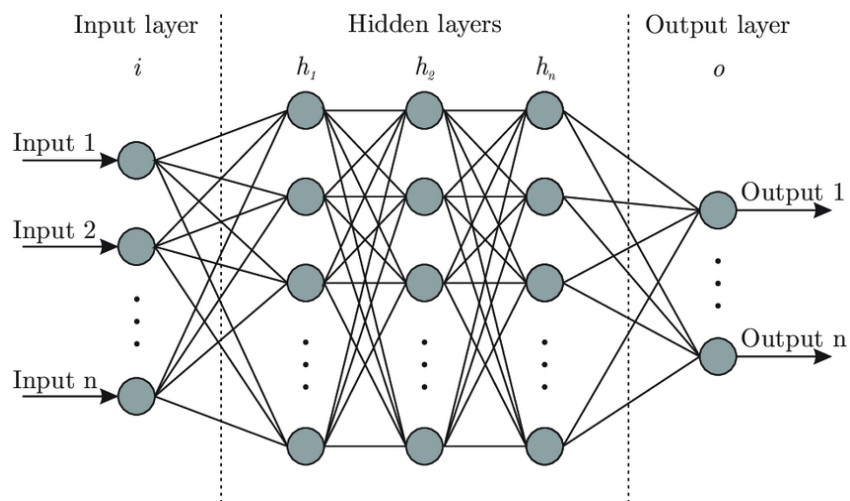


Figure 3.26 Deep Neural Network Architecture

Neural Network architectures may vary depending on the application, with the one above being a Feed Forward DNN because the data flows in only one direction, in case the network has a feedback loop as an extra input to the input layer, then this architecture is a Recurrent Neural Network (RNN). In this particular thesis we will focus our interest on another variation, Convolutional Neural Networks (CNNs) which will be discussed further in the next sections.

Depending on the architecture of the Neural Network, the model will be able to handle large and complex datasets, enabling it to perform demanding tasks in applications such as image detection, voice recognition and time series predictions. While multiple layers and complex neural networks may produce far more accurate results than simpler ones, they require much more computational resources to be developed and trained, they might overfit the input data and such complexity poses a challenge for the researchers to understand why the network made various decisions, to the point it might be treated as a “black box”. It is really important for various tasks to implement and trial/errors various neural network designs with varying layers and number of nodes as well as different activation functions prior choosing a final architecture.

3.3.4 Training Neural Networks and Backpropagation

Assuming we want to train the neural network of the figure below consisting of 3 inputs $\{x_1, x_2, x_3\}$ and 1 output $y_w = \sigma_w(x)$:

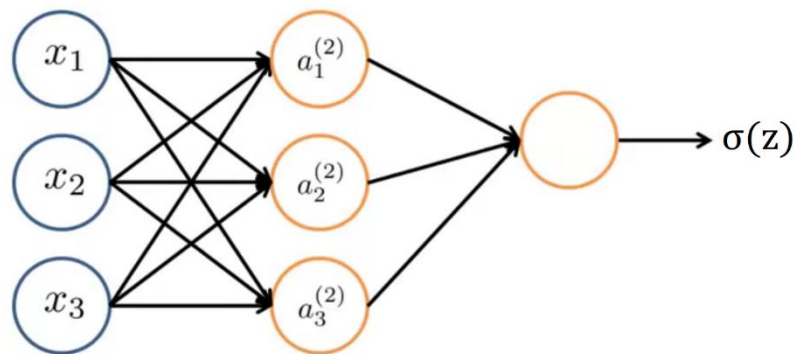


Figure 3.27 Neural Network with 1 Hidden Layer

We denote as $a_i^{(j)}$ = The activation unit i at the layer j and $w^{(j)}$ = the matrix of weights controlling the function mapping. We may perceive a mathematical representation of the neural network starting from the first activation units:

$$a_1^{(2)} = \sigma \left(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 \right)$$

$$a_2^{(2)} = \sigma \left(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 \right)$$

$$a_3^{(2)} = \sigma \left(w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3 \right)$$

Thus, the output would be:

$$y_w = \sigma_w(x) = a_1^{(3)} = \sigma \left(w_{11}^{(2)} a_1^{(2)} + w_{12}^{(2)} a_2^{(2)} + w_{13}^{(2)} a_3^{(2)} \right)$$

We introduce the variable $z_k^{(2)} = w_{k1}^{(1)}x_1 + w_{k2}^{(1)}x_2 + \dots + w_{kn}^{(1)}x_n$ for the layer $j = 2$ and node k , as we can rewrite the expressions above as follows:

$$a_1^{(2)} = g(z_1^{(2)}) \quad a_2^{(2)} = g(z_2^{(2)}) \quad a_3^{(2)} = g(z_3^{(2)})$$

The vector representation of the x and z^j is:

$$x = [x_1 : x_n] \quad z^{(j)} = [z_1^{(j)} : z_n^{(j)}]$$

If we denote $x = a^{(1)}$ we can rewrite the equation as:

$$z^{(j)} = w^{(j-1)}a^{(j-1)} \rightarrow z^{(j+1)} = w^{(j)}a^{(j)}$$

And for $a^{(j)} = g(z^{(j)})$ we may write our output result as:

$$\sigma_w(x) = a^{(j+1)} = g(z^{(j+1)})$$

If the network has s_j units in the layer j and s_{j+1} units in the layer $j + 1$ then the weights matrix $w^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

Loss Function & Backpropagation:

Supposing we have $L = 3$ Layers on our network (one input, one hidden and one output), and s_l the number of nodes in the layer l ($s_1 = 3, s_2 = 3, s_3 = 1$) and $K = 1$ the number of output nodes/classes, the Loss Function of can be defined as a more complex version of the logistic regression we seen already:

$$J(w) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K \left[y_k^{(i)} \log \left[\left(\sigma_w(x^{(i)}) \right)_k \right] + (1 - y_k^{(i)}) \log \left[1 - \left(\sigma_w(x^{(i)}) \right)_k \right] \right] + \frac{\lambda}{2m} \sum_{l=1}^L \sum_{i,j=1}^{s_l, s_{l+1}} \left(w_{ji}^{(l)} \right)^2$$

In order to minimize our loss function $J(w)$ and optimize the weights $w_{ji}^{(l)}$ similarly to the gradient descent we need to calculate the partial derivatives:

$$\frac{\partial}{\partial w_{ij}^{(l)}} J(w)$$

For that we may employ the Backpropagation algorithm:

Given a training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(1)}, y^{(2)})\}$ we set $\Delta_{ij}^{(l)} := 0$ for all (l, i, j) .

Backpropagation Algorithm

For the training examples $t = 1$ to m we perform the following steps:

- 1) Perform *Forward Propagation* to compute $a^{(l)}$ for $l = 2, 3, \dots, L$ as following:

$$a^{(1)} = x \rightarrow z^{(2)} = w^{(1)}a^{(1)} \rightarrow a^{(2)} = \sigma(z^{(2)}) \rightarrow z^{(3)} = w^{(2)}a^{(2)} \rightarrow a^{(3)} = \sigma(z^{(3)})$$

...

$$z^{(L)} = w^{(L-1)}a^{(L-1)} \rightarrow a^{(L)} = \sigma(z^{(L)})$$

- 2) Using $y^{(t)}$, perform Loss Computation $\delta^{(L)}$

$$\delta^{(L)} = a^{(L)} - y^{(t)}$$

L is the total number of layers and $a^{(L)}$ is the vector of outputs of the activation unit of the last layer. Hence the error values are the differences of the forward propagation result and the correct outputs in y .

- 3) Perform Backpropagation of Error, to obtain the error values of the layers before the last:

$$\delta^{(l)} = [w^{(l)}]^T * \delta^{(l+1)} .* a^{(l)} .* (1 - a^{(l)})$$

Till we calculate the values: $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

- 4) We calculate the $\Delta_{ij}^{(l)}$ necessary to perform the gradient calculations:

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

We repeat the process till we meet convergence, therefore we have a final Δ matrix which will be used to calculate the partial derivatives:

- 5) Calculate the Partial Derivative matrix and the Partial Derivatives:

If $j \neq 0$

$$D_{ij}^{(l)} = \frac{1}{m} (\Delta_{ij}^{(l)} + \lambda w_{ij}^{(l)})$$

If $j = 0$

$$D_{ij}^{(l)} = \frac{1}{m} (\Delta_{ij}^{(l)})$$

Thus, the partial derivatives are:

$$\frac{\partial}{\partial w_{ij}^{(l)}} J(w) = D_{ij}^{(l)}$$

- 6) Perform the weight updates, the weights of the neurons can be updated using the computed partial derivatives (gradients) using an optimization algorithm of our choice such as Gradient Descent.
- 7) We repeat the process until the loss function $J(w)$ reaches its minimum value.

3.3.5 Neural Network Performance Evaluation

Evaluating the performance of a neural network is an important step in the process of training and deploying a machine learning model. There are several metrics that can be used to evaluate the performance of a neural network besides the Mean Squared Error such as:

Accuracy: Accuracy is a simple and commonly used metric that uses the percentage of correct predictions made by the model, relatively to the total number of predictions. It can be computed using the following equation:

$$Accuracy = \frac{N_{correct}}{N_{total}}$$

For example, if our model makes 100 predictions and 90 of them are correct, the accuracy of the model would be 90%. It's important to understand that accuracy might not be the best metric to use as it can be misleading in cases where the dataset has an imbalanced class distribution where accuracy could be high, but the model might not perform well on the minority class. Thus, we introduce more metrics to evaluate its performance such as:

Precision and Recall: Assuming we want to predict a certain class (positive) and our model detects a number of predictions, some of them are truly positive (TP) and some of them are false positive (FP). Precision measures out of all the positive predictions (TP+FP) how many of them are truly positive:

$$Precision = \frac{TP}{TP + FP}$$

For example, Precision poses the question: "Out of all predicted shaft bearings, which objects are actually bearings?"

Moreover, we make predictions whether an entity is not of a certain class (negative) and our model predicts that some of them are truly not of that class (true negative – TN) but some of them are falsely not of that class (false Negative), making them real positive. Recall measures out of all real positive predictions (TP + FN) how many of them are truly positive:

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

For example, recall poses the question: "Out of all actual bearings (TP+FN), those who were correctly labeled as bearings (TP) and those who were falsely labeled as not bearings (FN), how many of them were correctly labeled as bearings?"

Both of those metrics provide us with a more nuanced understanding of the model's performance. High Precisions equals to the model being good at avoiding false positives, while high Recall equals to the model being good at detecting truly positive instances. Sometimes

high precision comes with a trade-off of recall, a visualization of which can be seen in a Precision-Recall Curve:

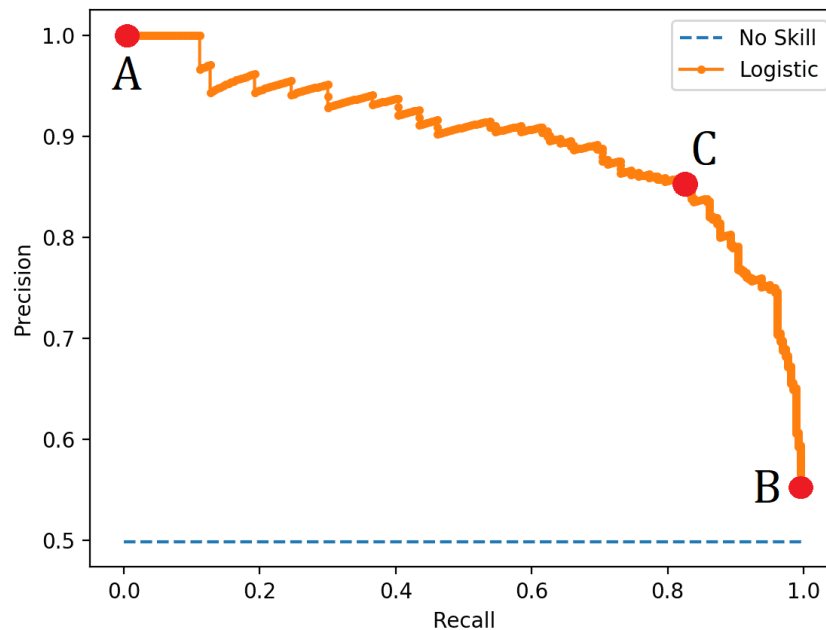


Figure 3.28 Precision - Recall Curve

Considering a compromise between precision and recall, it is recommended to complete the Neural Network training process when there is a good balance between precision and recall such as the point C in the figure above. A metric which applies such balance is

F1 Score: The F1 Score is a harmonic mean of precision and recall computed as below:

$$F1\ Score = 2 \frac{Precision \times Recall}{Precision + Recall}$$

The F1 Score combines the precision and recall in a single metric as it is being able to offer better perspective of the accuracy than the precision and recall alone. A high F1 Score indicates that the model is making accurate predictions with a good balance between precision and recall. On the contrary, a low F1 Score indicates an inaccurate model due to a poor balance between precision and recall.

F1 Score – Confidence curve: Confidence is the threshold we have set on our classifier (activator function) which determines the decision boundary for assigning the positive or negative class labels. When the threshold is increased, our model becomes more conservative in its predictions and only instances with high predicted probabilities will be assigned the positive class. This results in fewer false positives, thus an increase in *Precision* but on the same time the model may also miss some positive instances, which leads in a decrease in *Recall*. On the contrary when the threshold is decreased, reversely the *Precision* decreases and the *Recall* increases. In order to be able to define an optimal threshold/confidence where

there is a balance between the Precision and the Recall, we employ the F1 Score – Confidence curve:

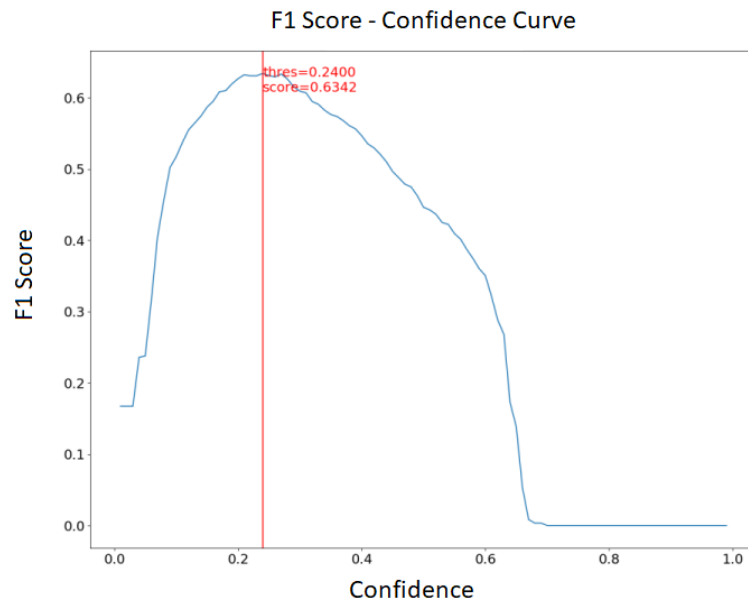


Figure 3.29 F1 Score - Confidence Curve

In the figure above we can see that we can optimize the threshold at 0.24 to get an F1 Score of 0.634. Although we aim for our curve to be as flat as possible to cover a wider variety of confidence values especially high ones, for a relatively large number of F1 Score.

Confusion Matrix: The confusion matrix offers a summary of the true positives, true negatives, false positives and false negatives of a model's performance. As mentioned above:

- True Positives (TP): The number of instances that are really positive and predicted as positive by the model.
- False Positives (FP): The number of instances that are really negative but falsely predicted as positive by the model.
- True Negatives (TN): The number of instances that are really negative and predicted as negative by the model.
- False Negatives (FN): The number of instances that are really positive but falsely predicted as negative.

Where positive is denoted as the desired class we want to detect and negative as the rest of the other classes. The confusion matrix provides a detailed breakdown of the model's performance and along with the precision and recall metrics we can diagnose how the model behaves, especially regarding imbalanced class issues. By visualizing the confusion matrix, we may be able to obtain a better understanding of how a trained neural network makes its predictions, thus being able to identify areas to improve.

A Confusion Matrix for a model which detects mechanical entities from shaft arrangement plans would be:

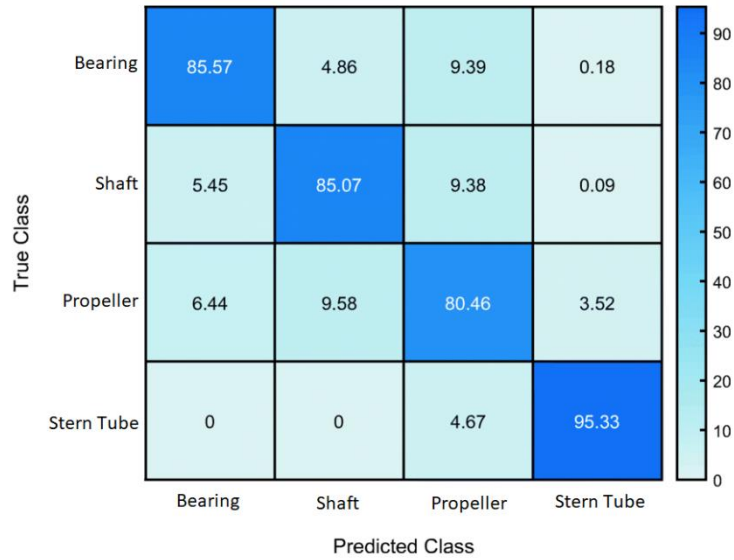


Figure 3.30 Confusion Matrix Example

The confusion matrix above has normalized values ranging from 0 to 100%. Each boxes number represents the percentage of the predicted class being the true, which is a measure of the Accuracy. For instance, for the bearing we have an accuracy of 85.57% (85.57% predicted bearings are actually bearings).

AUC – ROC Curve: A notable mention is the AUC-ROC Curve (Area Under the Curve – Receiver Operating Characteristics). It is a plot of the Recall (TPR) against the False Positive Rate:

$$FRP = 1 - \frac{FP}{TN + FP}$$

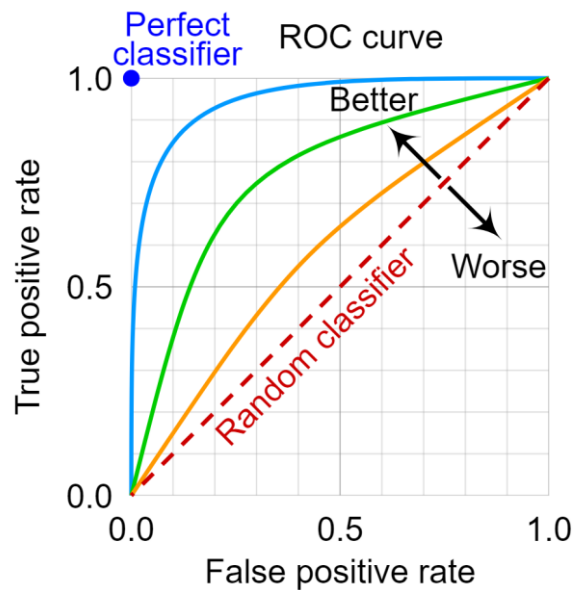


Figure 3.31 AUC-ROC Curve Example

A perfect model would have an AUC-ROC = 1 being always accurate while a random classifier would have an AUC-ROC = 0.5 being unable to distinguish positive from negative classes. We aim for our model to be as close to the perfect classifier as possible. In conclusion, besides the Mean Squared Error we ought to employ further metrics which will be able to improve the accuracy and the robustness of the neural network as well as being indicators for further improvement and development on the model itself.

3.3.6 Convolutional Neural Networks

Convolutional Neural Networks CNNs were first being developed by Yann LeCun, et al. (1980) in their paper “Backpropagation Applied to Handwritten Zip Code Recognition”. LeCun recognizes that DNNs and Feed Forward ANNs as fully connected networks are incapable of handling grid-sized (2D/3D) inputs such as images, because their large size results in an even larger number of parameters inside their hidden layers. To avoid this problem, LeCun was inspired by the function of the human visual cortex. The visual cortex features a hierarchical structure comprised of a series of layers which process different parts of the optical field and share their stimuli to generate the visual signals. The early layers, process simple features such as edges and lines and the later layers, process more complex features such as objects and scenes. The concept of having multiple levels of abstraction to recognize objects and patterns gave birth to the structure of the Convolutional Neural Networks. By implementing a weight sharing strategy between multiple layers for extracting features occurred in different locations of an image appears to drastically reduce the number of parameters required for the neural network to learn (e.g., the number of weights would no longer depend on the size of the input image).

The structure of a Convolutional Neural Network is composed of several layers which can be seen in the figure below consisting of Convolution, Pooling and Fully Connected Layers.

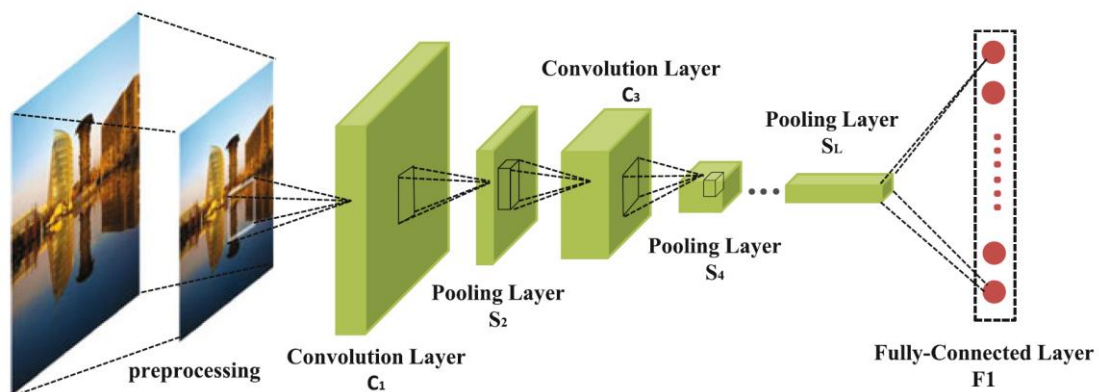


Figure 3.32 Structure of a CNN

Convolutional Layers

The Convolutional Layers are the most important building blocks of the CNN responsible for extracting features from the input image by applying a set of filters which produces a feature map representing the learned features. Reminding the chapter previously mentioned, our image can be represented by the matrix of the pixel intensities of the image with size equal to the number of pixels horizontally and vertically. A convolutional layer is actually an application of a linear filter which operates on convolution in order to produce a feature map.

In the convolution layer, a small kernel (the linear filter matrix) is slid across the input image and the values of the filter are multiplied with the values in the corresponding region of the input image. The result of these multiplications is then summed to produce a single output value which is stored in the feature map. Then, the filter slides to its next position and the process is repeated until the entire image is scanned. A detailed example may be illustrative to present how Convolutional Layers work:

Example

Let's consider the following image of a Shaft Bearing which for the sake of this example is digitized at a 6x6 input data matrix of pixel intensities:

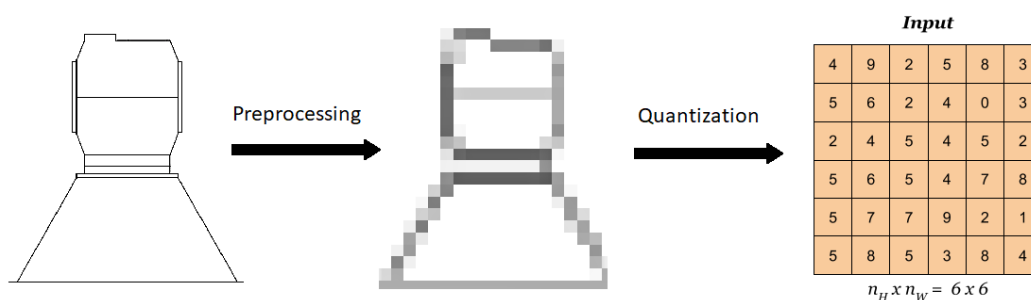


Figure 3.33 Image Convolution of Shaft Bearing Example

For the convolution layer we apply a 3x3 kernel (Kirsch filter) which is commonly used in edge detection. For this particular example we may use the kernel for detecting the edges in the y direction. For the convolution we may proceed using the following computational steps:

Step 1: Initiate the convolution by performing a scalar multiplication of the 3x3 kernel to the first 3x3 batch of pixels and sum their values to a single number output:

$$\begin{bmatrix} 4 & 9 & 2 \\ 5 & 6 & 2 \\ 2 & 4 & 5 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{matrix} 4 \cdot 1 + 9 \cdot 0 + 2 \cdot (-1) + \\ 5 \cdot 1 + 6 \cdot 0 + 2 \cdot (-1) + \\ 2 \cdot 1 + 4 \cdot 0 + 5 \cdot (-1) \end{matrix} = 2$$

The output of the multiplication is stored at the "Result" matrix which is the feature map as it can be seen from the figure below:

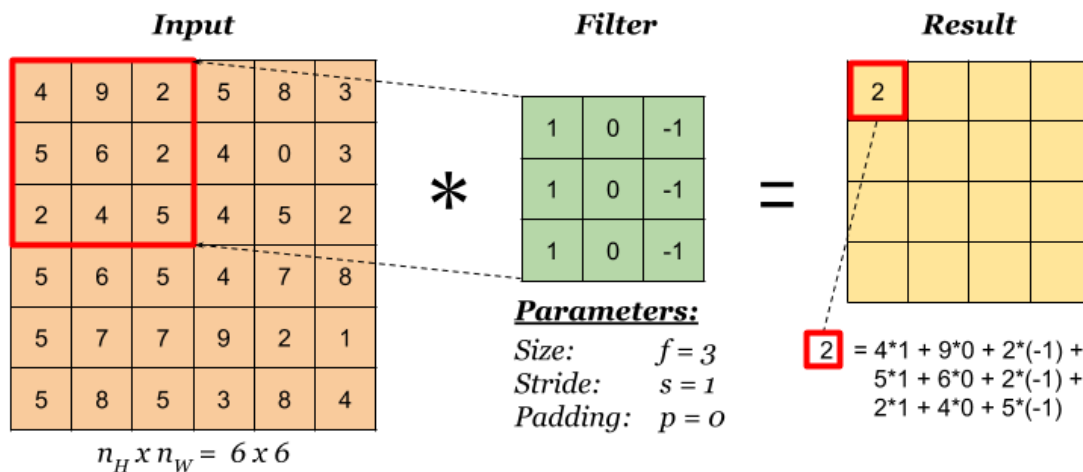


Figure 3.34 Step 1 of Convolution Example

Step 2: We slide the 3x3 batch by a number denoted as Stride ($s = 1$) which signifies how many steps further will the batch slide. We proceed by repeating the matrix multiplication with the new 3x3 batch and adding the result to the next position of the feature map.

$$\begin{bmatrix} 9 & 2 & 5 \\ 6 & 2 & 4 \\ 4 & 5 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{matrix} 9 \cdot 1 + 2 \cdot 0 + 5 \cdot (-1) + \\ 6 \cdot 1 + 2 \cdot 0 + 4 \cdot (-1) + \\ 4 \cdot 1 + 5 \cdot 0 + 4 \cdot (-1) \end{matrix} = 6$$

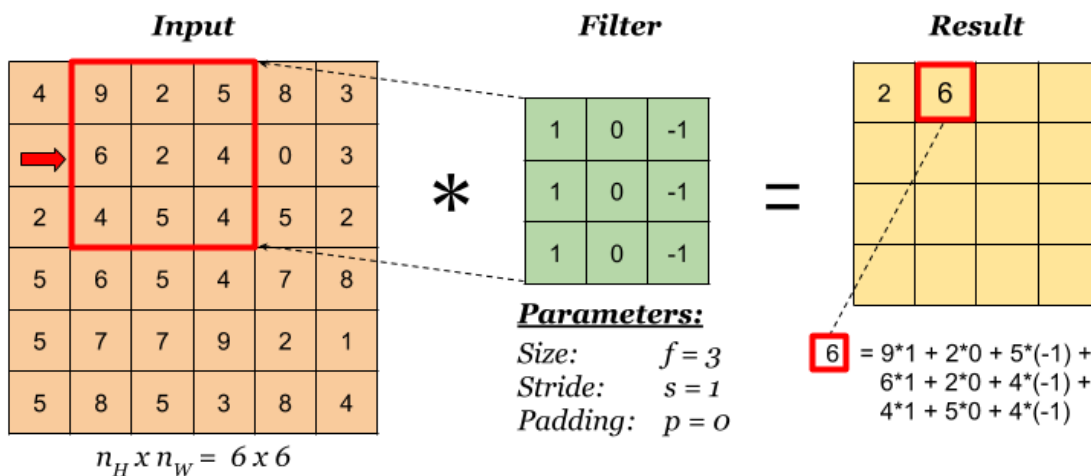


Figure 3.35 Step 2 of Convolution Example

Step 3: We repeat the multiplications using the steps above to complete and fill the feature map for all its rows and columns.

A really helpful parameter is the *padding value* which provides us with the following benefits:

- 1) It allows us to use the convolution layer without necessarily shrinking the height and width of the volumes, which is important for building deeper networks where the dimensions of the feature maps would shrink.
- 2) It helps us keep information at the border of an image, otherwise very few values at the next layer would be affected by pixels as the edges of the image.

Notably, when the input image has more than 3 channels (colored RGB image) the filter should have a matching number of channels as well. Thus we operate convolution on a volume, such means that in order to calculate one feature we perform convolution on each matching channel separately and then we add the result together:

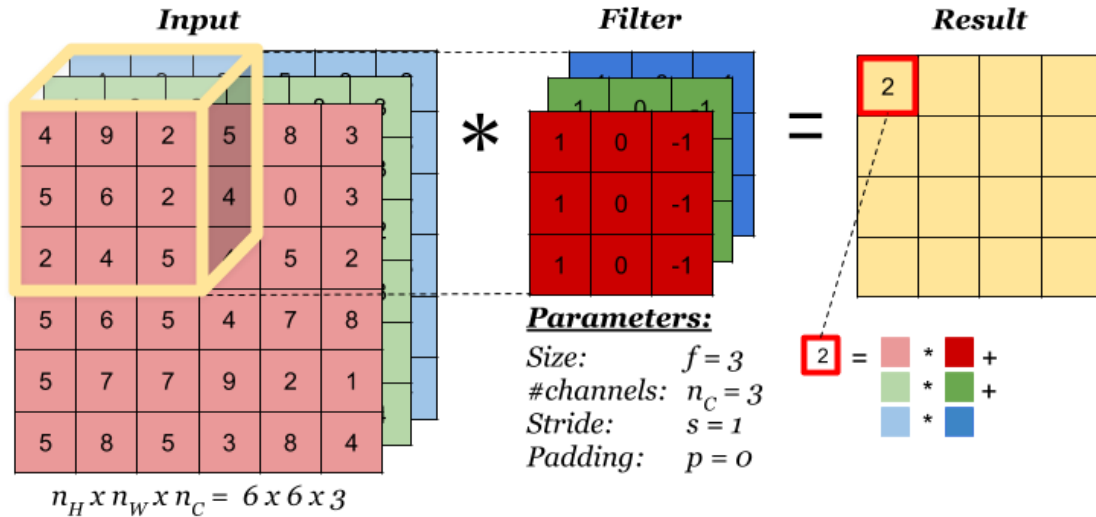


Figure 3.36 Convolution on a 3-channel image

Finally, to structure a convolution layer we may use more than one filter (for example 2) in order to output more channels from the image in a same manner we add multiple nodes in a CNN, which allows us to detect more features and extract hidden details. At the end of the convolution, we may add an activation function such as ReLU at the result matrix along with a bias b :

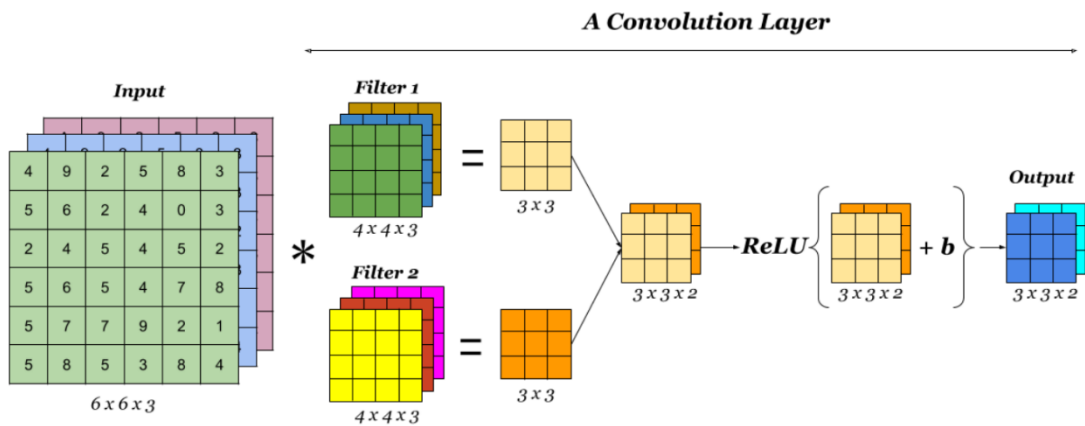


Figure 3.37A Fully detailed Convolution Layer

In order to shorten the such expression we may define a simpler representation of one convolutional layer with the following parameters: Size f , Stride s and Padding p with n the number of filters applied.

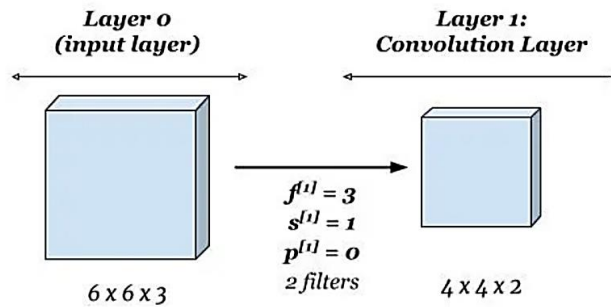


Figure 3.38 Convolution Layer representation

Using this notation, we may present a sample Convolutional Neural Network with three convolutional layers:

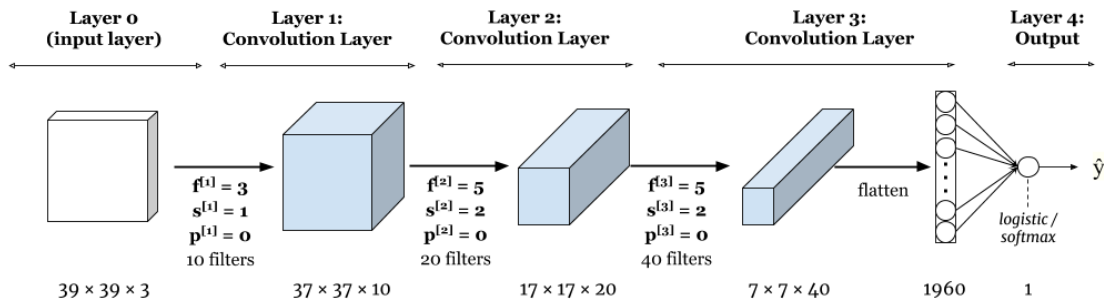


Figure 3.39 Convolutional Neural Network example

It is observed that each layer convolution with an image produces a single channel output (feature map). Thus, n filters produce an outcome of n channels. As the network progresses the output of the convolutional layer is getting flattened and flattened up until it is connected to a Fully Connected Layer. The Fully Connected Layer takes $7 \times 7 \times 40 = 1960$ inputs x_i and using a single logistic regression or softmax regression filter the final output \hat{y} is generated.

Notably if we were to develop a Feedforward ANN consisting of 1 hidden layer with 100 neurons, we would have required $39 \times 39 \times 3 = 4563$ input neurons and each neuron in the hidden layer would have 4563 connections to the input (plus a bias term) for a total of: $(4563 + 1) \cdot 100 = 456,400$ parameters. The output layer would have 100 neurons with each neuron connected to all 100 neurons of the hidden layer (plus a bias term) consisting a total of: $(100 + 1) \cdot 100 = 10,100$ parameters. Thus, the total number of parameters in the Feedforward neural network would be $456,400 + 10,100 = 466,500$. On the contrary the convolutional above we used would be:

- 3x3 kernel with 10 filters for 3 channels plus bias for each filter: $3 \cdot 3 \cdot 10 \cdot 3 + 10 = 280$
 - 5x5 kernel with 20 filters for 10 channels plus bias for each filter: $5 \cdot 5 \cdot 20 \cdot 10 + 20 = 5020$
 - 5x5 kernel with 40 filters for 20 channels plus bias for each filter: $5 \cdot 5 \cdot 40 \cdot 20 + 40 = 20040$
 - Fully Connected layer with 1960 inputs, making in total making a sum of: 27300 parameters
- Comparing to a reasonably designed ANN for the same task we required ~ 17 times more parameters than a CNN.

In order to speed up calculations inside a CNN we may employ Pooling Layers between the Convolutional Layers:

Pooling Layer: A pooling layers are used to downsample and reduce the spatial dimensions of feature maps produced by the convolutional layers as well as to make the features detected more robust. There are 2 different types of pooling, max pooling and average (avg) pooling but with most commonly used being the max pooling.

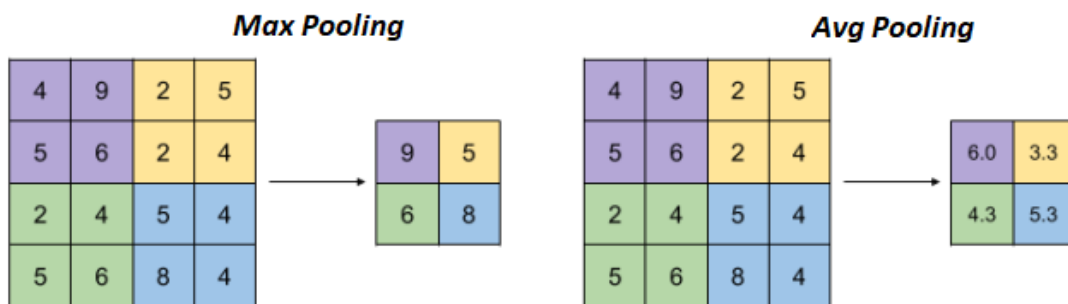


Figure 3.40 Max Pooling and Avg Pooling

As it is seen in the figure above, the pooling layer partitions the input feature map into batches of size f , calculates the max/average based on the pooling type and slides to the next batch with a stride s . It is important to understand that the pooling layer reduces only the height and the width of the image, but not the number of channels:

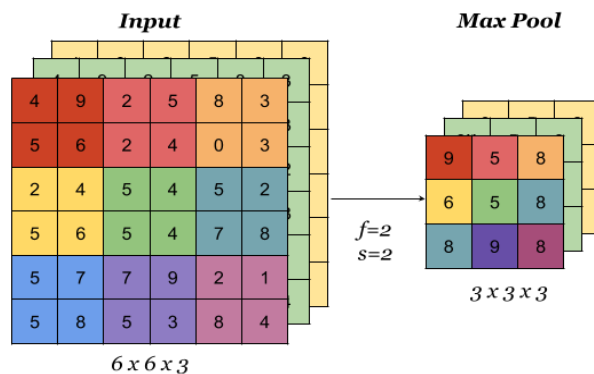


Figure 3.41 Pooling on Volume

Notable CNN Architectures

Some notable CNN architectures which have been studied and used by the academia throughout extended research are:

LeNet: LeNet was a pioneering CNN architecture which was proposed by Yann LeCun, et al. (1998) and it was one of the first successful attempts to employ a CNN for character recognition tasks such as recognizing handwritten digits. It was designed to accept a 32×32 grayscale image and classify it into 10 output classes corresponding to the digits 0 till 9. The LeNet structure can be seen in the figure below:

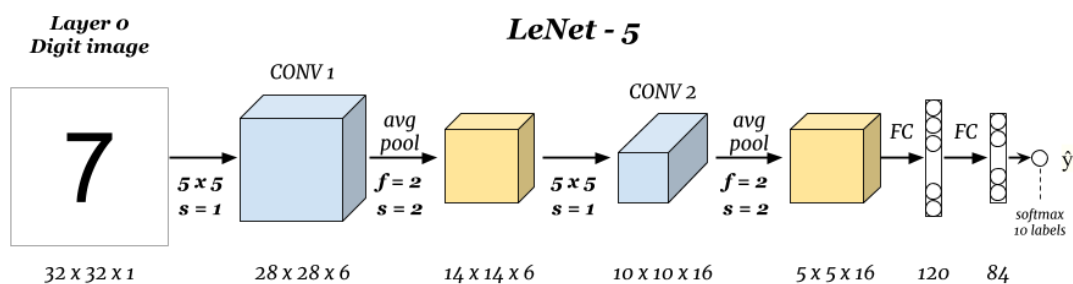


Figure 3.42 The LeNet Architecture

It was trained using Backpropagation and Stochastic Gradient Descent optimization, achieving high accuracy on the MNIST dataset consisting of handwritten digit images.

AlexNet: AlexNet was proposed by Alex Krizhevsky, et al. (2012) and it was designed to classify images from the ImageNet dataset which contains over 1 million images and 1000 categories.

The AlexNet structure can be seen in the figure below:

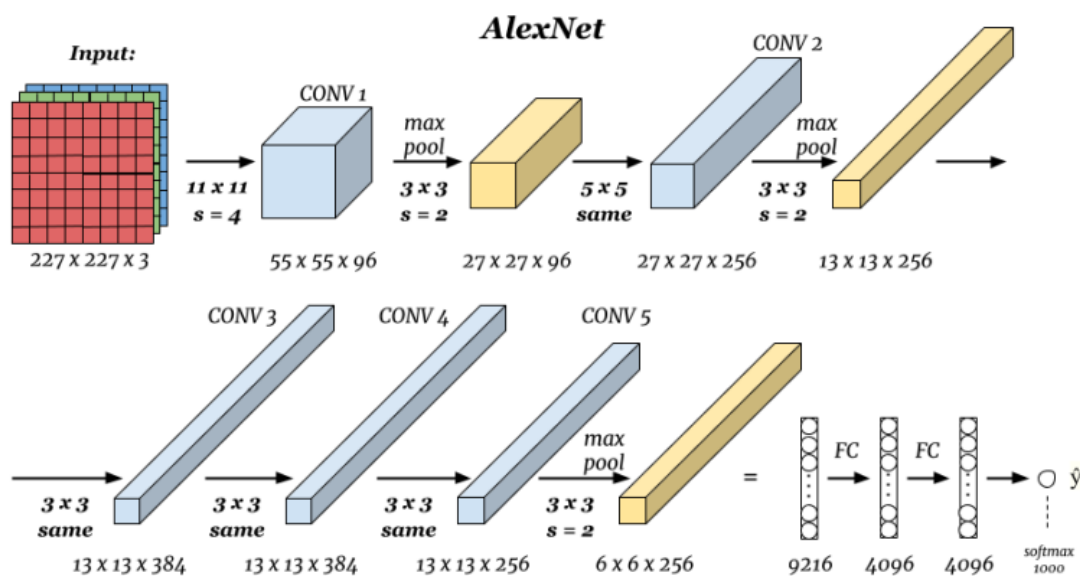


Figure 3.43 AlexNet Architecture

3.3.7 Object Detection using CNNs

Object detection is considered a fundamental problem in computer vision which involves: Identifying and Localizing objects within an image or a video sequence. The aim of object detection models is to:

- A) Classify the objects present in an image.
- B) Determine their precise location and boundary extent.

The output of an Object Detection process is a set of object classes which have been detected and their location usually as rectangular boundary defined as *bounding box*. Object detection has always been a challenging integral amongst computer vision researchers as it requires the ability to recognize objects being in different classes, scales, orientations and to handle variations in lighting, viewpoint occlusion and background clutter.

In recent years, there has been significant progress regarding object detection using deep learning, particularly with the development of CNNs which as we mentioned previously can learn features and classifiers directly from the raw pixel data. Object detection found numerous applications in the scientific and engineering field of research as well as in the maritime industry with the development of autonomous ships, AI surveying and fault detection/prediction and the field continues to be an active area of research with multiple challenges and opportunities for further improvement.

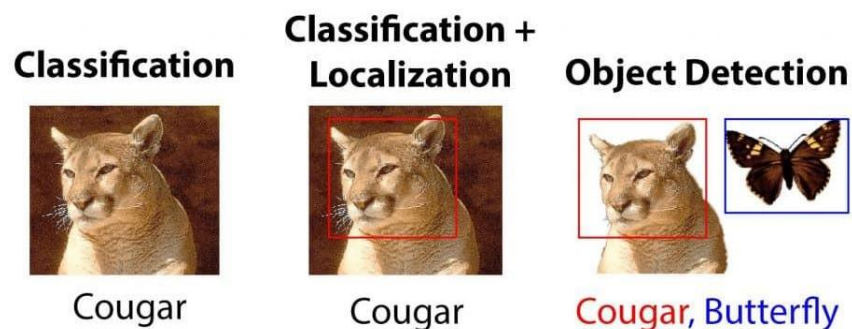


Figure 3.44 Classification and Localization on Object Detection

Object Detection Evaluation Metrics

In order to evaluate and compare the predictive performance of different object detection models, we may define some metrics specifically for the Object Detection process. The most commonly used in Object Detection are the Intersection over Union (IoU) and the Mean Average Precision (mAP)

Intersection Over Union (IoU): Considered one of the most popular metrics, the IoU measures localization accuracy and calculates localization errors in object detection models. In order to

calculate the IoU between the predicted and the ground truth bounding boxes we first take the intersecting area between the 2 corresponding bounding boxes for the same object. Following this, we may calculate the total area covered by the two bounding boxes defined as their “Union” and the area of overlap as their “Intersection”. The Intersection divided by the Union gives us the ratio of the overlap to the total area and it represents how close the predicted bounding box is to the original one:

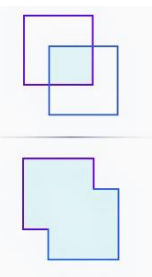
$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} =$$


Figure 3.45 Definition of IoU

Average Precision (AP) & Mean Average Precision (mAP): The Average Precision of a certain class is defined as the area under its Precision-Recall curve as we have already discussed. For all the given classes on the object detection model, the Mean Average Precision is a mean of all the APs we have calculated for all available classes:

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i$$

Additionally, the Mean Average Precision over different IoU thresholds ranging from 0.5 to 0.95 is one of the most common metrics to consider denoted as: $mAP_{[0.5-0.95]}$. In order to evaluate different Object Detection models, it is widely accepted amongst the academia to compare and validate them with some standardized image datasets such as the MS COCO dataset and the PASCAL VOC dataset.

Methods of Object Detection using CNNs

Primarily the methods of Object Detection using CNNs can be divided into 2 main categories: 2-stage methods and one stage methods. Two stage methods firstly generate some candidate object proposals and then classify those proposals into the specific categories defined. One stage methods simultaneously extract and classify all the object proposals. According to X. Jang et al (2019), two stage methods have a relatively slower detection speed but higher detection accuracy comparing to the one-stage methods which have a much faster detection speed mitigating a portion of their accuracy. Before employing a certain object detection

method is important to weigh the benefits of each method and compromise the model accordingly.

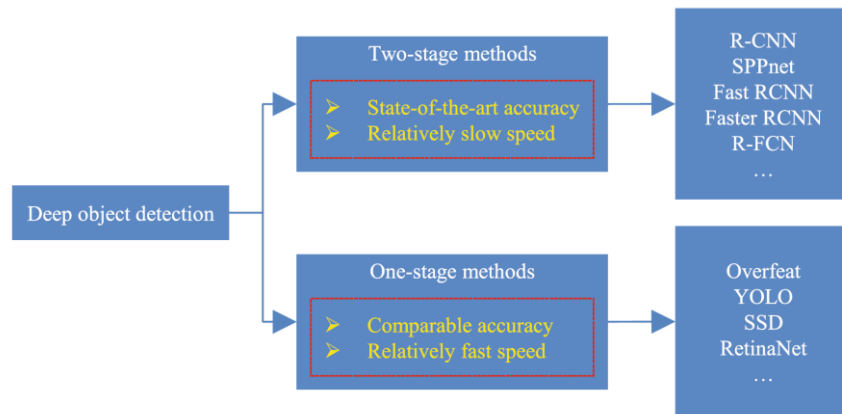


Figure 3.46 One-Stage Methods compared to Two-Stage Methods

Two-Stage Methods for Object Detection

Two-Stage Methods treat object detection as a multistage process comprising of a region proposal stage and an object detection stage. The main concept is to initially use a Region Proposal algorithm to generate a set of candidate object proposals and then use a separate object detection network to classify and refine those proposals. Using a two-stage method provides benefits such as reducing a potentially large number of proposals which are used in the classifier and providing robustness because the generated proposals allow the classifier to focus on the classification task with minimal influence from the background. The most commonly used Two-Stage Methods are:

A) R-CNN (Regional Convolutional Neural Network)

The R-CNN is consisting of a Regional Proposal step where a set of region proposals are generative using Selective Search. Selective search is a common region proposal algorithm which uses segmentation to group similar regions together. The output of this step is a set of bounding boxes which are likely to contain an object. Then the output passes to a Feature Extraction step which involves passing the bounding boxes through a pre-trained CNN (such as AlexNet) to extract the feature vector from the output of the CNN. The next step involves the classification of each region proposal into one of the predefined object classes by training a Support Vector Machine (SVM) for each class. And finally, the R-CNN refines the region proposals by applying bounding box regression. Bounding box regression adjusts the coordinates of the bounding box to better fit the object in proposal.

R-CNN has been highly successful in object detection tasks (for example mAp=21% at the PASCAL VOC2010), however it appears to have a slow runtime due to the computationally

expensive feature extraction step and its inability to jointly optimize the region proposal and classification steps.

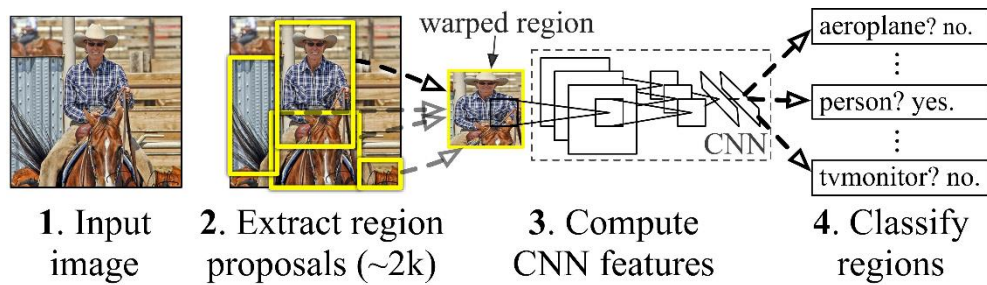


Figure 3.47 R-CNN Steps and Architecture

B) Faster R-CNN

As an improvement to the R-CNN Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun in 2015 developed Faster R-CNN. Its key improvements lay in its structure. Instead of a Selective Search algorithm, Faster R-CNN consists of a Region Proposal Network (RPN) a CNN for generating object proposals from a CNN backbone combined with a Feature Sharing technique called RoI (Region of Interest) Pooling which allows Faster R-CNN to share the CNN's features across the RPN and the object detection network, unlike the original R-CNN. This allows the Faster R-CNN to achieve greater speed and accuracy over the original R-CNN, paving the way to modern object detection.

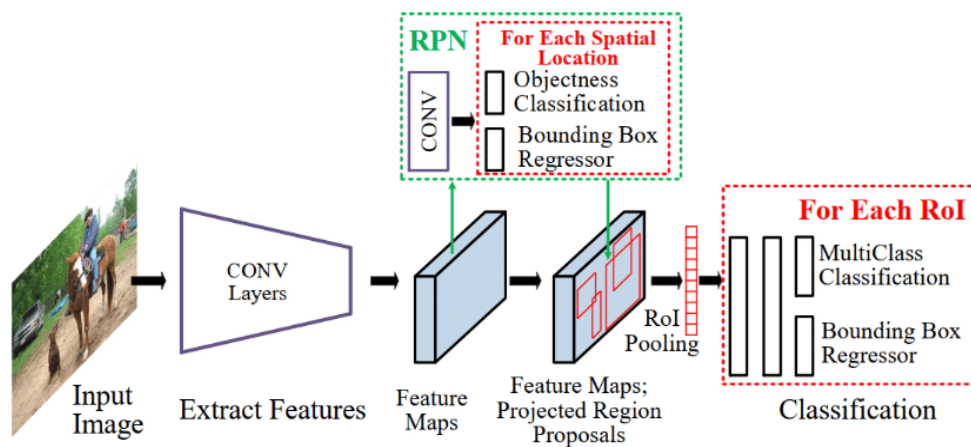


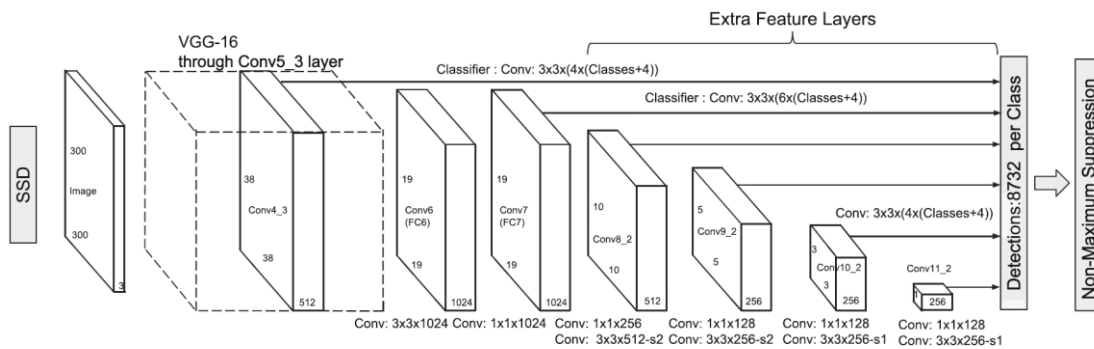
Figure 3.48 Faster R-CNN Steps and Architecture

One-Stage Methods for Object Detection

Unlike Two-Stage methods, One-Stage methods aim to simultaneously predict the object location and its class. Compared to the Two-Stage methods, One-Stage methods have much faster detection speed while maintaining a high degree of detection accuracy. The most important methods are the SSD and YOLO.

A) SSD (Single Shot Detector)

SSD is considered a well robust object detection algorithm. Its architecture consists of a base CNN which is usually a pre-trained CNN such as VGG-16, followed by several convolutional layers. Those multiple convolutional layers are of different resolution in order to predict objects of different scales thus being able to detect objects of different sizes. The SSD employs a set of predefined anchor boxes at different scales and aspect ratios for each prediction layer. For each anchor box, SSD predicts the class probabilities using a SoftMax function to the output of the convolutional filters as well as the bounding box offsets using a set of convolutional filters which output the offset between the anchor box and the ground truth bounding box. After the class probabilities and bounding box offsets have been predicted for each anchor box, the SSD applies Non-Maximum Suppression (NMS) filtering, to remove overlapping detections with lower confidence scores while keeping the ones with the highest confidence score.



B) YOLO (You Only Look Once)

YOLO (You Only Look Once) was first proposed by Joseph Redmon, et al. (2016) in their paper "You Only Look Once: Unified, Real-Time Object Detection" (2016), achieving a great performance in both terms of accuracy and speed on the PASCAL VOC dataset. Its architecture consists of 24 convolutional layers followed by 2 fully connected layers. The first 20 convolutional layers of the model are pre-trained using ImageNet. YOLO's main concept is to divide the input image into a $k \times k$ grid. If the center of an object falls into a grid cell, then that grid is responsible for detecting that object. Each grid cell predicts B bounding boxes and respective confidence scores for those bounding boxes. The confidence scores reflect how confident the model is that the box contains an object and how accurate it thinks the predicted bounding box is.

YOLO predicts multiple bounding boxes for each grid cell. During its training, YOLO assigns one predictor to be responsible for predicting an object based on which prediction has the highest IoU with the ground truth bounding box. As a result, this may lead to specialization between

the bounding box predictors, whilst each predictor gets better at forecasting certain sizes, aspect ratios or classes of objects, leading to a great improvement of the model's overall recall.

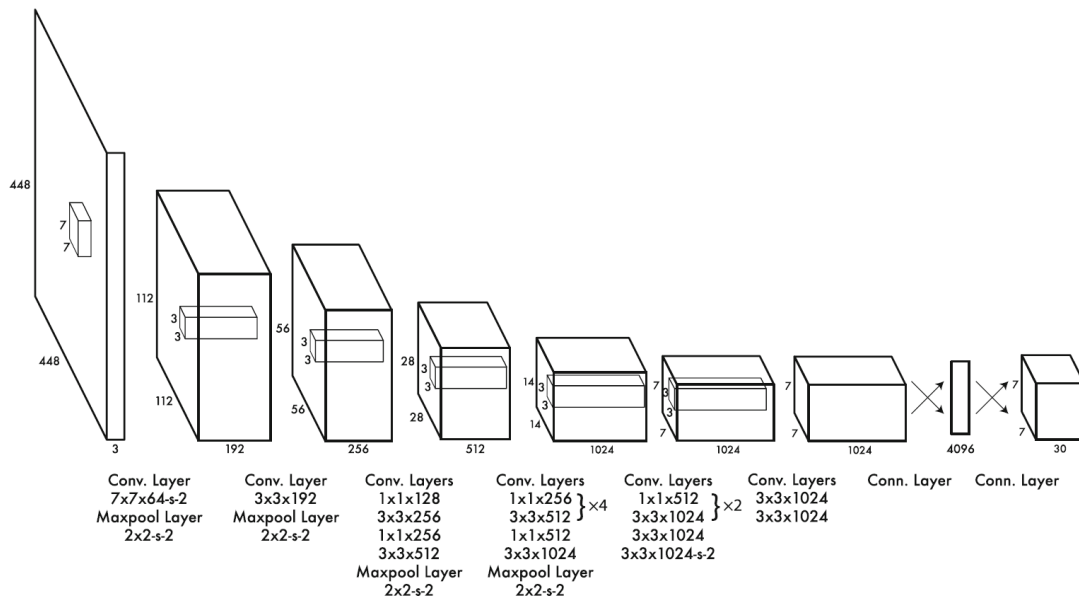


Figure 3.49 The YOLOv1 Architecture

Just like SSD, YOLO employs NMS filtering to identify and clear redundant and incorrect bounding box for each object in the image. YOLO and especially its future versions are considered to be one of the modern and best common practice approach model when it comes to object detection applications. YOLO will be the object detection model we will be using on this current thesis.

3.3.8 YOLO Versions and Improvements

Since its release, YOLO has undergone several versions and improvements in order to increase its robustness, accuracy and speed. The first YOLOv1 even though it provided fast detection, it suffered from lower accuracy compared to SSD. Since then, there have been several upgrades to the YOLO architecture, notably:

YOLOv2: YOLOv2 was introduced in 2016 as the first improvement over the original YOLO algorithm, designed to be fast and more accurate than YOLO and being able to detect a wider range of object classes. YOLOv2 uses a different CNN backbone, Darknet-19 which is a VGGNet variant with simple progressive convolution and pooling layers. A key improvement of YOLOv2 is the usage of anchor boxes just like SSD did in order to handle a wider variety of object sizes and aspect ratios. Additionally, YOLOv2 uses batch normalization, which improves the accuracy and stability of the model as well as a multi-scale training strategy which according

to Redmon (2016) involves training the model on images at multiple scales and then averaging the predictions for improving the detection performance on small objects. Last but not least, YOLOv2 integrates a new loss function which considered a weighted loss of the classification (Binary cross-entropy loss), localization (SSE) and confidence loss (IoU Binary cross-entropy loss). As a result, YOLOv2 improves significantly to its competitor models in both accuracy and speed, making it the one-way choice for researchers to develop their Object Detection applications.

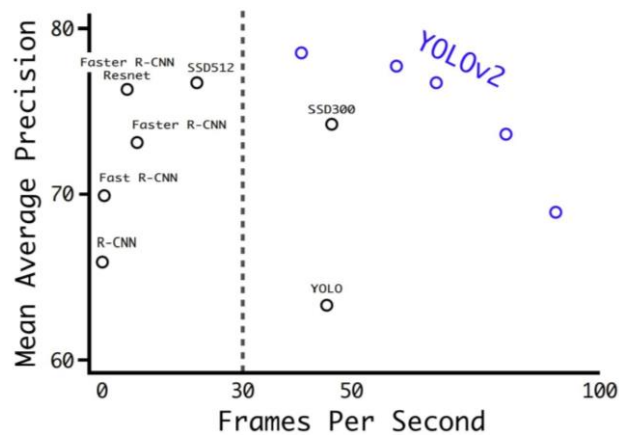


Figure 3.50 YOLOv2 Comparison Graph

YOLOv3: YOLOv3 was introduced at 2018 to further improve the YOLO model. It improves its backbone CNN using Darknet-53 which is a variant of the ResNet architecture with 54 convolutional layers, designed specifically for object detection with great accuracy. YOLOv3 features improved scaled anchor boxes with varied aspect ratios in order to improve the detection of objects at different aspect ratios and scales. Additionally, it introduces the concept of Feature Pyramid Networks (FPNs) which are CNNs constructing a pyramid of feature maps with each level of the pyramid used to detect objects at different scales. As a result, YOLO may now be able to handle a wider range of sizes and aspect ratios with being more stable and robust.

YOLOv4: YOLOv4 was introduced at 2020 by Alexey Bochkovskiy, et al. (2020) and features a novel CNN backbone named CSPNet (Cross-Stage Partial Network). Despite having a really shallow structure it achieves top-notch results on object detection benchmarks. YOLOv4 improves its anchoring methods with a k-means clustering algorithm to help in grouping the ground truth boxes into clusters and use the centroids of the clusters as anchor boxes. This allows the anchor boxes to be more closely aligned with the detected objects' shape and size.

In order to improve detection on imbalanced datasets, YOLOv4 features an improved loss function named GHM Loss (Gradient Harmonized Miscalibration). Whereas traditional loss functions such as focal loss and cross entropy treat all misclassifications equally and penalize them with the same weight, leading to the model being biased to the majority class, GHM measures the discrepancy between the predicted distribution and the predicted loss distribution and then rescales the loss gradients resulting in improved accuracy and robustness.

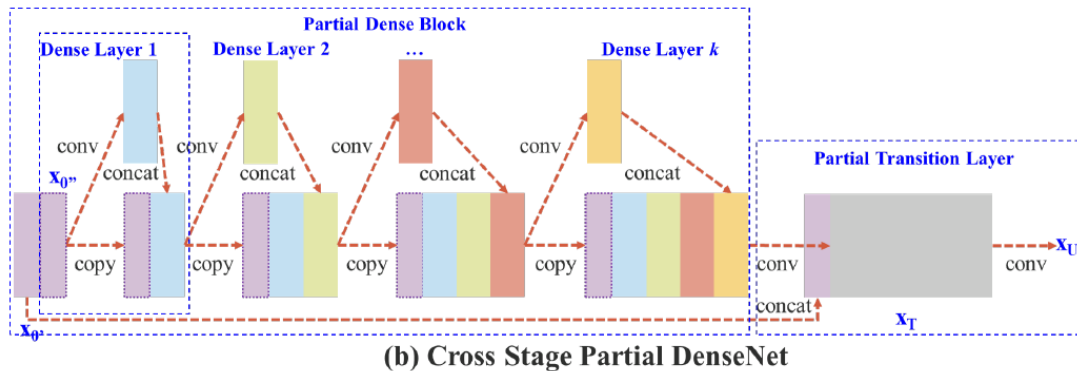


Figure 3.51 CSPNet Architecture

YOLOv5: YOLOv5 was introduced in 2020 and continues its improvements like the previous versions did. YOLOv5 features a complex backbone CNN based on the EfficientNet architecture, allowing higher accuracy and better generalization to a wider variety of objects. Unlike the previous YOLO versions which were trained on the PASCAL VOC dataset, YOLOv5 was trained on a larger and more diverse dataset called D5. YOLOv5 further enhances the anchor box generation using “dynamic anchor boxes” as an improved clustering technique. Additionally, YOLOv5 features “Spatial Pyramid Pooling” (SPP) a type of pooling layer to reduce spatial resolution on the feature maps while enabling detection at multiple scales. Last but not least, YOLOv5 introduces a new IoU loss variant the CloU loss which is designed to further improve the model’s performance on imbalanced datasets.

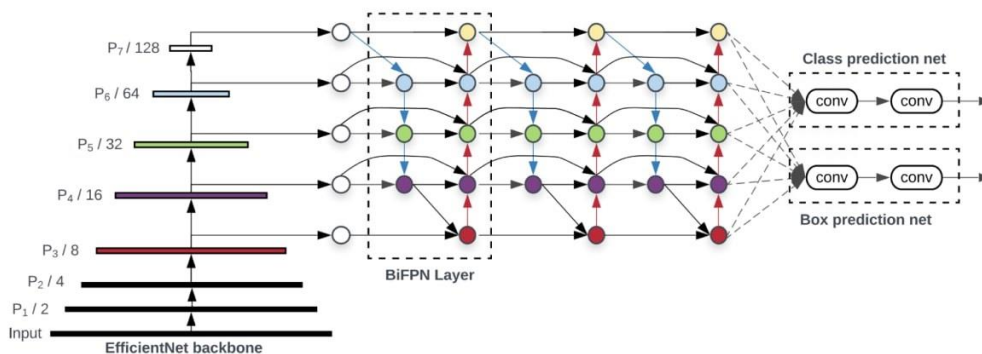


Figure 3.52 YOLOv5 EfficientDet Architecture

YOLOv6: YOLOv6 was introduced in 2022 by Li, et al. (2022) further improving the YOLO model. YOLOv6 features a new variant of the EfficientNet architecture named EfficientNet-L2. It is more efficient than YOLOv5's with fewer parameters and better efficiency. YOLOv6 improves its anchor box generation featuring "dense anchor boxes" which are able to capture small objects which might have been missed from traditional anchor box generation methods.

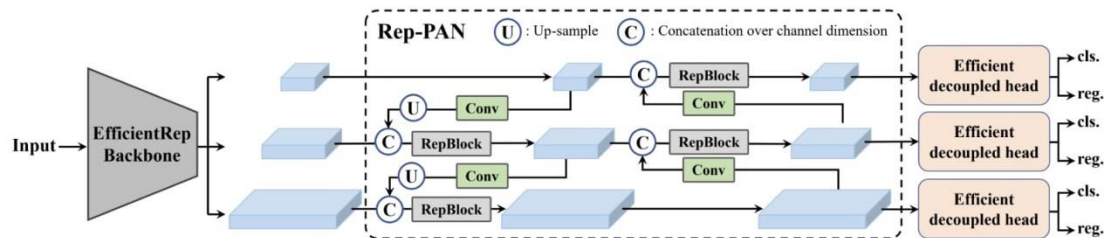


Figure 3.53 YOLOv6 Architecture

YOLOv7: YOLOv7 was introduced in July 2022 by Chien-Yao Wang and Alexey Bochkovskiy (2022). It features an improved backbone DNN, the Extended Efficient Layer Aggregation Network (E-ELAN) which consists of Convolutional Layers followed by a set of aggregation modules, notably named Multi-Level Channel Attention (MCA) modules which perform both spatial and channel attention operations in order to capture even the smallest of the object features. It also features Global Feature Integration which integrates the features from all convolutional layers and the MCA modules in combination with a novel gating mechanism to control the flow of information amongst the convolutional layers, something which significantly improves its detection accuracy and speed.

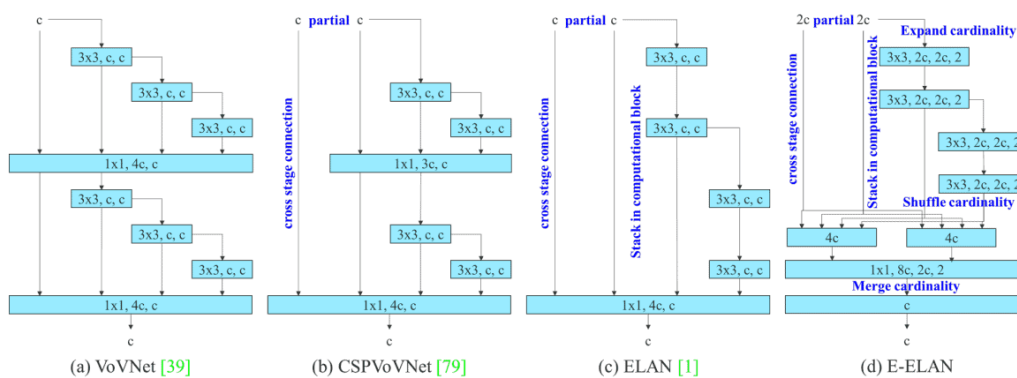


Figure 3.54 YOLOv7 E-ELAN Architecture

Not long after YOLOv7's release, YOLOv8 was released which is considered the most contemporary and state-of-the-art object detection model by Ultralytics LLC.

3.3.9 YOLOv8 – A modern approach to Object Detection

YOLOv8: YOLOv8 is currently the most state-of-the-art YOLO version being released at January 2023 by Ultralytics LLC and it is currently under development with multiple updates and an active community surrounding it. YOLOv8 at the moment this thesis being written does not have a published paper yet, thus our source comes only from the GitHub YOLOv8 repository which is sustained by Ultralytics. According to the figure below, YOLOv8 is comprised of a backbone which is a series of Convolutional Layers which pool the input image into multiple resolution sizes and the features are pooled together into the YOLOv8 head which is used to make the predictions based on an improved combination of box loss, class loss and objectness loss.

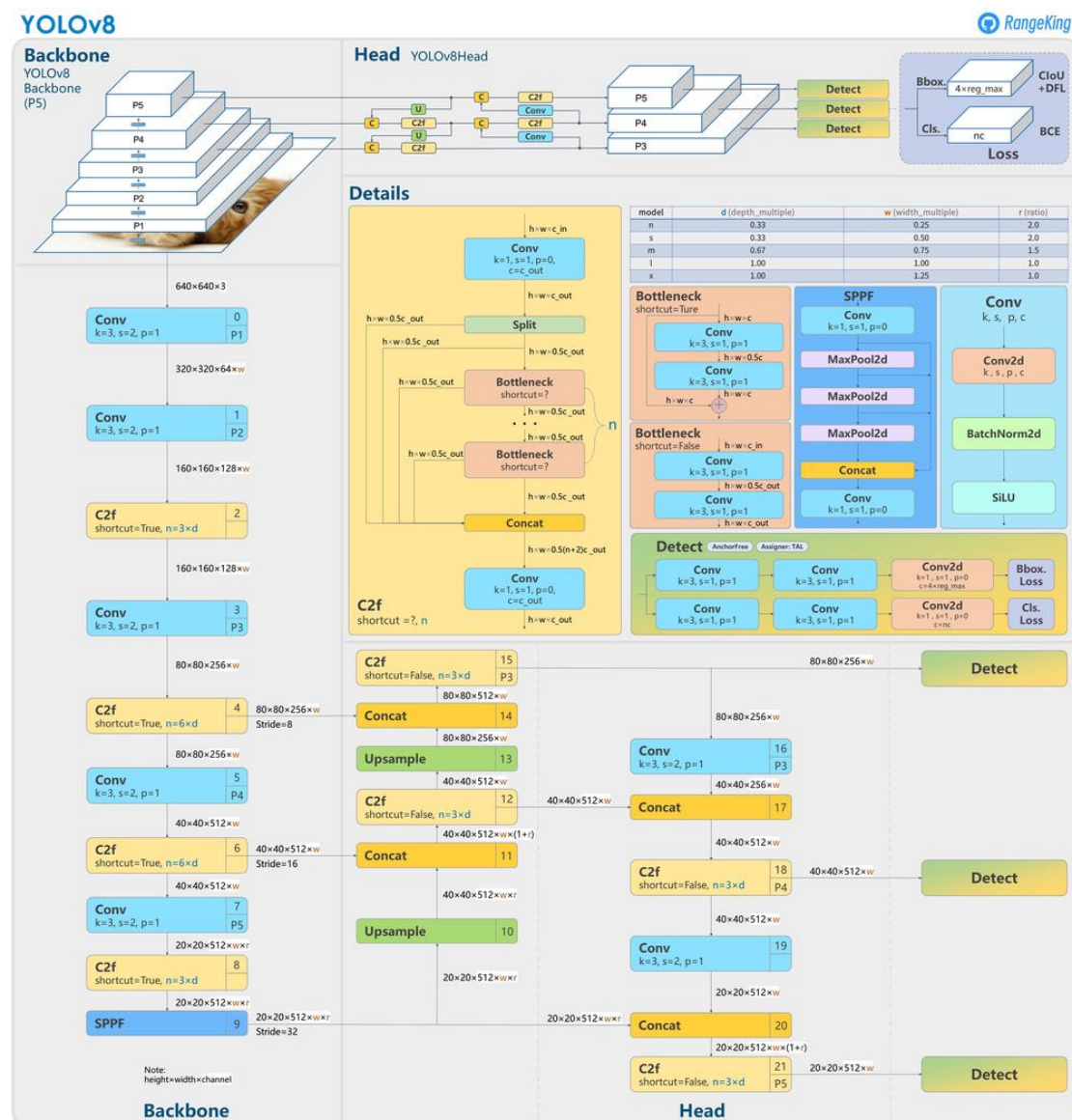


Figure 3.55 YOLOv8 Architecture

One of YOLOv8's greatest improvement amongst the rest of the versions is that it's an anchor-free model. YOLOv8 does not predict based on bounding box anchors which are proven to be not effective into detecting objects where it's one dimension is very greater than the other ($h > w$). As an anchor-free model with a novel head layer, appears to significantly improve and speedup Non-Maximum Suppression and thus enhancing the detection speed of the model. Additionally, new convolutions are being made on the network and the layers' structure has been improved.

Moreover, during YOLOv8's training process, the model augments images using mosaic augmentation, this involves stitching 4 images together, forcing the model to learn objects in new locations in partial occlusion and against surrounding pixels.

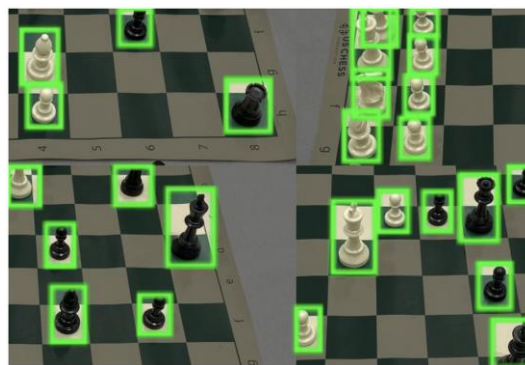


Figure 3.56 YOLOv8 Mosaic Augmentation

According to Jacob Solawetz, YOLOv8 is considered the best performing YOLO version from all the previous ones because of its high rate of accuracy at the COCO and RF100 datasets, as well as its continuous development and support from a growing community. More specifically:

COCO Dataset

YOLOv8 achieves a staggering 53.9% mAP on the COCO Dataset comparing to the previous versions of YOLO as it can be seen from the figure below

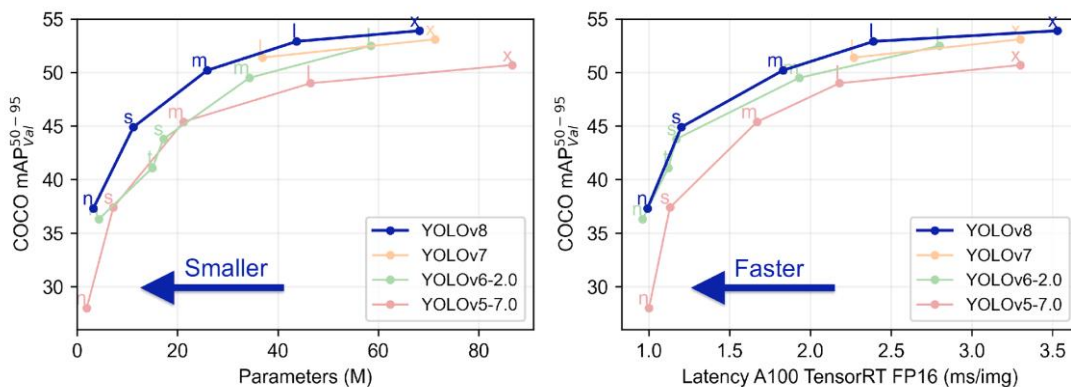


Figure 3.57 YOLOv8 COCO Comparison

RF100 (Roboflow 100) Dataset

Roboflow 100 is a dataset containing 100 separate databases with 224,714 images, which evaluates the model's performance on various domains and applications. Regarding the category "documents"

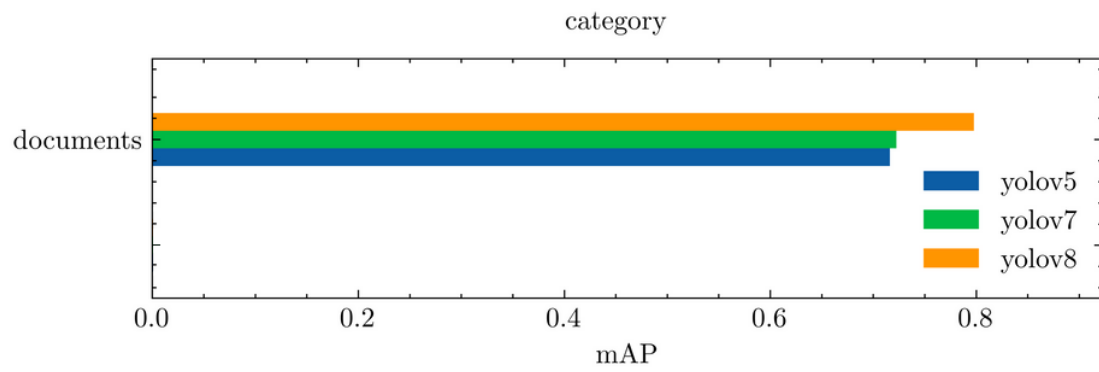


Figure 3.58 YOLOv8 on RF100 Documents Dataset

4

Methodology

At the following chapter we will describe the methods and steps which should be used in order to perform the object detection on the shafting arrangement plan drawings as well as the dimension recognition as well as their combination in order to extract the digital twin of the shafting arrangement.

4.1 Shafting Arrangement Modeling Process

The Shafting Arrangement Plan is one of the most crucial ship design drawings necessary to understand the propulsion system of the ship as well as the components necessary to transfer the rotational motion from the vessel's Main Engine to the propeller in order to produce the required thrust to move the vessel.

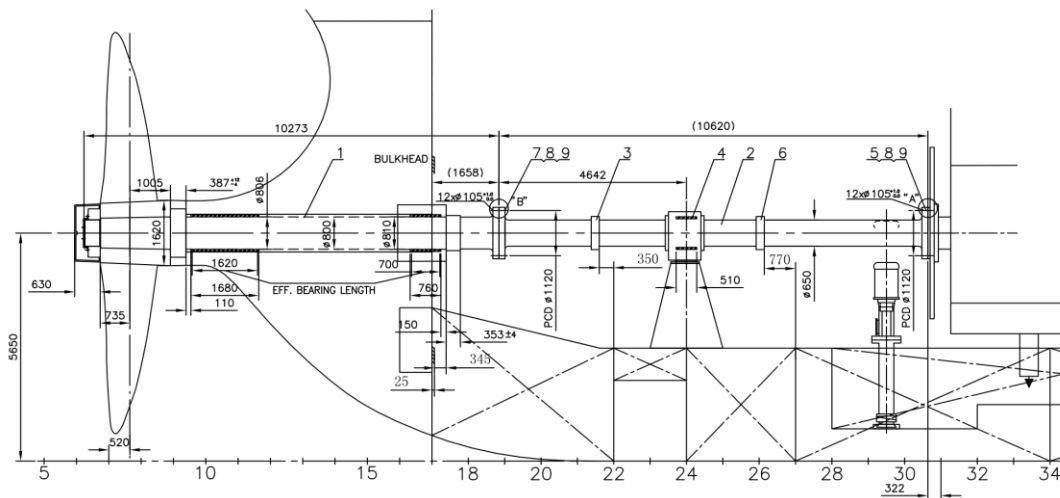


Figure 4.1 A Shaft Arrangement Plan

The shaft arrangement is comprised of the following components:

- A) Propeller Shaft, whose end is linked with the propeller via the propeller's hub and secured via the propeller's aft nut.
- B) Intermediate Shaft (or Shafts) which connect the Propeller Shaft with the Crankshaft of the Main Engine (with or without the presence of a reduction gear whether the engine is two-stroke or four-stroke)

- C) Shaft Bearings, which bear the static and dynamic loads of the shaft arrangement. The Intermediate Shaft is supported via Intermediate Shaft Bearings (Line Shaft Bearings) and the Propeller Shaft is supported via 2 usually Stern Tube bearings an Aft and a Fore.

Modeling the Shaft Arrangement

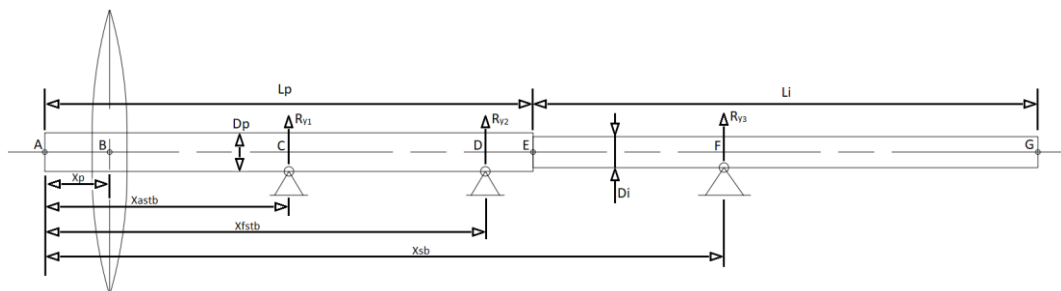
In order to generate a digital model of the shaft arrangement which would be ready for further calculations such as the Shaft Alignment we will represent the shaft arrangement as an FBD (Free Body Diagram) where the shaft will be divided into 2 segments:

- A) Propeller Shaft with diameter D_1 and length L_1
- B) Intermediate Shaft with diameter D_2 and length L_2

For those segments we have to know:

- 1) The location of the center of the Bearings: Aft Stern Tube Bearing, Fwd Stern Tube and the Intermediate Shaft Bearing. We may model the bearings as pinned supports at which the reaction forces may be acted.
- 2) The location of the center of the Propeller: Where the weight of the propeller will be acted upon the shaft.

A representation of the model of the shaft arrangement would be the following:



An approach to identify the positions mentioned above is to identify and detect the following necessary objects:

- 1) Propeller or/and Propeller Hub in case the drawing does not depict the propeller and find it's center.
- 2) Aft End of the Propeller Shaft, which is the Propeller Nut. It is necessary to detect this entity because its aft end indicates the start of the propeller shaft.
- 3) Aft Stern Tube Bearing and the Fwd Stern Tube Bearing (in some cases there's only the Aft Stern Tube Bearing) and find their center.
- 4) Flange Assembly between the intermediate and the propeller shaft, by finding its center we identify the end of the propeller shaft and the start of the intermediate shaft.
- 5) Shaft Bearing (Line Bearing) and detect its center as a support.

6) Main Engine Flange and its center, which indicates the end of the intermediate shaft and the start of the crankshaft.

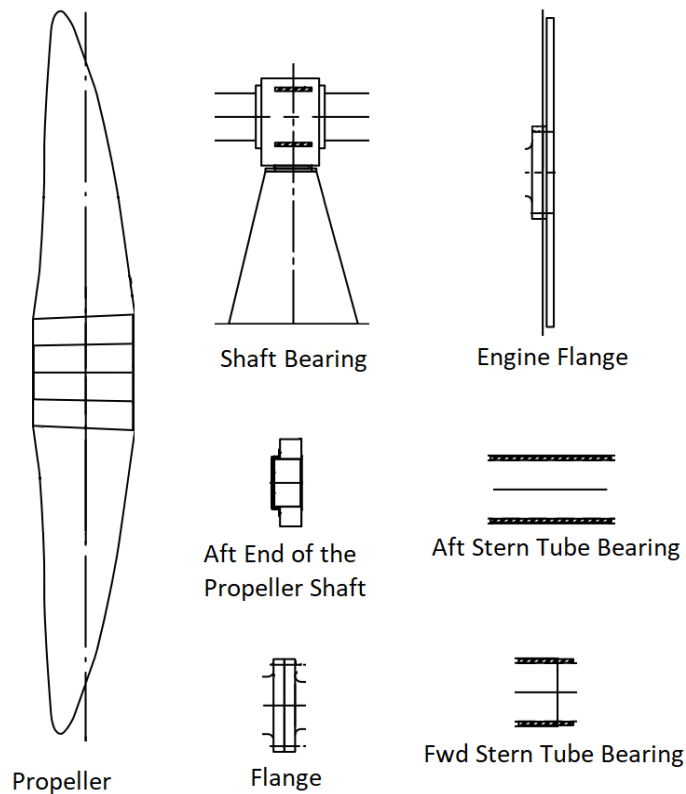


Figure 4.2 Objects required to be detected

By detecting the aforementioned objects inside the image, we may have achieved to identify the distances to the segments A till G, although the distance calculated will be expressed in units of image pixels and not in real units (mm). For this reason, we need to detect the dimensions inside the drawing as well. The dimensions we are required are expressed in the following manner:



Figure 4.3 Representation of Dimensions

When a bounding box surrounds the dimension entity, we can correlate its sides location with the object which is underneath it since both the dimension and the mechanical object have the same starting and ending point. Additionally, we need to develop an OCR (Optical

Character Recognition) which will allow us to convert the number in the dimension to a real integer dimension. One solution would be to train an OCR model from scratch, but a second solution is to use an already pre-trained model. We may use an already pre-trained model for this which will be highly accurate. Thus, it is also safe to derive an analogy between pixels and real dimensions by calculating a weighted average of the ratio mm/px which will exclude any dimension deviates from this average, as a false reading. This way we will have a way to express the x-dimension units from pixels to mm and be able to model accurately.

Finally, we need to define the *Diameters* of the propeller shaft and the intermediate shaft. A solution to this is to train a third model which detects the propeller shaft and the intermediate shaft objects, will rotate them clockwise and pass them through an OCR. In case there are multiple diameters in one of the shafts such as the propeller shaft, we will model it using the average of the detected dimensions.

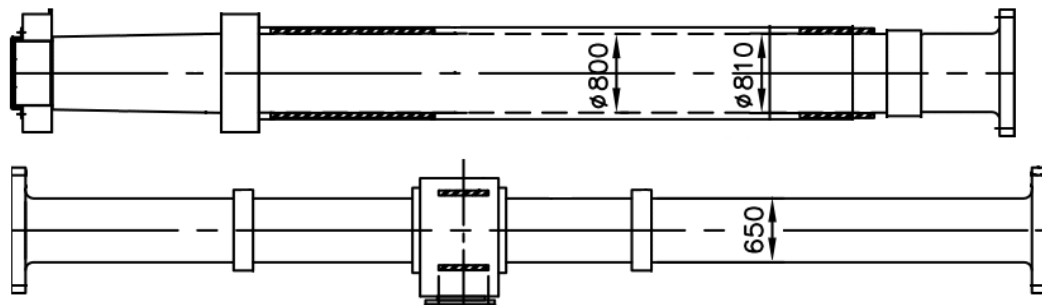


Figure 4.4 Propeller Shaft and Intermediate Shaft

4.2 YOLOv8 and Transfer Learning

The Object Detection model we will be using for this current thesis will be YOLOv8 which is a state-of-the-art object detection model. This thesis will be one of the few worldwide which will have implemented YOLOv8 for their object detection model since YOLOv8 is still under constant development. The number of images which are required to train an object detection model vary and is dependent to multiple factors such as the level of accuracy required, the variability of the dataset as well as the number of classes we want to detect. In our Object Detection case for Shaft Arrangements for 7 classes a general rule of thumb would be to have at least 2000 labeled images per class. Fortunately, we are not required to train YOLOv8 by scratch since YOLOv8 has already been on large-scale image datasets such COCO and RF100 as we previously mentioned. Therefore, in order to take advantage of this starting point we may only need fewer images in order to train YOLOv8 for the specific task of detecting

mechanical objects at shaft arrangement plans. This can be achieved using Transfer Learning. At Transfer Learning we begin with our pre-trained YOLOv8 model on the COCO and RF100 datasets and we fine-tune it for our particular task. The concept of transfer learning lies on the fact that the lower layers of the neural network which are responsible for capturing the low-level features of the images (edges and corners) are generally applicable to a variety of tasks. Thus, by leveraging the flexibility of the YOLOv8's edge and corner detection we are only required to train the higher layers of the DNN. With Transfer Learning we are able to reduce significantly the images required for training with a minimum recommendation of 70 images per class.

4.3 Object Detection Validation and Rules

Assuming our object detection model identifies a number of objects in a given shafting arrangement plan. A Neural Network might be accurately predicting objects, but it still cannot think as an engineer. In order to mitigate the cases of errors which may arise in case of false positives or false detections we are required to add validation rules which will apply to our given objects. The ones we will implement will be the following:

- A) The Shaft Arrangement has a centerline which is continuous from the propeller shaft to the intermediate shaft. All bounding boxes of the objects being detected must pass through this centerline, otherwise they don't belong in the shaft arrangement. This rule stems from the fact that sometimes, the plan may contain appendices with certain parts of the arrangement which might totally ruin the modeling and usually are of different scaling and sizes. Therefore, those objects must be excluded.
- B) The Propeller Center is the center of the acting weight of the propeller. Although in case the Propeller entity does not exist, treat the center of the weight as the center of the Propeller Hub.
- C) The Aft end of the propeller shaft is approximately distanced at 24.3% of the Propeller Hub. According to a derived statistic from our database of vessels we were able to conclude that there is a correlation between the length of the propeller hub and the aft end of the propeller shaft which is:

$$X_{Aft}[\%] = -2.2130 \cdot 10^4 \cdot L_{hub} + 0.5536$$

Where $X_{Aft}[\%]$ is considered the length of the Aft End of the Propeller Shaft as a percentage of the Propeller Hub, and its equal to the distance of the Propeller's Shaft Aft end from the aft end of the Propeller Hub. The graph below shows the cluster of the data being collected from the given dataset (100 vessels)

It is reasonable to assume that the aft end of the propeller shaft does not usually exceed the cluster shown below, and in case the object detection model did not calculate it, we can approximate it via the equation above.

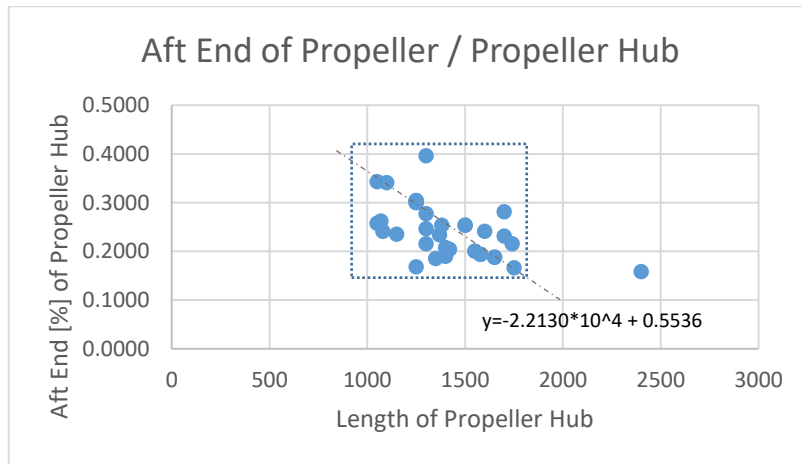


Figure 4.5 Correlation between Aft End and Propeller Hub

- D) There has to be an Aft Stern Tube Bearing Detected and we cannot detect more than one Aft Stern Tubes. In case the Object Detection model detects the Fwd Stern Tube Bearing as an Aft Stern Tube Bearing, the one who's the most aft between them is the Aft Stern Tube Bearing and the other should be the Fwd Stern Tube Bearing.
- E) According to NTUA Lab of Marine Engineering, there is a validation statistic between the length of the Propeller Shaft and the Length of the Intermediate Shaft which approximately estimates that if the dimensionless sum of the propeller and intermediate shaft is equal to 1, then the Propeller Shaft's Length is the 49.4% of the Total Shaft Length and the Intermediate Shaft's Length is the 50.6% of the Total Shaft Length with Standard Deviation of 2.2%

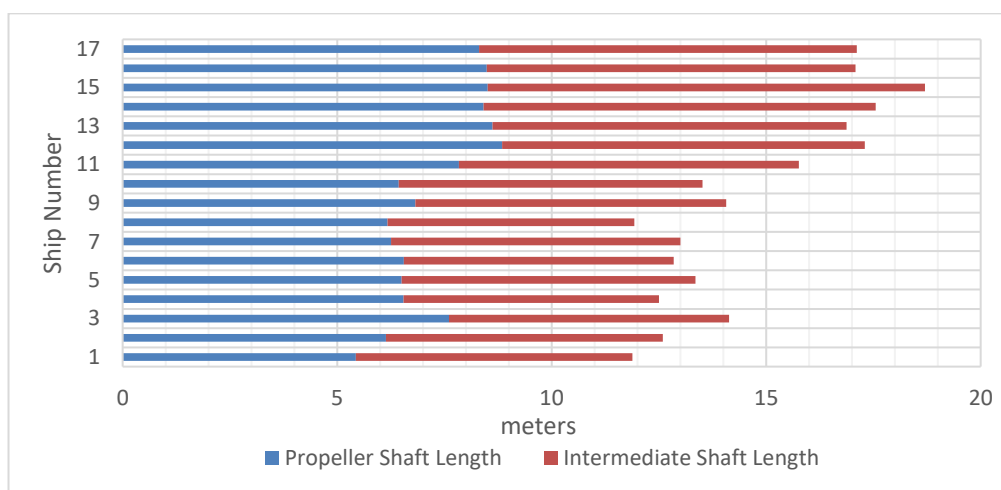


Figure 4.6 Propeller and Intermediate Shaft Correlation

4.4 Process Diagram for Shaft Modeling

The process diagram for the Shaft Modeling Algorithm would be the following:

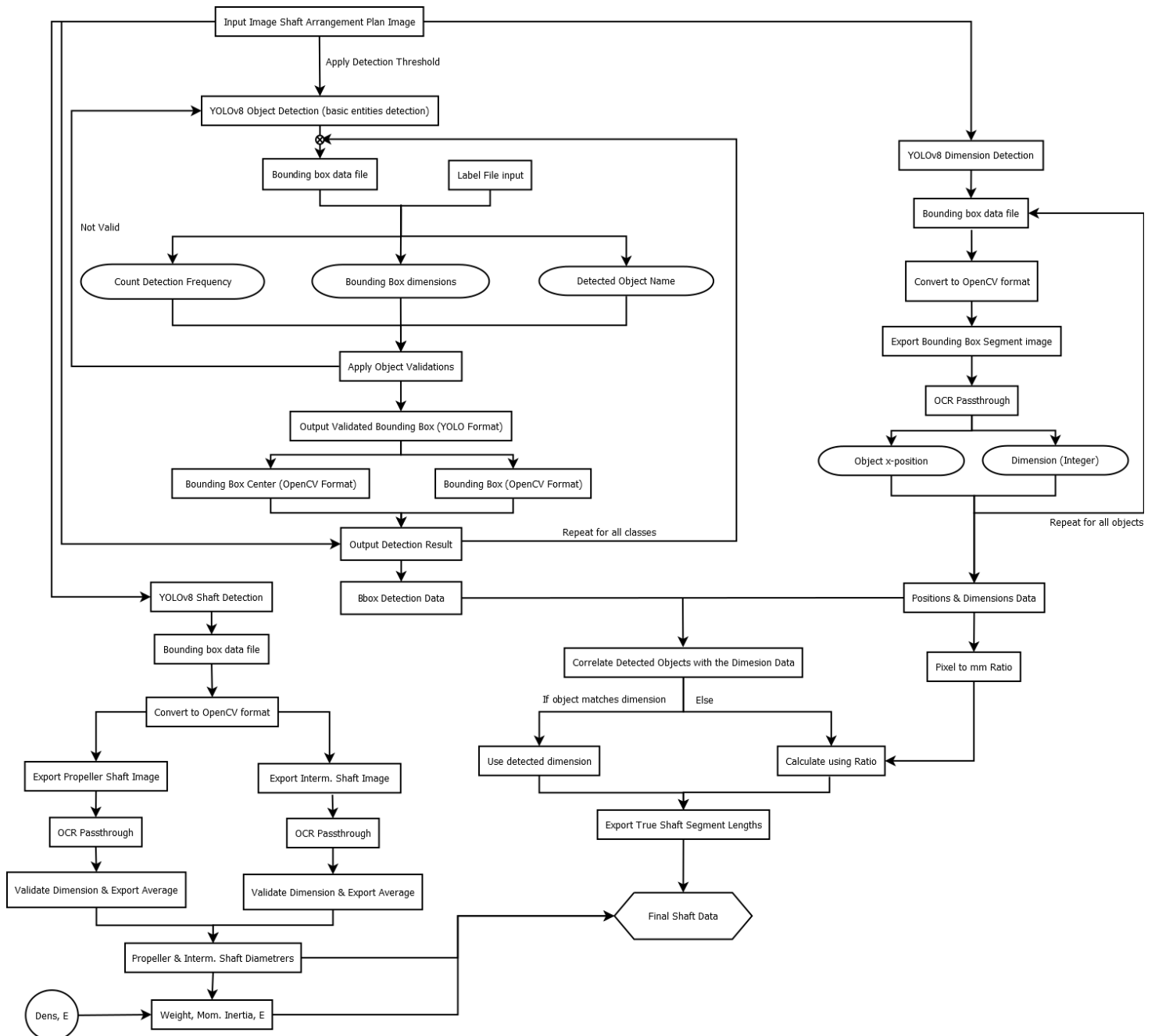


Figure 4.7 Shaft Modelling Algorithm Data Flow Diagram

The modeling in the following algorithm involves the use of 3 YOLO Object Detection Models, one for detecting the primary mechanical entities, one for detecting the shafts and one for

detecting the dimensions, therefore it is important to train the algorithm for 3 separate models. The process of the Shaft Modeling Algorithm is described as follows:

An Image of a Shaft Arrangement Plan will be used as an input, and this will be passed from 3 object detection models:

- A) Key Entities Detection Model: (Aft End of Propeller, ASTB, FSTB, Propeller, Propeller Hub, Shaft Flange, Shaft Bearing, Engine Shaft Flange)
- B) Shaft Detection Model: (Propeller Shaft, Intermediate Shaft)
- C) Dimension Detection Model: (Horizontal Dimensions)

The Key Entities Detection Model is responsible for detecting the aforementioned objects and save their bounding box data in a YOLO Format file. Then the program reads this file and with correlation to the class names file we associate which object is detected and where. We convert those from YOLO Format to OpenCV Format and we pass the result to a validation step which employs the aforementioned validations with the help of a counting class. Then from the validated output for each object we save the bounding box coordinates and the coordinates of its center (in pixel units). Thus, we know the distances from each segment that models the shafting arrangement but in pixel units.

In order to convert those dimensions that we found in real units, we employ the use of the Dimension Detection Model. The Dimension Detection model detects horizontal dimensions. The bounding box of the horizontal dimension gives us the start point and the end point of the dimension detected as well as its length in pixels. It includes the dimension number which is the real in (mm) length of the object. Thus we can:

- Correlate which object has the same start point and end point as the detected dimension.
- Perform OCR inside the bounding box image which will output the real length dimension and thus we can associate the mechanical entity with its real dimension, and we can also provide our algorithm with an analogy of pixels to mm in the horizontal direction.

Any mechanical entity whose dimension could not be arbitrarily detected; its dimension will be extracted from the final ratio which is being calculated as the median value of all ratios from all dimensions excluding the ones which diverge from the median value as false units not associated with the shaft arrangement.

Having the real lengths of the segments between the key mechanical entities we only miss the important dimension of the Diameter of the segments. The diameters of the propeller shaft can be detected using the Shaft Detection Model. It detects the bounding box of the Propeller Shaft and the Intermediate Shaft, thus we can output the height of each bounding

box as well as the dimension number inside the bounding box of each shaft. This allows us to rotate the image clockwise and perform OCR on the image segment we detected. This way we are able to associate this dimension with the specified object (shaft or intermediate) and thus we have the required information on the Shaft Diameter. In case there are multiple dimensions, we have to calculate an average diameter of them all. And if a shaft misses its diameter inside or the OCR fails to detect we either use the pixel to mm ratio from the other shaft or in the worst-case scenario we approximate it using the pixel to mm ratio from the horizontal dimensions.

Finally using the density of the material as an input alongside with the diameters we have detected, we are able to calculate the weight and the moment of inertia for each segment thus having the complete geometry of the shaft digitized.

In the next chapters we will discuss how we were able to generate the dataset necessary to perform the object detection, the training process, and the algorithms deployment process alongside with the results.

4.5 Optical Character Recognition Techniques

There are many available libraries which are specialized for reading numbers, notably Tesseract which is exceptionally well at handling numbers and letters. For this particular thesis we noted the most accurate results derived from the module "OCR.Space.". OCR.Space is a free OCR Service which scans images, in our case the detected entities image segments, and exports the letters and numbers which have been recognized as character variables. OCR.Space provides a free API with an API key which can be used in our Python programmed application. The API offers a variety of engines which are either specialized for different languages, handwriting and of course numbers. We choose the engine for numbers which is optimized for detecting numbers. Although the accuracy may vary thus our API script should be combined with additional filtering which may convert falsely detected numbers as letters such as "0" and "O" or "1" and "l" and convert those to letters, as well as in the case of diameters, to filter out any special characters "Φ" and only output the numerical part.



Figure 4.8 Performing OCR on Diameter example

5

Dataset Generation

The first step for developing our object detection application is the collection of the required drawings of the shaft arrangement plan and generating the appropriate datasets for the object detection training.

5.1 Dataset Generation

We collect a list of 100 Shaft Arrangement Plan drawings from the NTUA Laboratory of Marine Engineering which are either in PDF format or scanned in PNG format. In case the file is at PDF Format we convert it to PNG and create a library of images of Shaft Arrangement Plans. On the figure below we can see examples from the drawings being used in the training dataset.

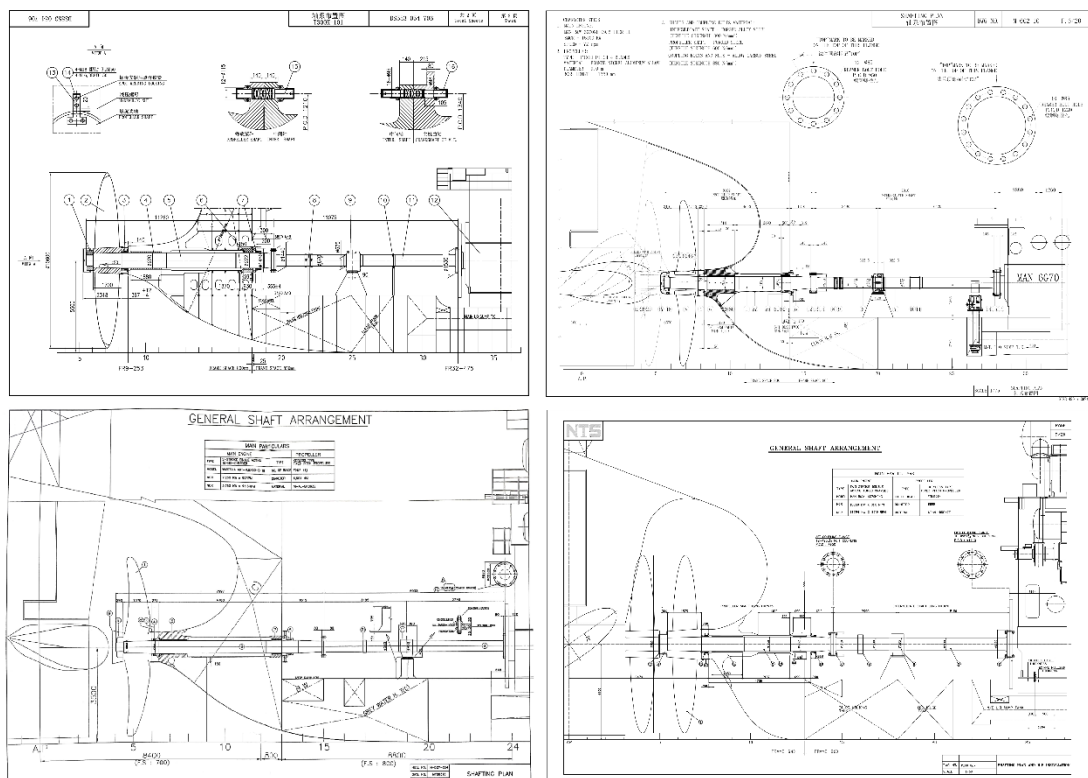


Figure 5.1 Examples from images in the Training Dataset

After the image collection, we ought to label the images for the objects which we want the object detection model to detect. Such requires us to label manually 3 separate datasets one

for the “Key Entities” one for the “Shafts” and one for the “Dimensions”. The labeling can be performed using a software named Labelling. Using Labelling, we set the directory path of the database and the save directory as the same path. Manually we select the bounding box areas of interest and add their respective classes. A labeling annotation process example can be seen in the image below.

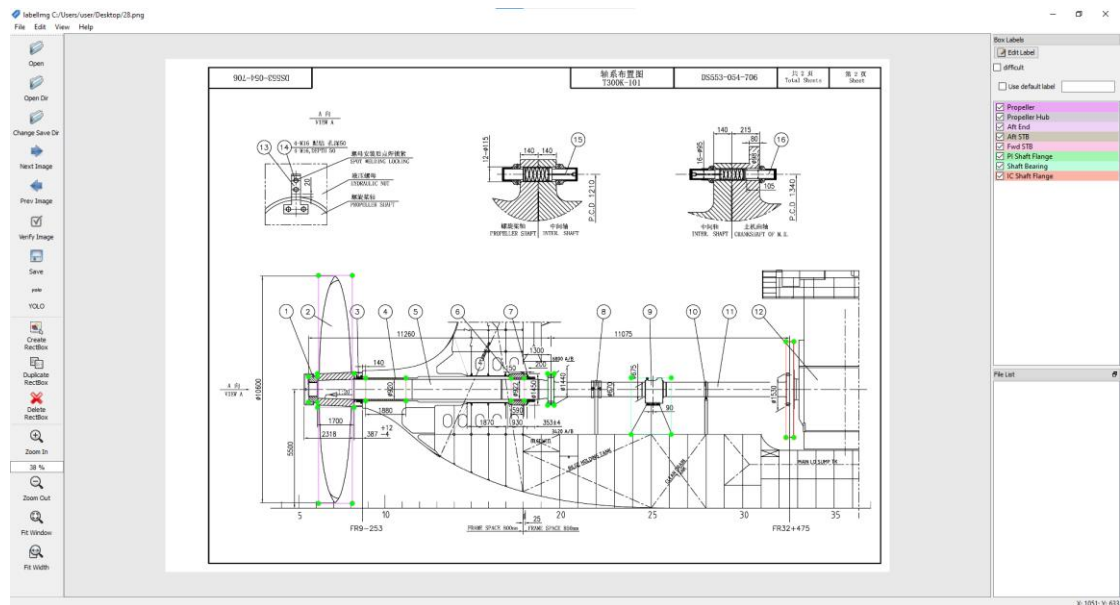


Figure 5.2 Labelling Annotation labeling example

The save output of each image is a text which is presented in the format seen in the figure below:

```

1 0.235133 0.644478 0.050971 0.067209
0 0.234527 0.643836 0.046117 0.444349
2 0.203580 0.643836 0.012743 0.050514
3 0.304460 0.643836 0.056129 0.045377
4 0.486499 0.644478 0.017294 0.061216
5 0.532464 0.644050 0.010316 0.062928
6 0.671117 0.675728 0.055825 0.111729
7 0.862864 0.644050 0.011529 0.187928

```

Figure 5.3 Bounding Box file output

The bounding box output file can be represented in 2 Formats: YOLO Format and PascalVOC Format. The PascalVOC Format is being commonly used for transfer training an SSD and MobileNet Object Detection Model, but since we use YOLOv8 we may use the YOLO Format. The YOLO Format above is consisted of 5 columns the 1st column is an integer representing the ID number of the detected object’s class and the 2nd – 4th columns represent the coordinates of the bounding box of the respective object which is been detected. A notable

example of how YOLO datasets are being structured and its bounding boxes represented can be seen in the figure below:



The origin of the image is being placed on the top-left corner of the image and its dimensions are being normalized as (1,1), hence its position is represented as a percentage of their real pixel dimension:

$$X_{YOLO} = \frac{X_{actual}}{Width} \quad Y_{YOLO} = \frac{Y_{actual}}{Height}$$

When we want to represent a bounding box in YOLO Format, we may represent it using its center and its width and height:

1st Column	2nd Column	3rd Column	4th Column	5th Column
Class ID	X-center	Y-center	Width	Height

The Class ID is the integer number $\{1,2, \dots N\}$ and it is associated with the row of the class file:

```

Propeller
Hub
Aft End of Propeller Shaft
Aft Stern Tube Bearing
Fwd Stern Tube Bearing
PI Shaft Flange
Shaft Bearing
IC Shaft Flange
    
```

Figure 5.4 Class File representation

5.2 Dataset Preprocessing Step

Building a custom dataset can be a really lengthy time-consuming process, hence we may accelerate the time required by using Roboflow, an online computer vision platform which allows users to build computer vision models faster and more robust. Roboflow offers a series of useful modules for data collection, preprocessing techniques and training techniques, which are accompanied with a novel API to import our project datasets directly in our training script. Before we create a new project, we need to create a Roboflow account and right after we create a new project in the main dashboard screen. Next, we upload our dataset file alongside with the class and bounding box files. Roboflow automatically recognizes the YOLO annotation format and generates a Dataset version.

For each of the 3 datasets, Roboflow allows us to perform preprocessing transformations such as modifying the image orientations, resizing, contrast and performing data augmentation. For our current thesis we may employ the following preprocessing steps:

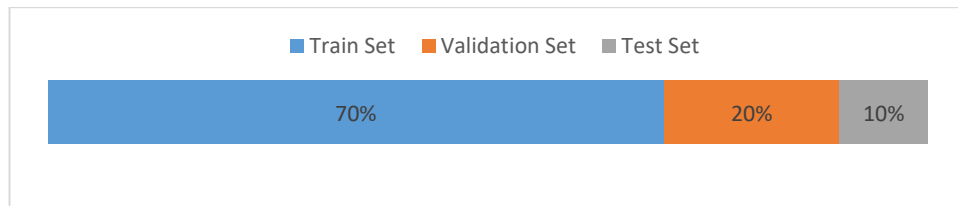
Preprocessing & Augmentation Steps

- A) Grayscale: In case the input drawing (usually an CAD exported file) has lines with different colors, we may convert it grayscale for the reasons we mentioned in the theoretical part of the thesis.
- B) Auto Orient: When an image is being captured it usually contains metadata which dictates the orientation by which the image should be displayed on a native platform. Although this might be an issue when the application is displaying images unaware of the metadata. This preprocessing step discards EXIF rotations and standardizes pixel ordering.
- C) Static Crop: Static Crop allows us to generate cropped images from the original of a random size ranging from 25%-75% for the height and the width which models faults and horizontal misalignments when it comes to a scanned drawing, thus making the model more robust in case of a scanned document.
- D) Tile Operation: Splits the images into a 3x3 grid and generates new images from that grid. It is really helpful with large images such as drawings when we want to detect efficiently small objects such as the Stern Tube Bearings in our case.
- E) Rotations: We may perform minimal rotations of $\pm 1^\circ$ which emulates a rotational misalignment when it comes to a scanned drawing, thus making more robust to real-life scanned scenarios.

F) Noise: We generate Gaussian noise of 1% of the noise in order to prevent overfitting and make it more robust in case of scanned document noise

After performing this augmentation through Roboflow, we are able to generate a dataset consisting of 888 images. We split and shuffle the image data into:

Train Set	Validation Set	Test Set
622	175	91
70%	20%	10%



5.3 Dataset Inspection & Quality Evaluation

For each of the generated datasets we may present an overview inspection of their attributes and evaluate them as per their aspects. Notably our evaluation presents:

Dataset Details: Including, the number of images prior the augmentation, the number of the Annotations found as well as the average number of annotations per image, the average image size and the median image sizes being used.

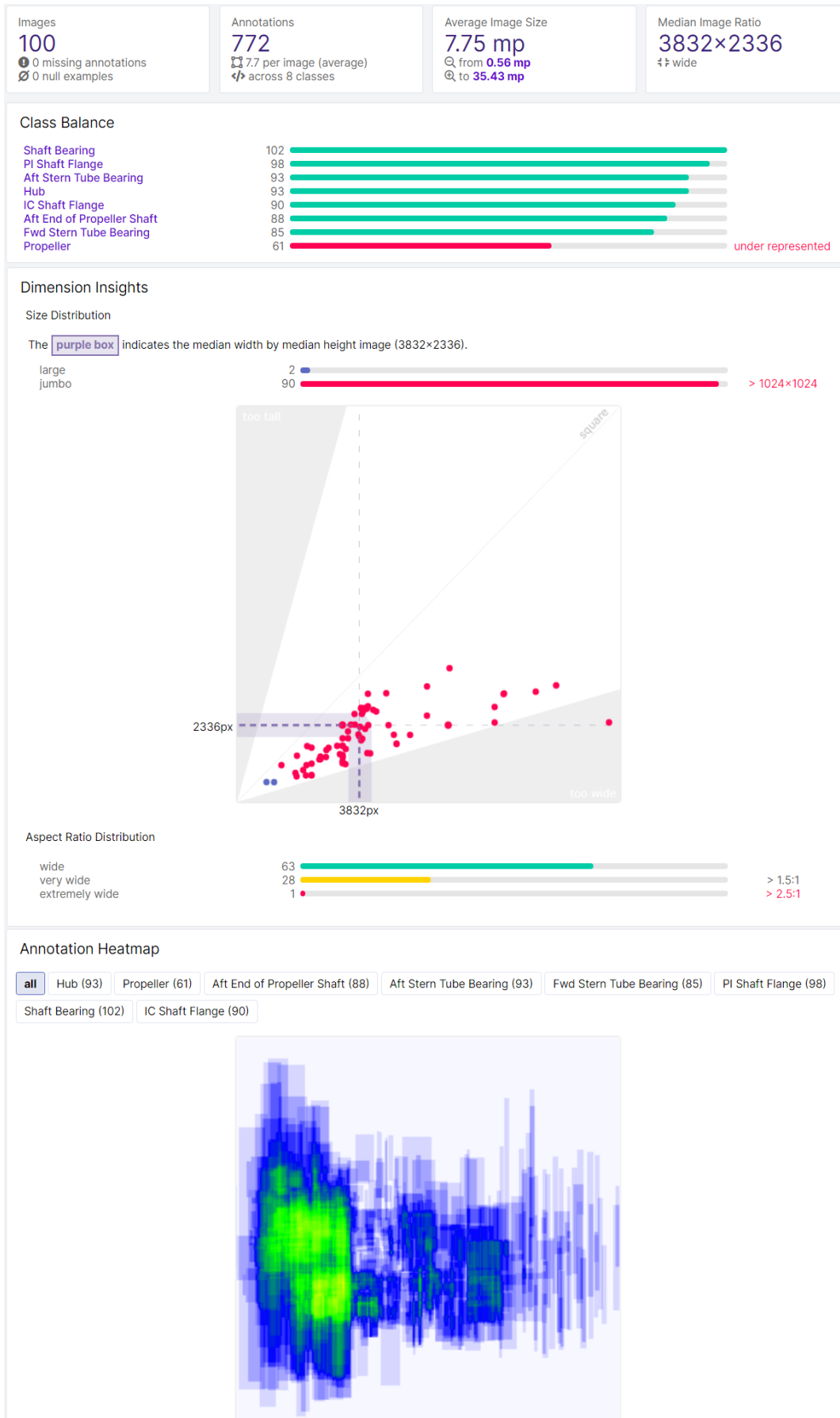
Class Balance: It evaluates the balance between the class annotations, and we can see whether a certain class is over-presented or under-presented. A balanced dataset is usually more robust, unbiased towards the classes, and able to detect all of them accurately.

Dimension Insights: It evaluates the distribution of the sizes of our images in our dataset as well as their aspect ratio distribution. Commonly for the majority of neural networks the input images are usually square even if the aspect ratio of the original images is not. This is due to the fully connected layers which perform on square arrays. Images which are close to square sized can be resized by YOLO maintaining the detail of their features. Although too wide or too narrow images can cause great distortions in the image features. Hence it is important to have a dataset of images of a similar size, ideally not too narrow or wide.

Annotation Heatmap: The annotation heatmap visualizes the area on the image where the objects are most commonly annotated. This is an important value for YOLOv8 as it understands that an object found outside the heatmap area with the most probability is likely to be a false positive annotation and should not be regarding during the models training.

The reports can be found for each dataset below.

Key Entities Detection Dataset Report



Shaft Detection Dataset Report

Images 100 0 missing annotations 0 null examples	Annotations 209 2.1 per image (average) across 2 classes	Average Image Size 7.75 mp from 0.56 mp to 35.43 mp	Median Image Ratio 3832×2336 wide
---	---	--	---

Class Balance

Intermediate Shaft	100	<div style="width: 100%;"></div>
Propeller Shaft	91	<div style="width: 91%;"></div>

Dimension Insights

Size Distribution

The **purple box** indicates the median width by median height image (3832×2336).

large	2	<div style="width: 2%;"></div>
jumbo	90	<div style="width: 90%;"></div> > 1024×1024

Aspect Ratio Distribution

wide	63	<div style="width: 63%;"></div>
very wide	28	<div style="width: 28%;"></div>
extremely wide	1	<div style="width: 1%;"></div> > 1.5:1 > 2.5:1

Annotation Heatmap

all Propeller Shaft (91) Intermediate Shaft (100)

Dimensions Detection Dataset Report

Images 100 0 missing annotations 0 null examples	Annotations 1,276 12.8 per image (average) across 1 classes	Average Image Size 7.75 mp Q from 0.56 mp Q to 35.43 mp	Median Image Ratio 3832×2336 wide
---	--	--	---

Class Balance

Dimension 1,276

Dimension Insights

Size Distribution

The **purple box** indicates the median width by median height image (3832×2336).

large 2
jumbo 90 > 1024×1024

Aspect Ratio Distribution

wide 63
very wide 28 > 1.5:1
extremely wide 1 > 2.5:1

Annotation Heatmap

all Dimension (1,276)

As we can see from the reports above, we have a total of 100 original images in our dataset with $n_1 = 7.7$ annotations per image (772 in total) for the Key Objects Dataset, $n_2 = 2.1$ annotations per dataset for the Shafts Dataset (209 in total) and $n_3 = 12.8$ for the Dimensions Dataset (1276 in total). All the datasets are on average 7.76 megapixels with a median size of 3832x2236. According to the Dimension Insights such dataset lies into the Wide/Very wide spectrum of size distribution which is in the acceptable spectrum of dataset images. As for the Annotation Heatmap we perceive the following spatial relations between the key entities which are being detected:

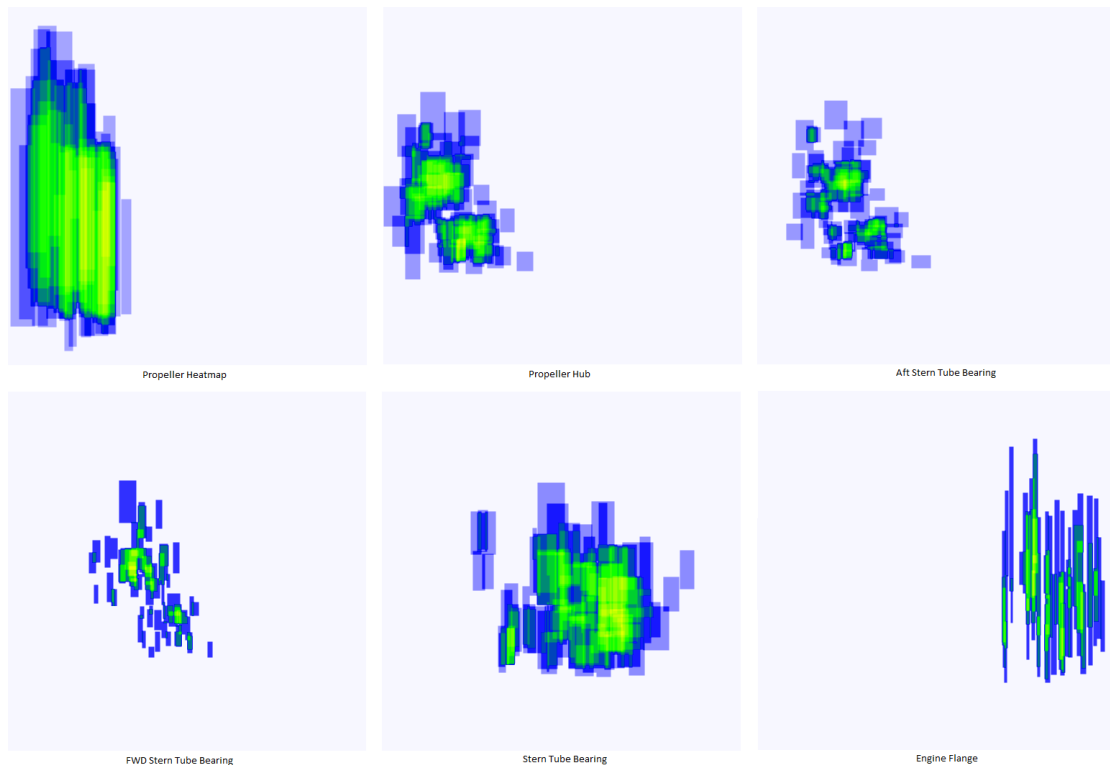


Figure 5.5 Annotation Heatmaps for Key Object

YOLOv8 will be able to understand the positional relationships between the key objects: The Propeller is a tall/narrow object which is located with great probability on the leftmost side of the image, following the Propeller Hub, which has a rectangular heatmap and it is mostly located on the central area of the Propeller's heatmap. Next, the Aft and the Fwd Stern Tube Bearings are visible with a spatial relation of the Aft always being at the left side of the Fwd. The Stern Tube Bearing is mostly centrally located but it's always before the Engine flange which is perceived as a very narrow object (due to the engine's flywheel). Thus, during YOLOv8 model's training in the next chapter, YOLOv8 will be able to use those spatial relations to detect false positive detected objects and exclude them by assigning them a low confidence rate. It is necessary to understand that it is nearly impossible for example to detect a Propeller

near the Stern Tube and away from the Propeller Hub since it's a violation of common engineering logic. This is a significant step into partaking engineering knowledge and intuition of some level to our Object Detection model in order for it to think and act as a Marine Engineer and not plainly like an AI.

Similarly, we may have as an output the heatmap for the Shafts as seen in the figure below:

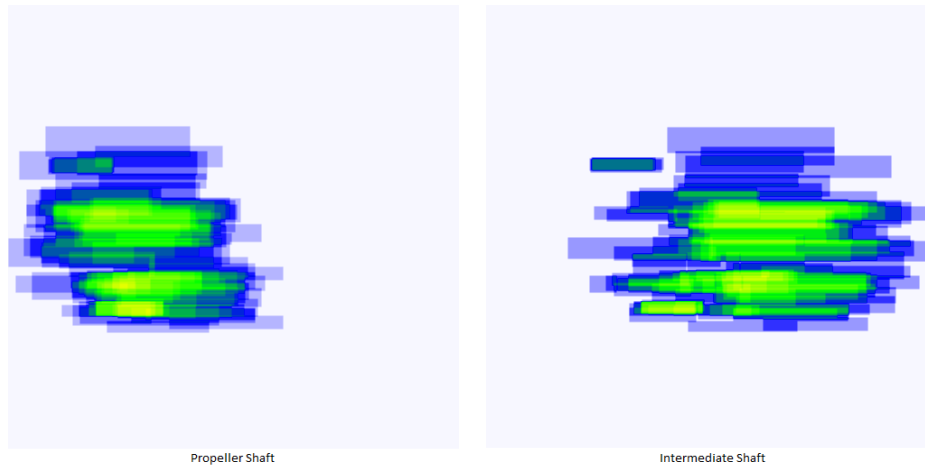


Figure 5.6 Annotation Heatmap of the Shaft Objects

Such heatmap reveals the spatial relation that the Propeller Shaft is to be detected on the left side of the Intermediate Shaft. Notably the aforementioned relationships are standardized since the Shaft Arrangement Plan is always being displayed as a Side View from the Starboard size, hence the propeller will almost always be located on the leftmost side and the engine part on the rightmost side of the arrangement. Lastly, regarding the dimensions, as seen in the heatmap below they are identified as wide objects which are mainly located on top of the shaft arrangement, and on the bottom especially below the propeller hub:

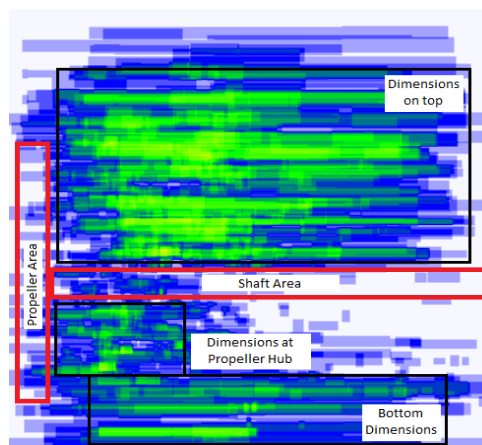


Figure 5.7 Heatmap of Dimensions

Our datasets are now ready to be used for our training process of the three object detection models which will be discussed on the next chapter.

6

Training & Validation

After creating our training datasets, we will use them to perform transfer learning on the pretrained YOLOv8 models. After choosing the appropriate YOLOv8 model from those available we train them using the YOLOv8 library from GitHub, we will validate our dataset, perform inference with the test dataset and lastly, we will export the models in PyTorch (.pt) format, ready to be used in our Shaft Arrangement Modeling software.

The steps to train the YOLOv8 Object Detection Models can be summarized into the following:

- 1) Step 1: Import the necessary Libraries (Ultralytics YOLO Library, PIL, OpenCV)
- 2) Step 2: Import the preprocessed ready-to-use dataset using the Roboflow API
- 3) Step 3: Perform the YOLOv8 training with the optimal parameters which are already predefined by the YOLOv8 such as:

- a. Loss Function: Weighted YOLOv8 IoU Binary cross-entropy loss function

$$L_{YOLOv8} = w_1(L_{IoU}, L_{BCE}) \cdot L_{IoU} + w_2(L_{IoU}, L_{BCE}) \cdot L_{BCE}$$

- b. Optimizer: Stochastic Gradient Descent (with momentum)

$$w := w - \eta \nabla J_i(w) - a \Delta w$$

- c. Learning Rate: Variable $\eta = 10^{-3}$ for the first 60 epochs and then $\eta' = \eta/10$ till the 160 epochs which is again $\eta'' = \eta'/10$ with Momentum Factor: $a = 0.9$
- d. Number of Epochs: $epochs = 200$

- 4) Step 4: Perform training and evaluate the performance metrics in each iteration:
 - a. Box Loss: Represents how well the model fits the true bounding box
 - b. Class Loss: Represents how well the model predicts the object classes.
 - c. Dual Focal Loss: Alleviates the class imbalance of the dataset (DFL)
 - d. $mAP_{0.50}$ & $mAP_{0.50-0.95}$: Mean Average Precision which is already defined.
- 5) Step 5: Output the plots of the metrics and evaluate the model's performance as well as performing the validation for all the classes.
- 6) Step 6: Perform Inference with the trained YOLO model on our test dataset.

7) Step 7: Judge whether the performance of the trained YOLO model is acceptable. If it is acceptable then we export the model, else we change the above parameters or we have to improve our dataset.

6.1 Choosing YOLOv8 Model Variation

According to Ultralytics (GitHub), YOLOv8 has 5 model variations which vary with an increase in performance, accuracy and parameters but with a substantial decrease in the processing speed. Choosing a balance between speed and accuracy is reasonable when it comes to real-time applications where the slightest increase in processing speed (ms) can reduce the FPS of the detection, but in our case since it is a single image detection application, we are free to choose the one which offers the best performance regardless the milliseconds increase which is barely noticeable.

YOLOv8 Models				
Model	Model Size	mAP ^(0.50-0.95)	Speed (CPU)	Parameters
YOLOv8n	Tiny	37.3	80.4 ms	3.2 x 10 ⁶
YOLOv8s	Small	44.9	128.4 ms	11.2 x 10 ⁶
YOLOv8m	Medium	50.2	234.7 ms	25.9x 10 ⁶
YOLOv8l	Large	52.9	375.2 ms	43.7 x 10 ⁶
YOLOv8x	Extra Large	53.9	479.1 ms	68.2 x 10 ⁶

Hence, we choose the YOLOv8x model the largest and most accurate of all which boasts a 53.9 accuracy on the COCO Dataset and has 68.2 million parameters which need to be fine-tuned to our dataset.

6.2 System Specifications for Training

For the YOLO Object Detection application, we will be using the Google Collab resources which come with an online interactable Jupyter Notebook and cloud computing service (Python 3 Google Computing Engine) with the following specifications:

- A) GPU: Tesla T4 – 16GB GDDR – CUDA Enabled (2560 CUDA Cores) & FP32: 8.1 TFLOPS
- B) CUDA: NVIDIA-SMI: 525.85.12 and CUDA Version 12.0
- C) ENV: Linux with Python 3.8.10 - PyTorch 1.13.1 and Ultralytics YOLOv8.0.11

It is important to perform the command 'nvidia-smi' prior to the training to ensure that the training is GPU Accelerated which significantly increases the training speed.

6.3 YOLOv8 CLI and Python SDK

YOLOv8 offers an amazing CLI for Linux accompanied with an excellent SDK for Python. As long as the necessary libraries (according to GitHub requirements) are installed, YOLOv8 can be used for detection and classification tasks with a training or prediction mode using the CLI as seen below:

```
yolo task=detect mode=train model=yolov8n.yaml args...
      classify  predict  yolov8n-cls.yaml args...
      val      yolov8n-seg.yaml args...
      export   yolov8n.pt  format=onnx args...
```

Figure 6.1 YOLOv8 CLI Example

Our task is set to detect, and we can simply choose between the train mode for training and prediction mode for the inference. With the model option we choose the pretrained initial model which will be used for transfer learning if we choose the training mode, or the final model when we perform inference.

YOLOv8 is still on an experimental stage and new updates continue to be publicly released on GitHub which add new functionality and improve the library's performance and fix various bugs.

6.4 Roboflow API and Importing Training Dataset

We import our training datasets using the Roboflow API. Using Roboflow we have already annotated our Dataset, performed several preprocessing and augmentation steps and we already split the images into a Training – Test – Validation set. Roboflow enables us to directly import the dataset into the training notebook of Google Collab by simply installing the Roboflow library. We use our API key to sign into our account and we import the project we work on as well as the dataset with the appropriate variables which are generated at the dataset export tool of Roboflow. A sample script can be seen below:

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX")
project = rf.workspace("test-cyztp").project("dimensions")
dataset = project.version(2).download("yolov8")
```

Figure 6.2 Roboflow API Importing Dataset to Google Collab

6.5 YOLOv8 Training Hyperparameters

We begin our YOLOv8 training using the simple CLI command as seen in the figure below:

```
%cd {HOME}
!yolo task=detect mode=train model=yolo_naoum.pt data={dataset.location}/data.yaml epochs=200
```

Figure 6.3 YOLOv8 CLI Training Command

YOLOv8 already has predefined its architecture, its pretrained weights as well as its optimal parameters which according to the YOLOv8 Ultralytics development team provide a robust and accurate training:

Loss Function: YOLOv8 offers a weighted IoU Binary cross-entropy loss function, a linear combination of the IoU Loss Function and the Binary Cross-Entropy Function:

$$L_{IoU} = 1 - \frac{|B \cap B^{gt}|}{|B \cup B^{gt}|} + \frac{|C - B \cap B^{gt}|}{|C|}$$

With C the smallest box covering the predicted and ground truth bound boxes and the BCE Loss Function:

$$L_{BCE} = H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log \log (p(y_i)) + (1 - y_i) \cdot \log \log (1 - p(y_i))$$

Thus, the YOLOv8 Loss can be modeled as:

$$L_{YOLOv8} = w_1(L_{IoU}, L_{BCE}) \cdot L_{IoU} + w_2(L_{IoU}, L_{BCE}) \cdot L_{BCE}$$

Optimizer: YOLOv8 uses its own the Stochastic Gradient Descent (SGD) with momentum and adaptable learning rate as an Optimizer:

$$w := w - \eta \nabla J_i(w) - a \Delta w$$

SGD is considered one of the most well performing optimizers for object detection since it appears to converge quickly especially when we have high-dimensional feature spaces. YOLOv8's SGD is able to handle image data really well since by randomly selecting mini batches of pixel data from each iteration, it can avoid getting stuck at local minima and provide a good training result. YOLOv8's SGD has momentum as well which remembers the last update of the weight Δw and uses it in a linear combination with the SGD loss to determine the next update.

Learning Rate: YOLOv8 has an adaptable learning rate which varies depending on the epoch we are: The first 60 use $\eta = 10^{-3}$ and between 60-160 use $\eta = 10^{-4}$ and above 160 $\eta = 10^{-5}$ Lastly the momentum factor is $a = 0.9$ according to the Optimizer above.

Epochs: An epoch refers to a complete iteration through the entire image dataset. During each epoch, the YOLO model goes through all the bounding boxes in the dataset and updates its model parameters based on the above loss function and the optimization algorithm. The higher the number of epochs the more times the YOLO model will pass through the entire dataset, although the choice of the epoch number has to be picked carefully because a low epoch number can result in underfitting with the model not capturing the image features or with a high epoch number the model can overfit and become biased to the training data while performing poorly on new data. Hence, choosing an epoch number is often a matter of trial and error. For our current model we will chose $epochs = 200$. The batch number is set at 16. By defining the above training parameters, we are able to commence the training on the 3 datasets on the YOLOv8x model consisting of 365 Layers and 68,160,312 parameters.

```

Ultralytics YOLOv8.0.11 Python-3.8.10 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
yolo/engine/trainer: task=detect, mode=train, model=yolo_naoum.pt, data=/content/datasets/Drawings-3/data.yaml, epochs=200, patience=50, batch=16
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 91.4MB/s]

  from n  params module                                arguments
  0      -1 1    2320 ultralytics.nn.modules.Conv                    [3, 80, 3, 2]
  1      -1 1   115520 ultralytics.nn.modules.Conv                    [80, 160, 3, 2]
  2      -1 3   436800 ultralytics.nn.modules.C2f                    [160, 160, 3, True]
  3      -1 1   461440 ultralytics.nn.modules.Conv                    [160, 320, 3, 2]
  4      -1 6   3281920 ultralytics.nn.modules.C2f                    [320, 320, 6, True]
  5      -1 1   1844480 ultralytics.nn.modules.Conv                    [320, 640, 3, 2]
  6      -1 6   13117440 ultralytics.nn.modules.C2f                    [640, 640, 6, True]
  7      -1 1   3687680 ultralytics.nn.modules.Conv                    [640, 640, 3, 2]
  8      -1 3   6969600 ultralytics.nn.modules.C2f                    [640, 640, 3, True]
  9      -1 1   1025920 ultralytics.nn.modules.SPPF                   [640, 640, 5]
 10     -1 1     0 torch.nn.modules.upsampling.Upsample        [None, 2, 'nearest']
 11     [-1, 6] 1     0 ultralytics.nn.modules.Concat                 [1]
 12     -1 3   7379200 ultralytics.nn.modules.C2f                    [1280, 640, 3]
 13     -1 1     0 torch.nn.modules.upsampling.Upsample        [None, 2, 'nearest']
 14     [-1, 4] 1     0 ultralytics.nn.modules.Concat                 [1]
 15     -1 3   1948800 ultralytics.nn.modules.C2f                    [960, 320, 3]
 16     -1 1   922240 ultralytics.nn.modules.Conv                    [320, 320, 3, 2]
 17     [-1, 12] 1     0 ultralytics.nn.modules.Concat                 [1]
 18     -1 3   7174400 ultralytics.nn.modules.C2f                    [960, 640, 3]
 19     -1 1   3687680 ultralytics.nn.modules.Conv                    [640, 640, 3, 2]
 20     [-1, 9] 1     0 ultralytics.nn.modules.Concat                 [1]
 21     -1 3   7379200 ultralytics.nn.modules.C2f                    [1280, 640, 3]
 22     [15, 18, 21] 1 8725672 ultralytics.nn.modules.Detect                 [8, [320, 640, 640]]

Model summary: 365 layers, 68160312 parameters, 68160296 gradients, 258.2 GFLOPs

Transferred 595/595 items from pretrained weights
Logging results to runs/detect/train
Starting training for 200 epochs...

  Epoch  GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
  1/200   12.9G   0.6831    2.717    0.8275     36         640: 100% 13/13 [00:29<00:00, 2.25s/it]
    Class  Images  Instances  Box(P  R  mAP50  mAP50-95): 100% 1/1 [00:02<00:00, 2.62s/it]
    all      18      141    0.569  0.713  0.703  0.571

  Epoch  GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
  2/200   12.9G   0.7027    0.7028    0.8357     74         640: 100% 13/13 [00:26<00:00, 2.04s/it]
    Class  Images  Instances  Box(P  R  mAP50  mAP50-95): 100% 1/1 [00:01<00:00, 1.34s/it]
    all      18      141    0.906  0.898  0.93  0.755

  Epoch  GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
  3/200   12.9G   0.675    0.4543    0.8159     57         640: 100% 13/13 [00:26<00:00, 2.00s/it]
    Class  Images  Instances  Box(P  R  mAP50  mAP50-95): 100% 1/1 [00:01<00:00, 1.84s/it]
    all      18      141    0.923  0.934  0.952  0.744

  Epoch  GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
  4/200   12.9G   0.7052    0.4601    0.8132     53         640: 100% 13/13 [00:27<00:00, 2.08s/it]
    Class  Images  Instances  Box(P  R  mAP50  mAP50-95): 100% 1/1 [00:01<00:00, 1.31s/it]
    all      18      141    0.944  0.932  0.964  0.749

```

Figure 6.4 YOLOv8 Training Console Output Sample

The figure above shows the output of the Jupyter Notebook when the training process is initiated. We can clearly see the architecture of the YOLOv8x network, and we can see for each epoch: how much GPU Memory does it use, the Box Loss, the Class Loss, the Dual Focal Loss and the Mean Average Precision.

6.6 YOLOv8 Training Evaluation Metrics

After we perform the training process for each Dataset we output and evaluate the results from the model training:

6.6.1 Key Entities Detection Model Evaluation

The model reached its best version when exported at $epoch = 179$ with the following performance metrics:

Object Detection Model - Key Entities - Metrics					
Epoch	Box Loss	Class Loss	Dual Focal Loss	mAP _{0.50}	mAP _{0.50-0.95}
179	0.3614	0.2188	0.7767	0.976	0.773

Both the Box, Class and DF Loss are minimal (below 1) which reveal the model to be very accurate in terms of prediction the bounding box and the object class correctly even when the classes are under-represented. The Mean Average Precision for an IoU threshold of 0.5 is **mAP_{0.50} = 97.6**, and the mAP for all thresholds of 0.5 – 0.95 is **mAP_{0.50-0.95} = 77.3** which are exceptionally high, hence our model is very precise and accurate on its predictions.

After the model's training we may output the performance graphs of the training process:

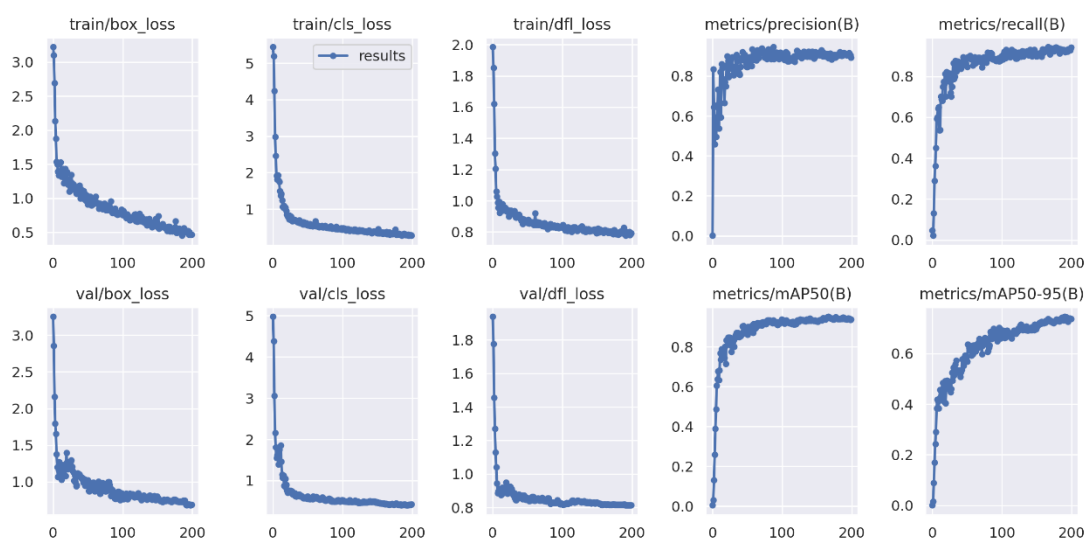


Figure 6.5 YOLOv8 Key Entities Training Performance

According to the figure above, the box loss, class loss and train loss is being minimized safely without overshoot for both the train and the validation sets, meaning that the model is well robust against overfitting and the its performance on new data is acceptable. Both precision and recall increase. Precision appears to be constant at the 80 first epochs and forth, and the

Recall appears to be increasing throughout the epochs. The $mAP_{0.50}$ shows minimal increase after the 100th epoch and the mean average precision for all thresholds of 0.5 – 0.95 appear to increase adequately. We estimate that *further training would not benefit* the model and would only lead to overfitting, hence *no change in the training hyperparameters* is required.

A representation of the Precision and the Recall of the model can be visualized with the Precision-Recall Curve:

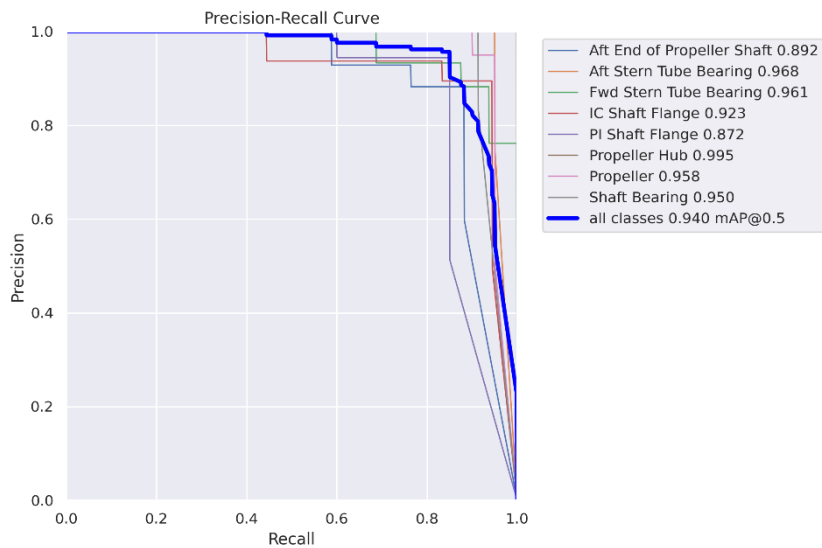


Figure 6.6 Precision - Recall Curve

The model appears to have a high precision even when the recall is high which signifies that a model outputs a high ratio of true accurately labeled classes while keeping a low ratio of falsely labeled classes. A better representation of their relationship can be seen in the F1 Score/Confidence curve below where F1 Score is the harmonic mean of Precision and Recall:

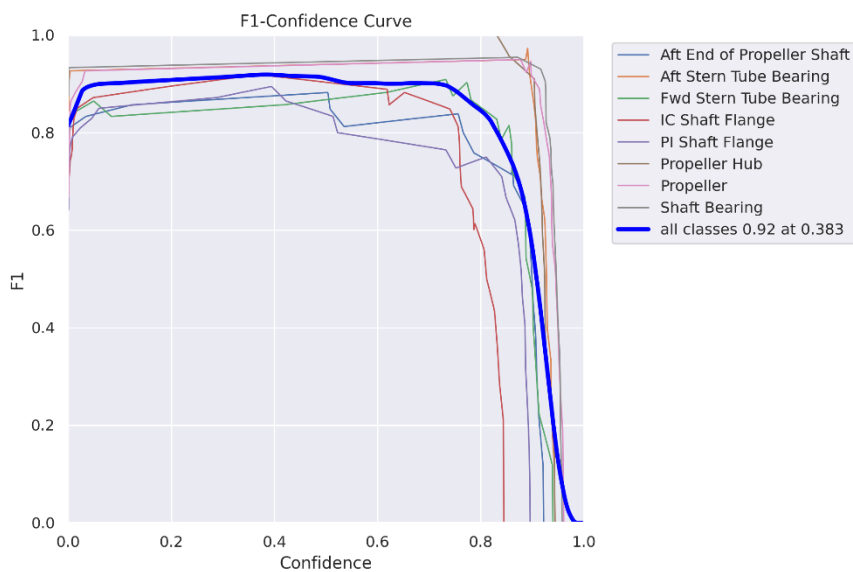


Figure 6.7 F1 - Confidence Curve

The curve appears to be flat at a high average F1 Score of 0.90 – 0.92 for a wide range of confidence values up to 0.8 which means that the model remains precise with minimal false detections, even when the model is conservative with higher confidence rate.

Lastly, we may output the confusion matrix which shows the behavior of our model on how many predicted classes are accurately predicted:

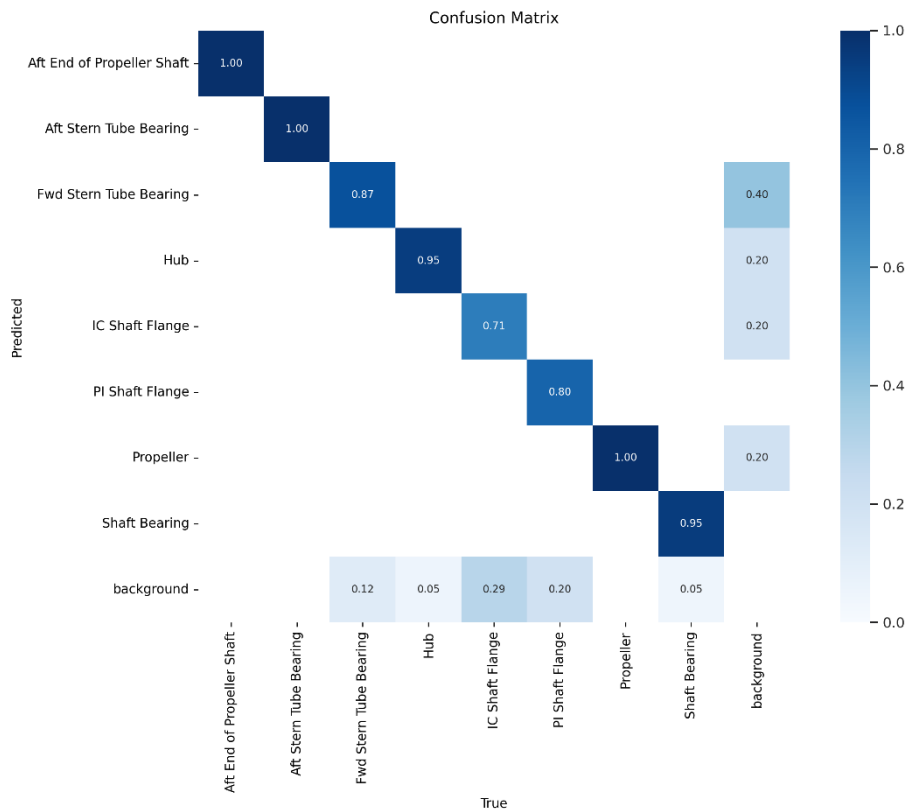


Table 6.1 – Confusion Matrix for the Key Entities Detection Model

According to the Confusion Matrix above, our model seems to be predicting accurately the majority of the classes very accurately with a minimal confusion of some entities with the background lines which is on the spectrum of the acceptable error.

Overall, the Key Entities Detection Model is a very accurate YOLO model which is sufficient for this thesis case and will be able to perform accurately on the vast majority of Shaft Arrangement Plans, detecting all the necessary geometries needed to model the shaft, its constraints, and their positions.

6.6.2 Shafts Detection Model Evaluation

In a similar manner we evaluate the performance of the Shafts Detection Model. The Shaft Detection model has only 2 classes: Propeller Shaft and Intermediate Shaft hence training was easier and faster to be performed.

The model reached its best version when exported at *epoch* = 200 with the following performance metrics:

Object Detection Model - Key Entities - Metrics					
Epoch	Box Loss	Class Loss	Dual Focal Loss	mAP _{0.50}	mAP _{0.50-0.95}
200	0.4659	0.3318	0.8058	0.957	0.832

Both the Box, Class and DF Loss are minimal (below 1) which reveal the model to be very accurate in terms of prediction the bounding box and the object class correctly even when the classes are under-represented. The Mean Average Precision for an IoU threshold of 0.5 is **mAP_{0.50} = 95.7**, and the mAP for all thresholds of 0.5 – 0.95 is **mAP_{0.50-0.95} = 83.2** which are exceptionally high, hence our model is very precise and accurate on its predictions.

After the model’s training we may output the performance graphs of the training process:

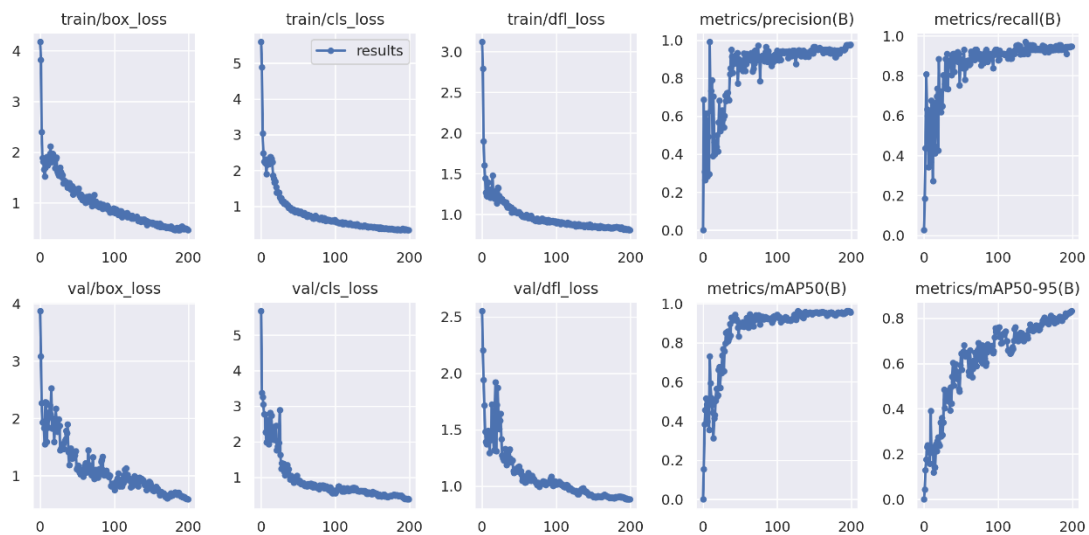


Figure 6.8 YOLOv8 Key Entities Training Performance

According to the figure above, the box loss, class loss and train loss is being minimized safely without overshoot for both the train and the validation sets, meaning that the model is well robust against overfitting and the its performance on new data is acceptable. Both precision and recall increase. Precision appears to be constant at the 80 first epochs and forth, and the Recall appears to be increasing throughout the epochs. The mAP_{0.50} shows minimal increase after the 100th epoch and the mean average precision for all thresholds of 0.5 – 0.95 appear to increase adequately. We estimate that *further training would not benefit* the model and would only lead to overfitting, hence *no change in the training hyperparameters* is required.

A representation of the Precision and the Recall of the model can be visualized with the Precision-Recall Curve:

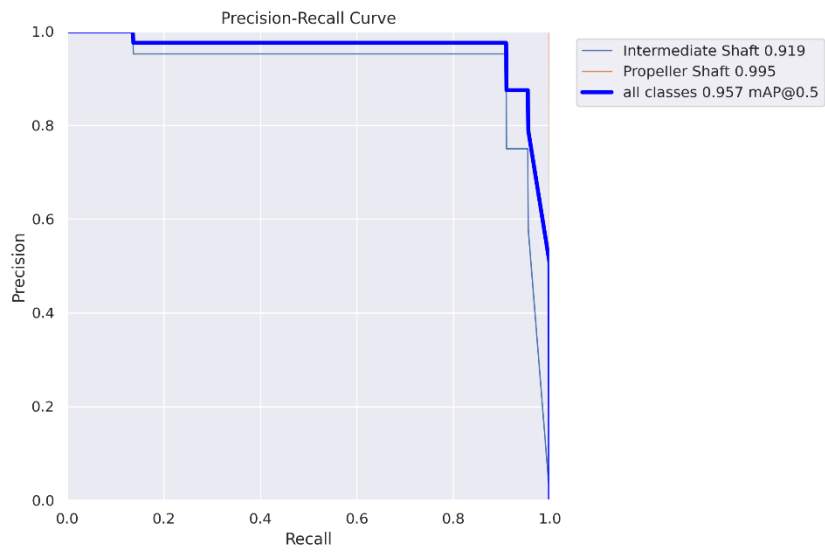


Figure 6.9 Precision - Recall Curve

The model appears to have a high precision even when the recall is high which signifies that a model outputs a high ratio of true accurately labeled classes while keeping a low ratio of falsely labeled classes. A better representation of their relationship can be seen in the F1 Score/Confidence curve below where F1 Score is the harmonic mean of Precision and Recall:

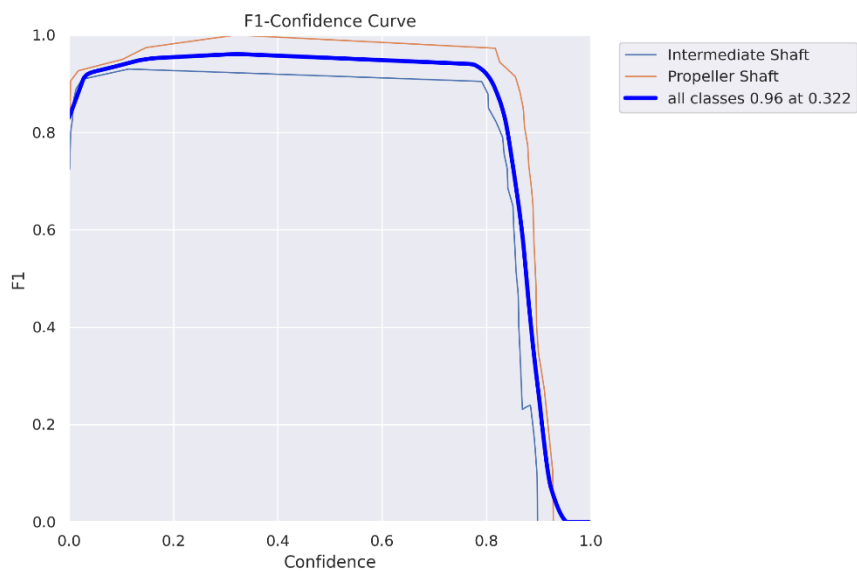


Figure 6.10 F1 - Confidence Curve

The curve appears to be flat at a high average F1 Score of 0.91 – 0.93 for a wide range of confidence values up to 0.8 which means that the model remains precise with minimal false detections, even when the model is conservative with higher confidence rate.

Lastly, we may output the confusion matrix which shows the behavior of our model on how many predicted classes are accurately predicted:

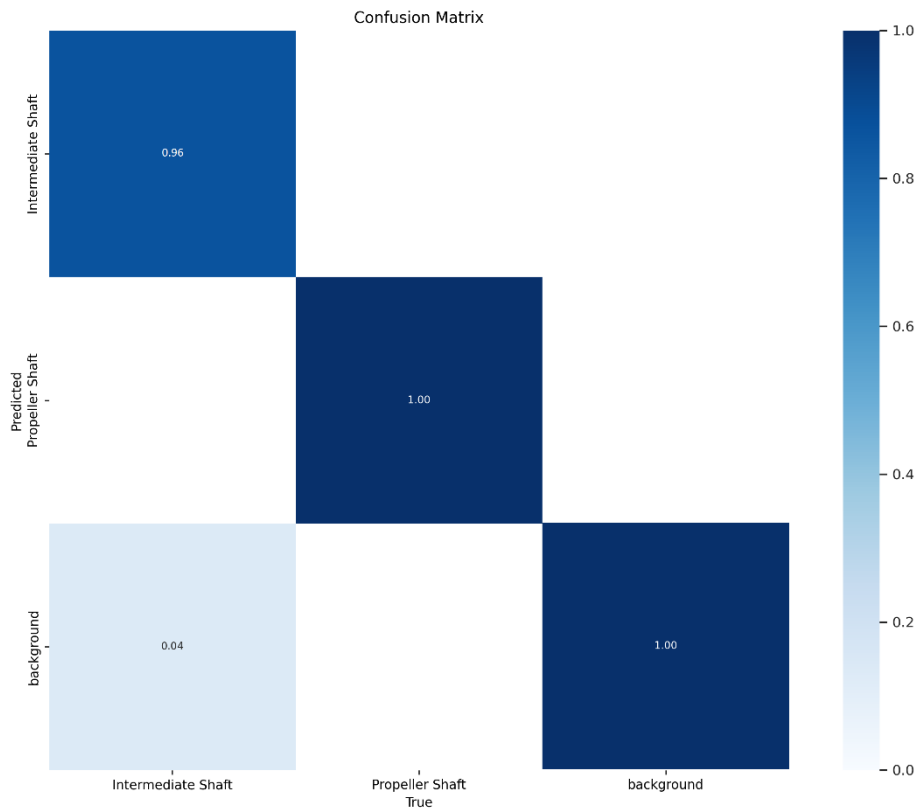


Table 6.2 – Confusion Matrix for the Shaft Detection Model

According to the Confusion Matrix above, our model seems to be predicting accurately the majority of the classes very accurately with a minimal confusion of the intermediate with the background lines which is on the spectrum of the acceptable error.

Overall, the Shafts Detection Model is a very accurate YOLO model which is sufficient for this thesis case and will be able to perform accurately on the vast majority of Shaft Arrangement Plans, detecting the Propeller and Intermediate Shaft with a great accuracy.

6.6.3 Dimensions Detection Model Evaluation

Finally, we evaluate the performance of the Dimensions Detection Model. The Dimension Detection model has only 1 class the horizontal dimension, and since each image includes a variety of horizontal dimensions, the model should be able to learn and become robust and accurate in detecting dimensions. Those dimensions will be later be passed on the OCR module of the main program for the dimension extraction procedure.

The model reached its best version when exported at $epoch = 200$ with the following performance metrics:

Object Detection Model - Key Entities - Metrics					
Epoch	Box Loss	Class Loss	Dual Focal Loss	mAP _{0.50}	mAP _{0.50-0.95}
200	0.6297	0.4174	0.7989	0.833	0.4599

Both the Box, Class and DF Loss are minimal (below 1) which reveal the model to be very accurate in terms of prediction the bounding box and the object class correctly even when the classes are under-represented. The Mean Average Precision for an IoU threshold of 0.5 is **mAP_{0.50} = 83.3**, and the mAP for all thresholds of 0.5 – 0.95 is **mAP_{0.50-0.95} = 46.0** which are exceptionally high, hence our model is very precise and accurate on its predictions.

After the model's training we may output the performance graphs of the training process:

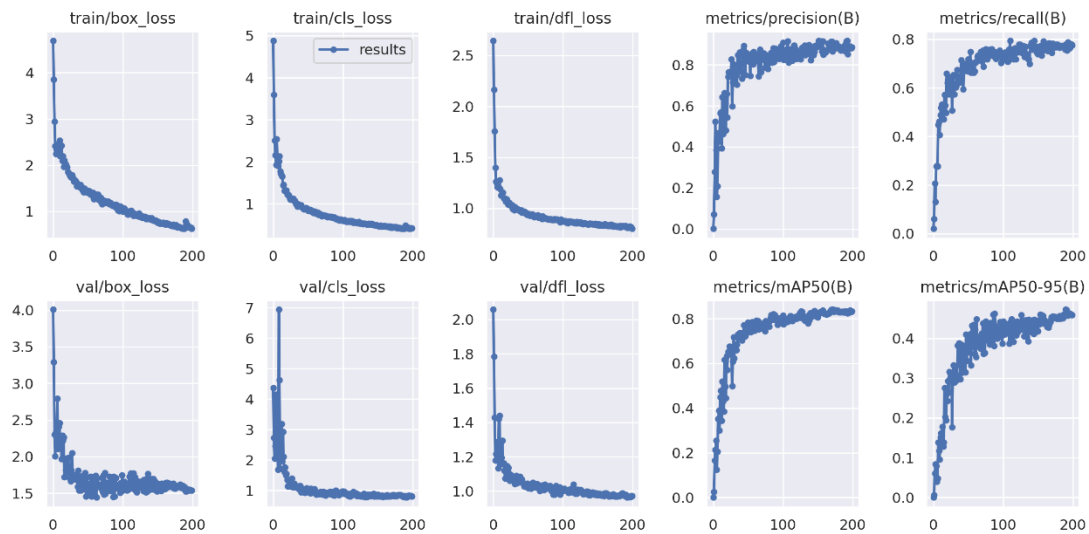


Figure 6.11 YOLOv8 Key Entities Training Performance

According to the figure above, the box loss, class loss and train loss is being minimized safely without overshoot for both the train and the validation sets, meaning that the model is well robust against overfitting and its performance on new data is acceptable. Both precision and recall increase. Precision appears to be constant at the 80 first epochs and forth, and the Recall appears to be increasing throughout the epochs. The mAP_{0.50} shows minimal increase after the 100th epoch and the mean average precision for all thresholds of 0.5 – 0.95 appear to increase adequately. We estimate that *further training would not benefit* the model and would only lead to overfitting, hence *no change in the training hyperparameters* is required.

A representation of the Precision and the Recall of the model can be visualized with the Precision-Recall Curve:

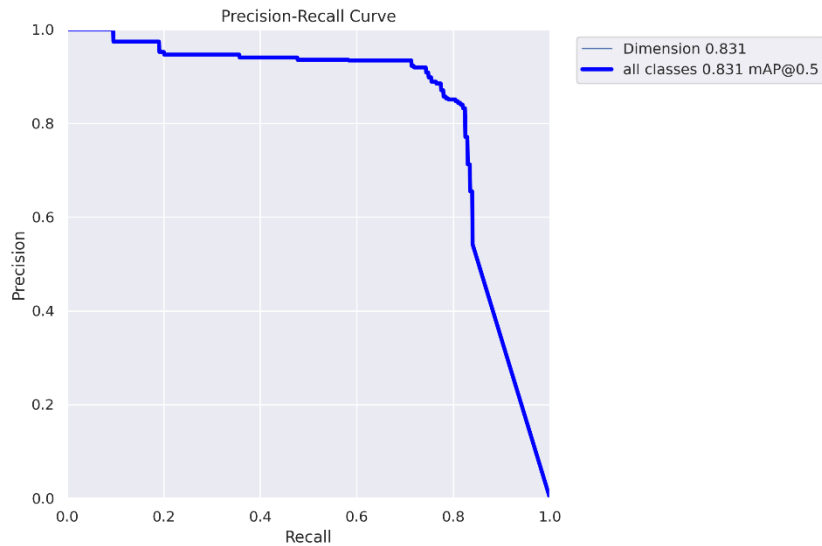


Figure 6.12 Precision - Recall Curve

The model appears to have a high precision even when the recall is high which signifies that a model outputs a high ratio of true accurately labeled classes while keeping a low ratio of falsely labeled classes. A better representation of their relationship can be seen in the F1 Score/Confidence curve below where F1 Score is the harmonic mean of Precision and Recall:

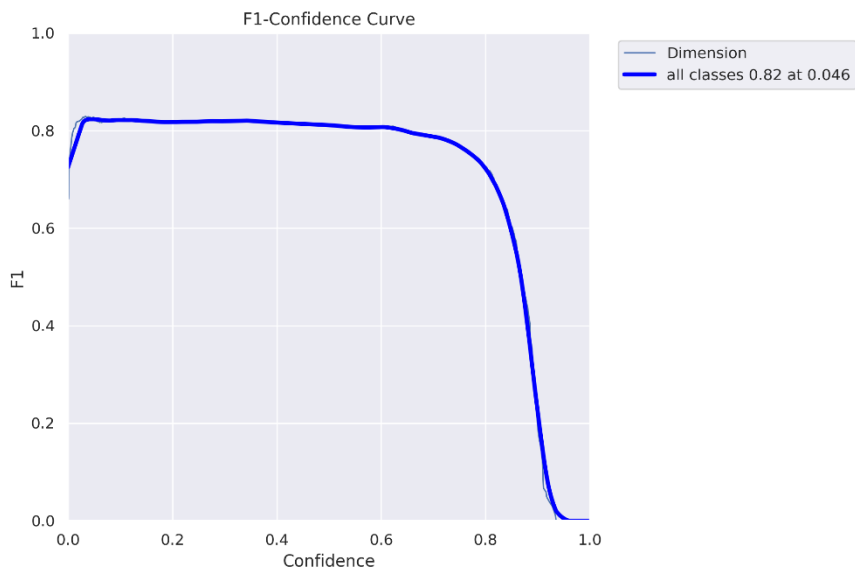


Figure 6.13 F1 - Confidence Curve

The curve appears to be flat at a high average F1 Score of 0.90 – 0.92 for a wide range of confidence values up to 0.8 which means that the model remains precise with minimal false detections, even when the model is conservative with higher confidence rate.

Lastly, we may output the confusion matrix which shows the behavior of our model on how many predicted classes are accurately predicted:

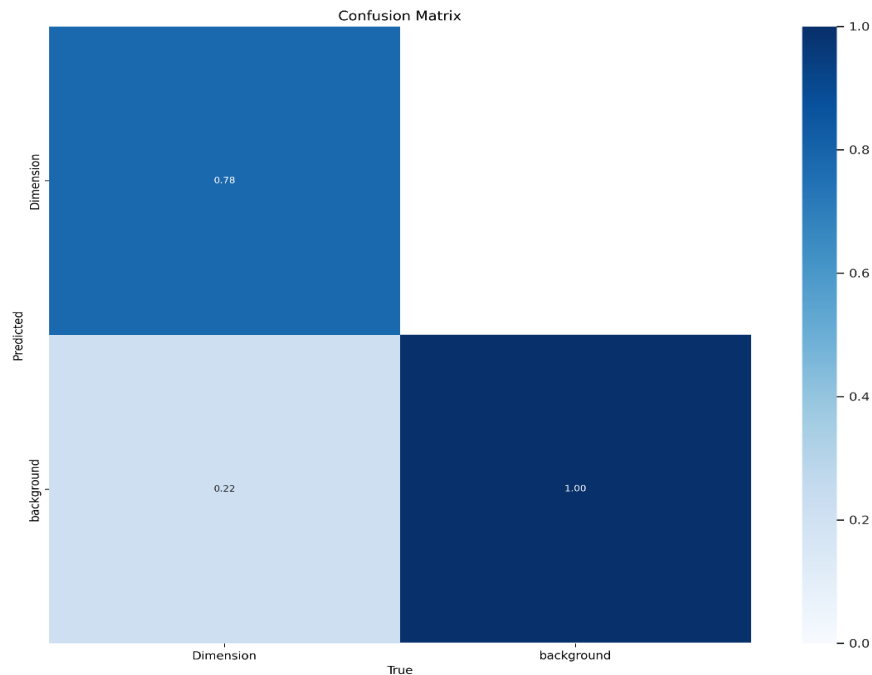


Table 6.3 – Confusion Matrix for the Dimension Detection Dataset

According to the Confusion Matrix above, our model seems to be predicting accurately our dimension class very accurately with a minimal confusion with the background lines which is on the spectrum of the acceptable error.

Overall, the Dimension Detection Model is a very accurate YOLO model which is sufficient for this thesis case and will be able to perform accurately on the vast majority of Shaft Arrangement Plans, detecting the Propeller and Intermediate Shaft with a great accuracy.

6.7 YOLOv8 Inference with Shaft Arrangement Samples

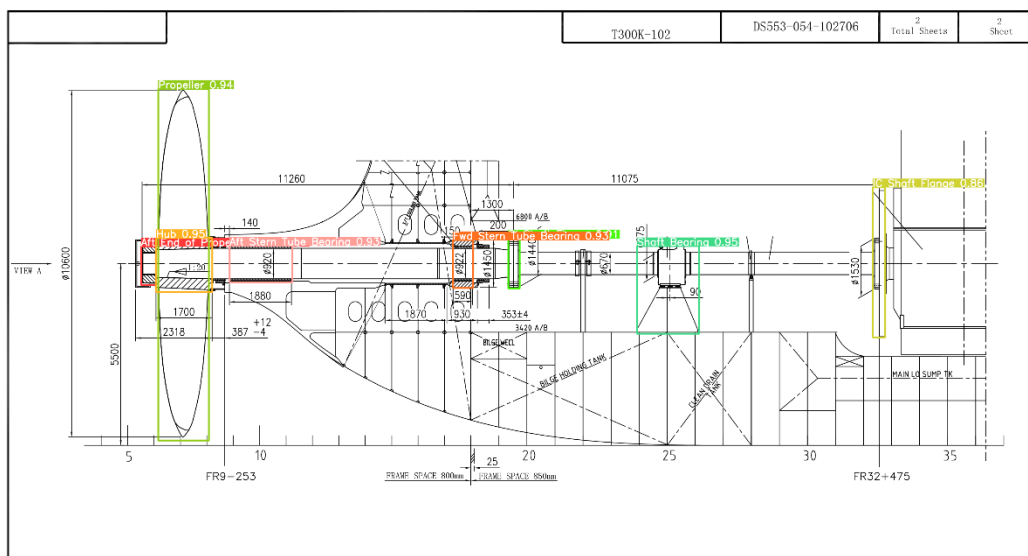
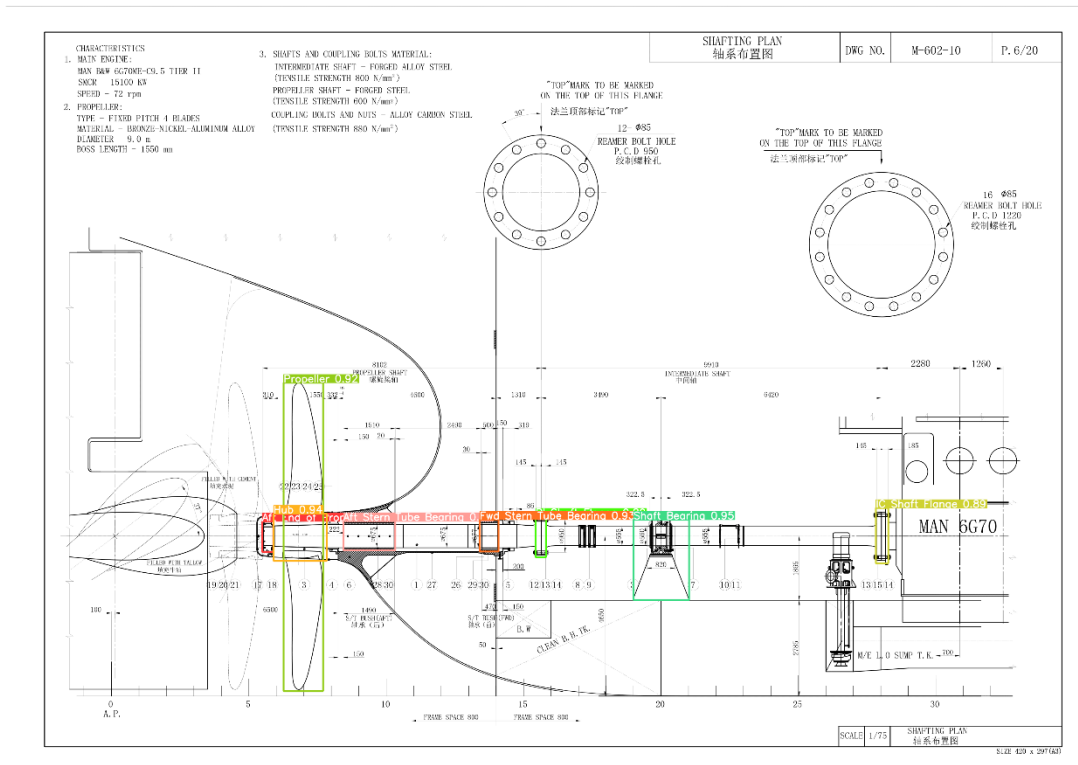
Finally, we may perform Inference using the final trained models for each task:

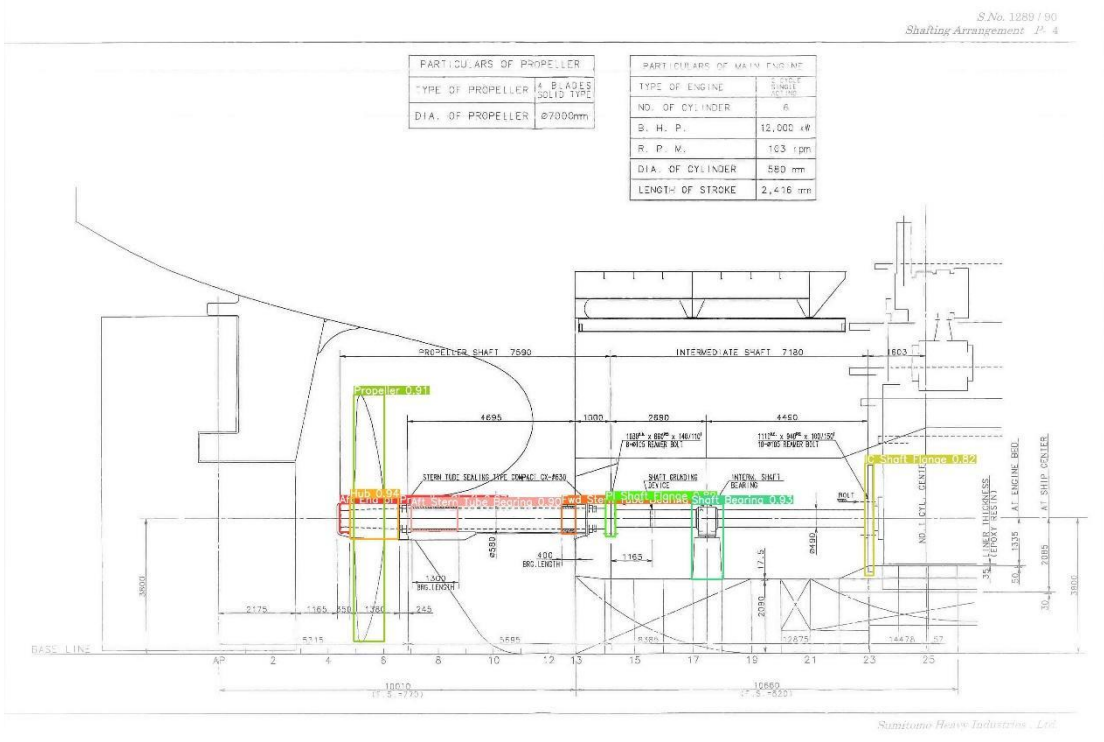
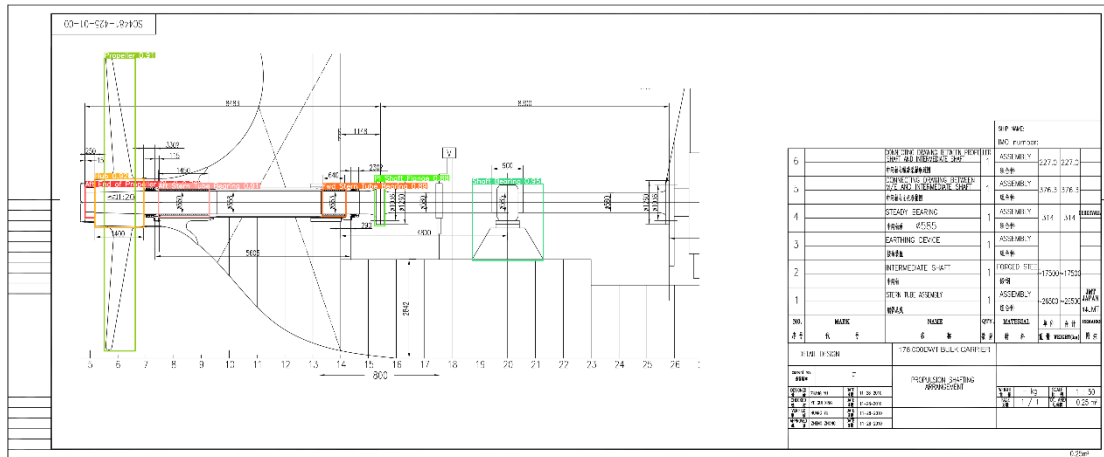
- 1) Detecting the Key Entities (Propeller, Aft End, Aft Stern Tube Bearing, Fwd Stern Tube Bearing, Flanges and Shaft Bearings)
- 2) Detecting the Shafts themselves: (Propeller Shaft, Intermediate Shaft)
- 3) Detecting Horizontal Dimensions

YOLOv8's CLI will be proven useful again, because with a simple command and with prediction mode we can easily define the model and the image, and the output shall be generated along with a detection file of the bounding box coordinates. For each of the models:

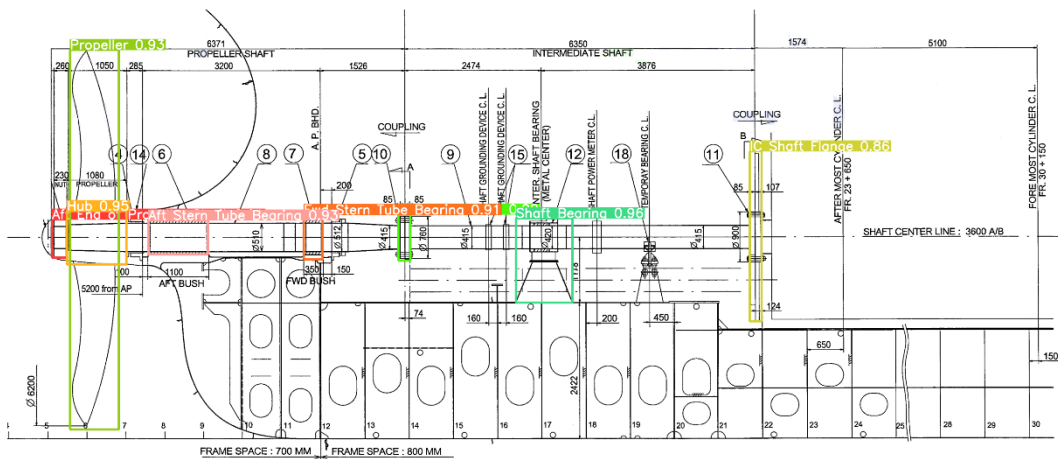
6.7.1 Key Entities Detection Model Inference

The inference with the Key Entities Detection Model outputs the following results:



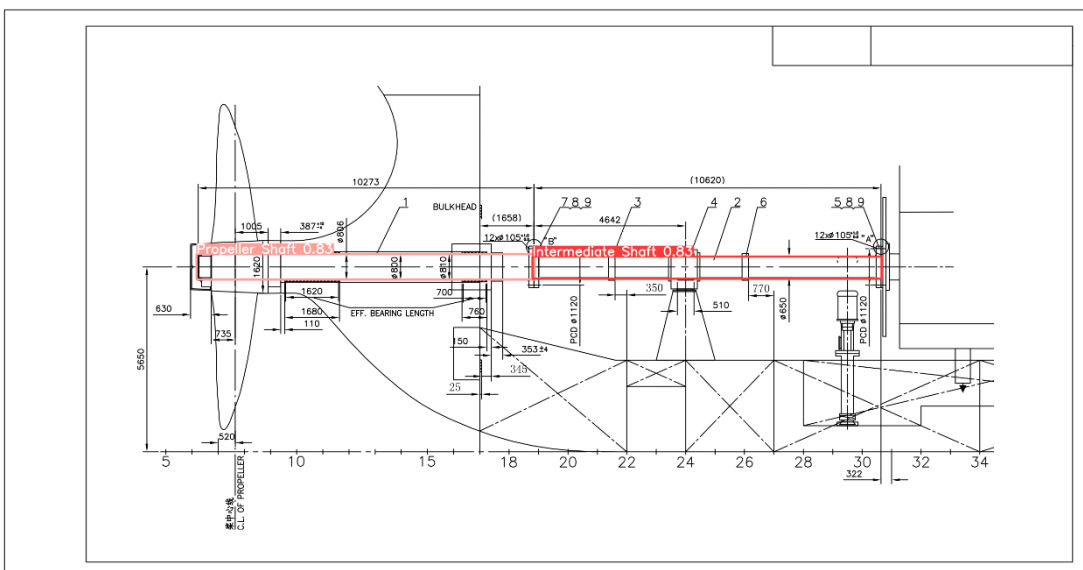
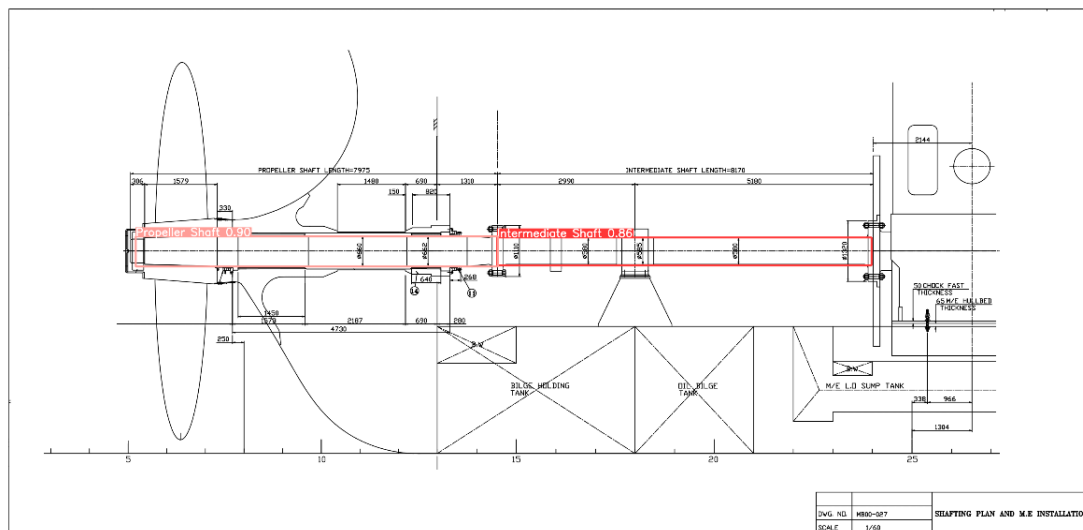
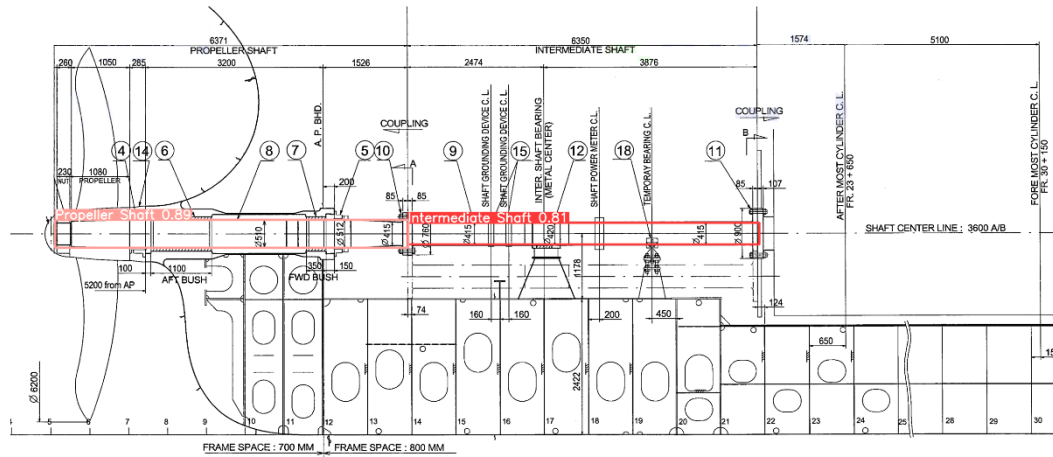


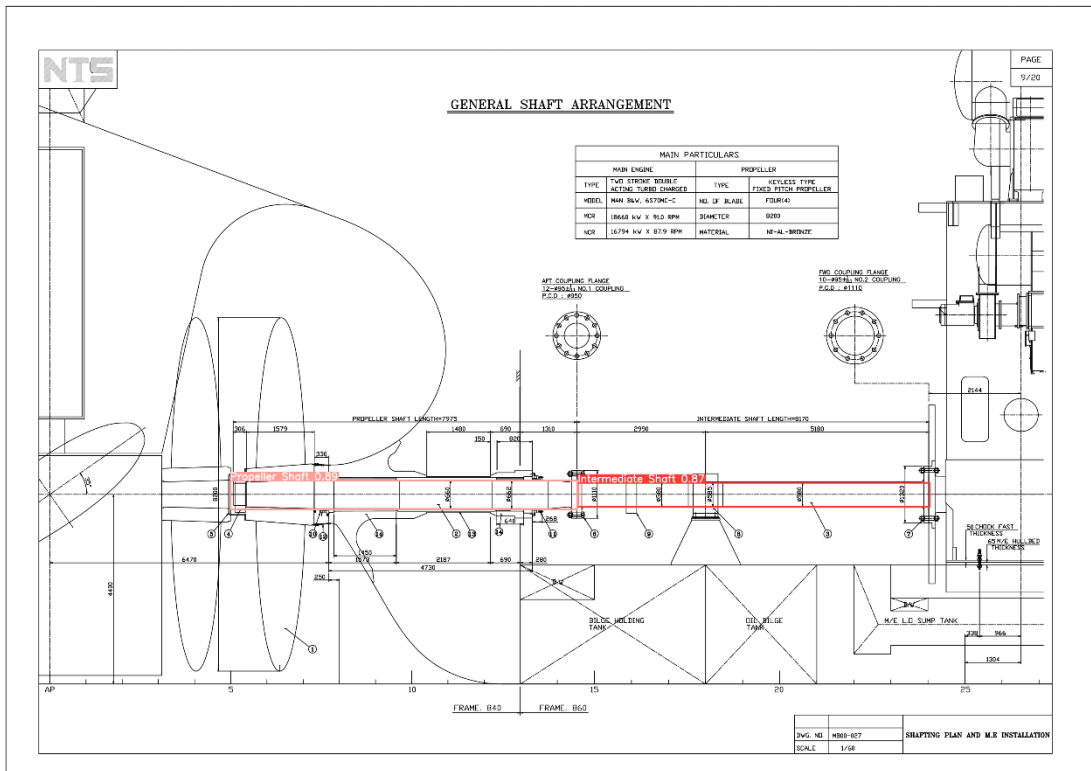
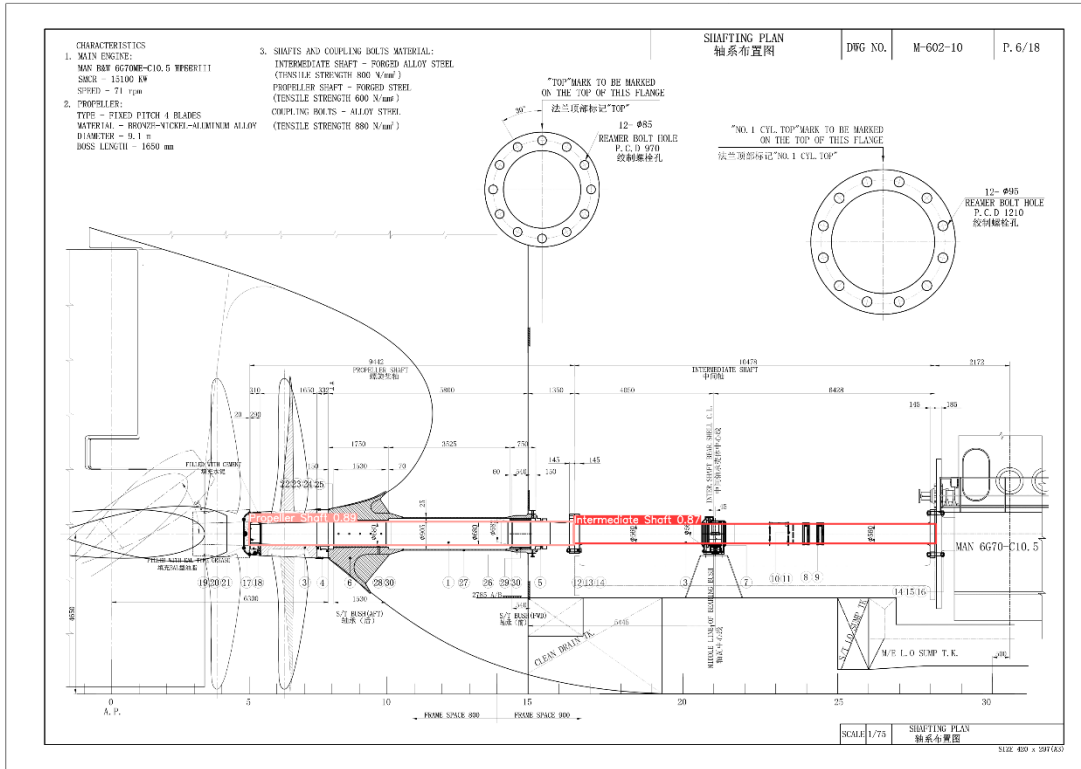
SHAFTING ARRANGEMENT



6.7.2 Shafts Detection Model Inference

We continue the inference using the Shaft Detection Model for a sample of images in our dataset.





Figures 6.14 – 6.24 – Inference results with the trained Object Detection Models

Hence the model predicts with an amazing accuracy all the entities of the Shaft Drawing which we want in order to develop a digital model of the Shaft Arrangement. The 3 models are now ready to be implemented inside our main shaft modeling algorithm.

7

Shaft Modeling Results

Since our Object Detection Models are able to predict the bounding boxes of the mechanical entities we want, we are able to use them for the Shaft Modeling.

7.1 Details of the Application

As we have previously mentioned in the Methodology section, our Application uses an image of the Shaft Arrangement Plan as an input and it exports a data file which has the diameter D , the weight w , the moment of inertia I and the distance L of each of the following segments:

- A) AB Segment from Aft End of Propeller till the Center of the Propeller.
- B) BC Segment from the Center of the Propeller till the Center of the Aft Stern Tube Bearing.
- C) CD Segment from the Center of the Aft Stern Tube till the Fwd Stern Tube Bearing.
- D) DE Segment from the Center of the Fwd Stern Tube Bearing till the Shaft Flange center.
- E) EF Segment from the Center of the Shaft Flange till the Shaft Bearing Center
- F) FG Segment from the Center of the Shaft Bearing till the Engine Flange Center.

The Center of the Propeller is modeled as the point load of the propeller weight and the center of the aforementioned bearings are modeled as the reactions where the reaction forces are acted upon. The Center of the Engine Flange as the end of the Intermediate Shaft could also act as the center of the Main Engine's weight, although in case there is a more detailed analysis of the piston weights and the bearings in the crankshaft, the detailed analysis is preferred.

7.2 Output Results for the Shaft Modeling

The model as inputs requires us to have the imported trained YOLOv8 models for the Key Entities, the Shafts and the Dimension Object Detecting, alongside with their appropriate class file. As long as we have imported the main libraries, we may run our main code and generate the following outputs:

We initially perform the Inference with the input Shaft Arrangement Image with the appropriate commands in order to export the inference image and the bounding box file:

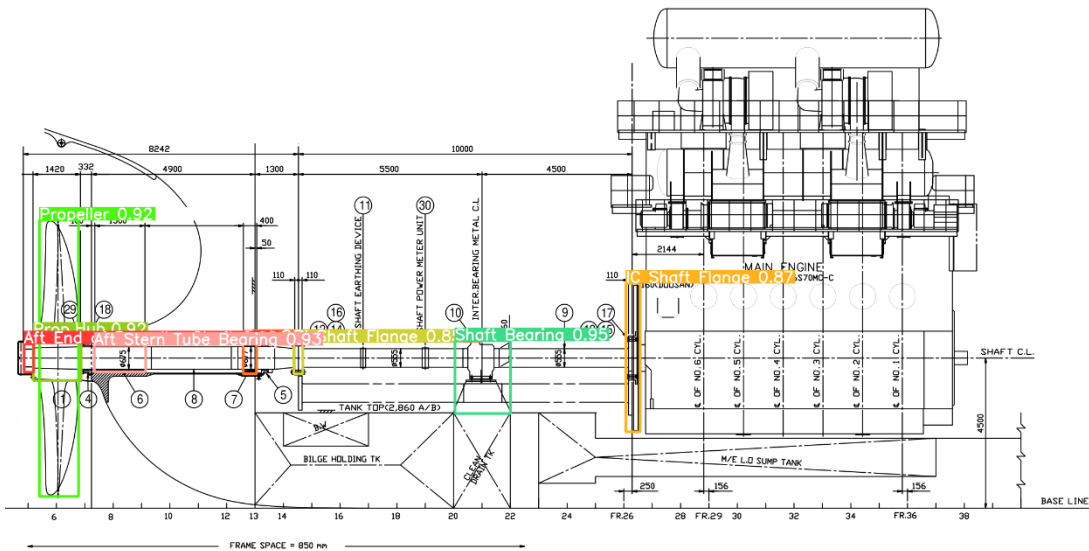
```

results = model.predict(source='PNG/9882.png', save=True, save_txt=True, conf=0.25)

Ultralytics YOLOv8.0.11 Python-3.8.10 torch-1.13.1+cpu CPU
Fusing...
Model summary: 268 layers, 68131272 parameters, 0 gradients, 257.4 GFLOPs
Results saved to c:\Users\user\Documents\YOLOv8\ultralytics\runs\detect\predict
1 labels saved to c:\Users\user\Documents\YOLOv8\ultralytics\runs\detect\predict\labels

```

Both the outputs are exported in the folder the project has been worked on. The inference image we receive is the following:



From there we use the generated bounding box file (labels file) and import it in our script as a Pandas dataframe:

```

import pandas as pd
data = pd.read_csv(f'{HOME}/ultralytics/runs/detect/predict/labels/9882.txt', sep=' ', header=None)

display(data)
✓ 0.7s

```

	0	1	2	3	4
0	2	0.220845	0.628342	0.011938	0.055259
1	4	0.264463	0.628342	0.008264	0.055259
2	3	0.566575	0.627451	0.012856	0.260250
3	5	0.048439	0.628788	0.035354	0.483957
4	6	0.046832	0.627897	0.043159	0.079323
5	0	0.021350	0.628342	0.008724	0.049911
6	1	0.103535	0.629234	0.046373	0.046346
7	7	0.430900	0.661765	0.050046	0.127451

Each of those columns represent the class ID number and their YOLO positions, hence develop modules which can process such Pandas Dataframes and extract the necessary information of the detected objects as well as convert the YOLO bounding box format to OpenCV format.

Following those necessary modules, we add a validation module for each of the detected class, with a necessary first step to count how many objects have we detected as seen below.

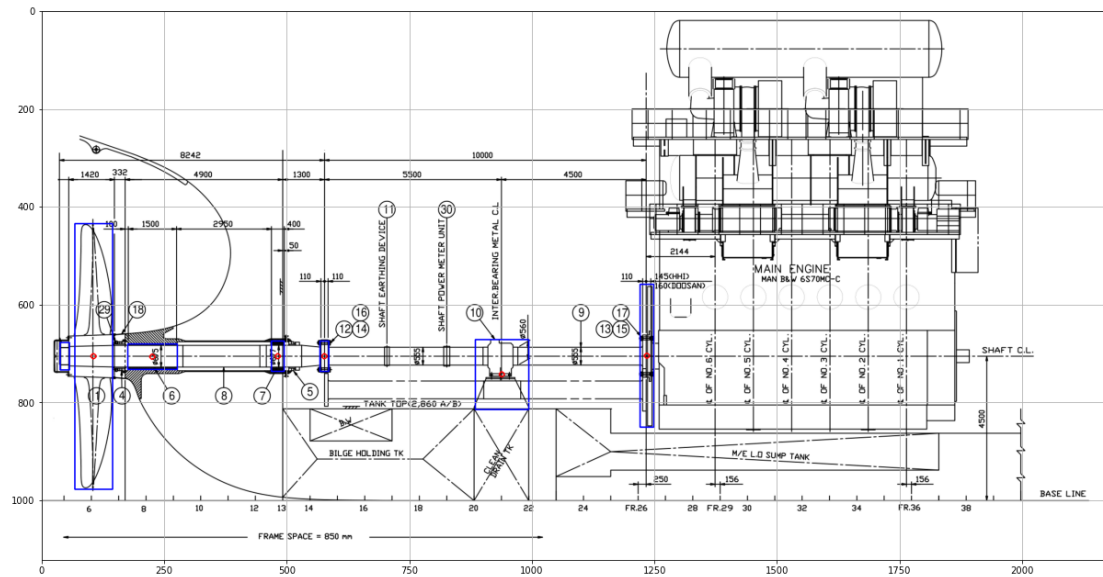
```
#Validating our objects how many times we found them
for x in range(labels.size):
    names_index = x
    obj_count = obj_freq(data, names_index)
    obj_name = class_name(names_index, labels)
    print(obj_name, ' was found: ', obj_count, 'times')
✓ 0.2s

Aft End of Propeller Shaft was found: 1 times
Aft Stern Tube Bearing was found: 1 times
Fwd Stern Tube Bearing was found: 1 times
IC Shaft Flange was found: 1 times
PI Shaft Flange was found: 1 times
Propeller was found: 1 times
Propeller Hub was found: 1 times
Shaft Bearing was found: 1 times
```

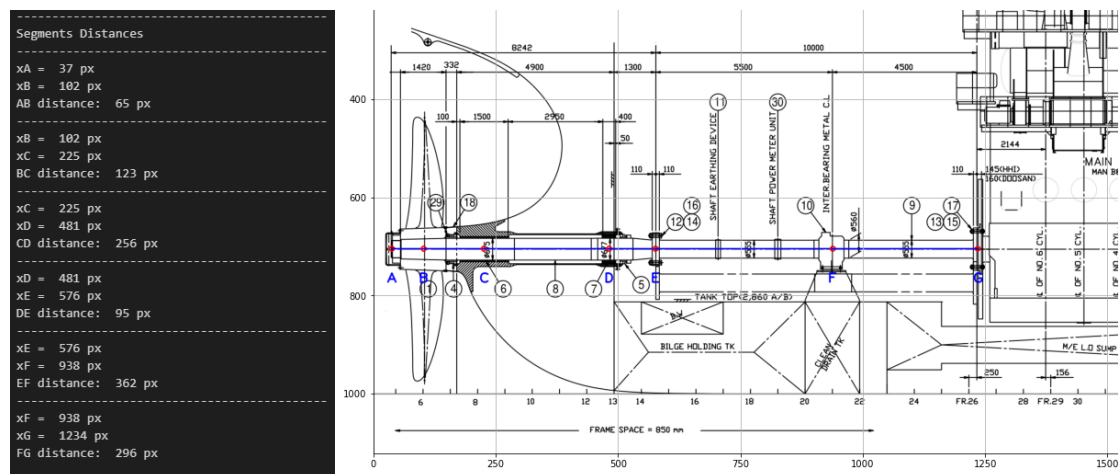
After performing the necessary validations, we are able to export the validated bounding box and its center for each validated object. Our following output consists of the times we detected each object, the bounding box coordinates conversion and the message of the validation. After the object is being validated its been assigned a validated status with a validated center whose coordinates we output in the following report:

```
Propeller was found: 1 times
Boudning box of: Propeller
YOLO Coordinates converted!
YOLO Coordinates: ( 0.0484389 0.628788 0.0353535 0.483957 )
OpenCV Coordinates: (67, 434, 144, 977)
-----
Propeller validly detected
Propeller has the following properties
Point P: Propeller Center
Coordinates: ( 105 705 )
Aft End of Propeller Shaft was found: 1 times
Boudning box of: Aft End of Propeller Shaft
YOLO Coordinates converted!
YOLO Coordinates: ( 0.0213499 0.628342 0.0087236 0.0499109 )
OpenCV Coordinates: (37, 677, 56, 733)
-----
Aft End of Propeller validly detected
Aft End has the following properties
Point A: Unconstrained
Coordinates: ( 37 705 )
Aft Stern Tube Bearing was found: 1 times
Boudning box of: Aft Stern Tube Bearing
YOLO Coordinates converted!
YOLO Coordinates: ( 0.103535 0.629234 0.0463728 0.0463458 )
OpenCV Coordinates: (175, 680, 276, 732)
-----
Aft Stern Tube Bearing validly detected
Aft Stb has the following properties
Point C: Constrained: Aft STB center
Coordinates: ( 225 706 )
Fwd Stern Tube Bearing was found: 1 times
Boudning box of: Fwd Stern Tube Bearing
YOLO Coordinates converted!
YOLO Coordinates: ( 0.220845 0.628342 0.0119376 0.0552585 )
OpenCV Coordinates: (468, 674, 494, 736)
-----
Fwd Stern Tube Bearing validly detected
Fwd Stb has the following properties
Point D: Constrained: Fwd STB center
Coordinates: ( 481 705 )
PI Shaft Flange was found: 1 times
Boudning box of: PI Shaft Flange
YOLO Coordinates converted!
YOLO Coordinates: ( 0.264463 0.628342 0.00826446 0.0552585 )
OpenCV Coordinates: (567, 674, 585, 736)
-----
Propeller-Interm. Shaft Flange validly detected
At this point the diameter of the shaft changes dimension
PI Flange has the following properties
Point E: Constrained: PI Flange center
Coordinates: ( 576 705 )
Shaft Bearing was found: 1 times
We have: 1 intermediate shafts
Boudning box of: Shaft Bearing
YOLO Coordinates converted!
YOLO Coordinates: ( 0.4309 0.661765 0.0500459 0.127451 )
OpenCV Coordinates: (884, 671, 993, 814)
-----
Shaft Bearing validly detected
Bearing has the following properties
Point F: Constrained: Bearing
Coordinates: ( 938 742 )
IC Shaft Flange was found: 1 times
Boudning box of: IC Shaft Flange
YOLO Coordinates converted!
YOLO Coordinates: ( 0.566575 0.627451 0.0128558 0.26025 )
OpenCV Coordinates: (1220, 558, 1248, 850)
-----
Engine Flange validly detected
IC Flange has the following properties
Point G: Constrained: IC Flange
Coordinates: ( 1234 704 )
Propeller Hub was found: 1 times
Boudning box of: Propeller Hub
YOLO Coordinates converted!
YOLO Coordinates: ( 0.046832 0.627897 0.0431589 0.0793226 )
OpenCV Coordinates: (55, 660, 149, 749)
-----
Propeller Hub validly detected
Hub Center has the following properties
Point B: Will be used as the propeller center only if no propeller
Coordinates: ( 102 704 )
```

A figure of all validated object bounding boxes can be found in the figure below. The grid the figure below has is in pixels units with the start of (0,0) at the up-left side of the canvas:



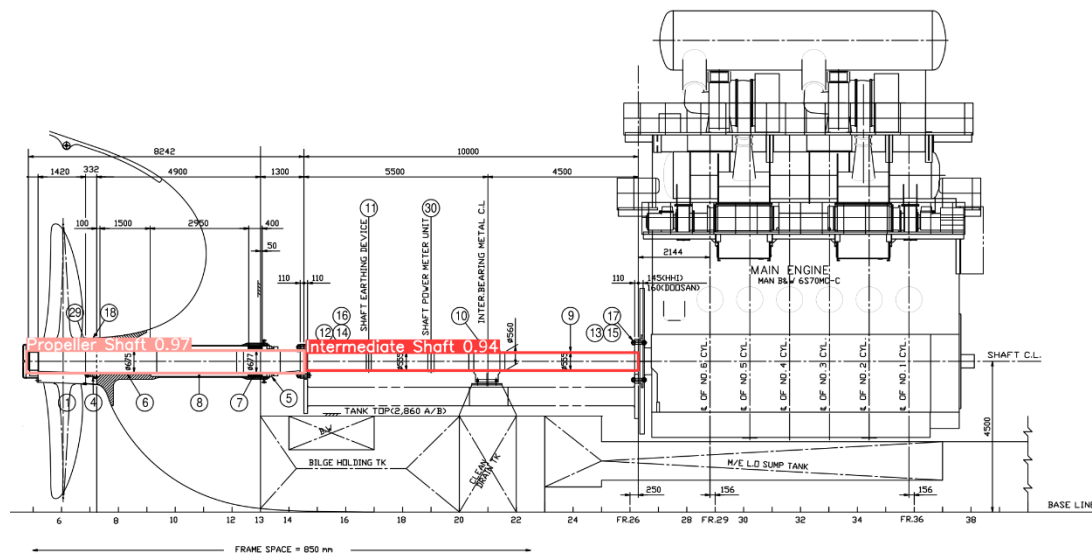
Since we have exported the validated geometry of the detected objects, we can now generate the segments A to G which are required to model the shaft arrangement. We are able to export their distances in pixel units:



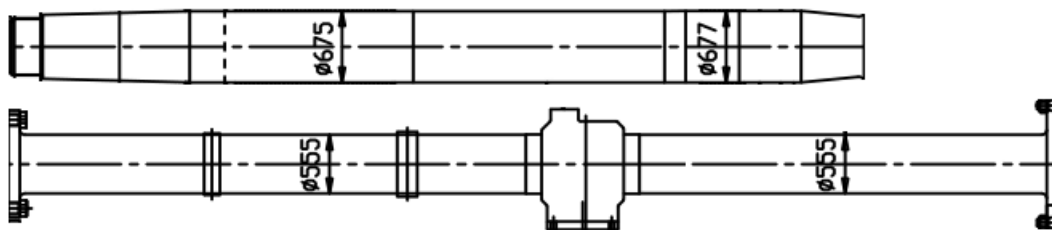
Above the generated shaft line is a results of the average of the y-centers of all the objects excluding the Shaft bearing, whose geometrical center does not belong in the shaft line.

Our next step is to convert those segment distances into real (mm) dimensions as well as finding the diameter of the Propeller Shaft (Segment AE) and the Intermediate Shaft (Segment EG). In order to find the Diameters in (mm) of the shafts we employ the use of our trained YOLO shaft detection model, find the bounding boxes of the 2 shafts, and process them into our OCR module with the necessary preprocessing and output the average diameter in case the shaft has more than one diameters in its length:

In the figure below we perform the inference of the Shafts Detection Model with great accuracy:



Our program extracts the bounding boxes of the 2 shafts and process them in our OCR routine:



Our OCR module performs the digits recognition using the following steps:

Rotating Image: We rotate the image 90 degrees clockwise for the characters to be readable by OCR Space. Enhancing Details: We enhance the image by performing Edge Sharpening and Noise removal in order to make it easier to OCR the image successfully. Enlarging image: We enlarge the image 3x its original size while maintaining the aspect ratio using OpenCV's native library in order to make the characters larger. Securing a connection with OCR Space using an API Key and delivering the result in a tuple variable which contains the numbers we have processed. Inside the tuple itself

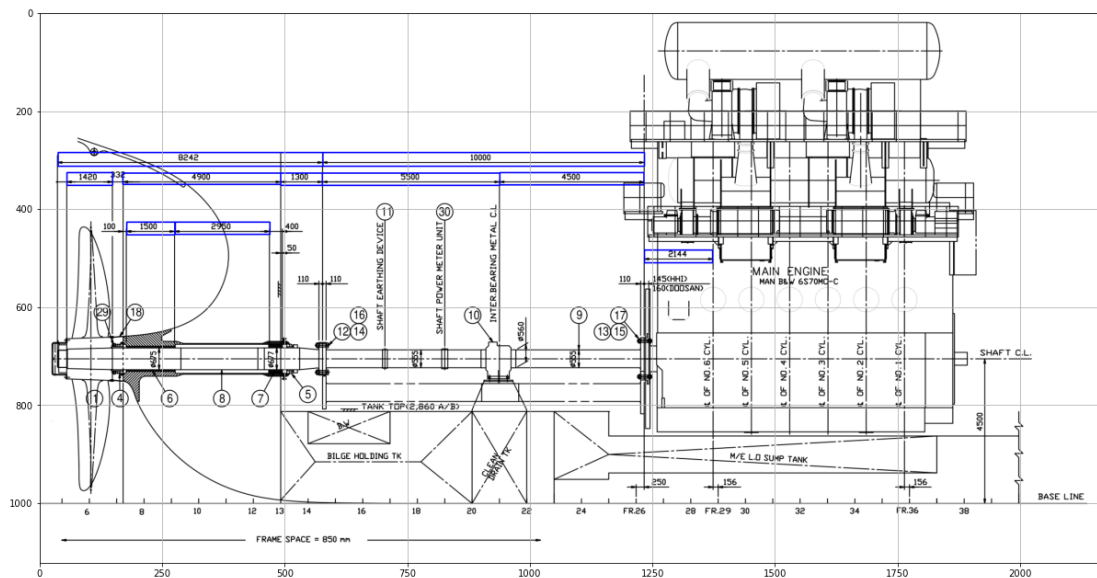
```

...Performing OCR...
-----
Propeller Shaft
-----
Preprocessing Image
- Extracting Image: Propeller Shaft - Done
- Rotating Image: ROTATE_90_CLOCKWISE - Done
- Enhancing Details: Edge Sharpening - Done
  Noise Removal - Done
- Enlarge Image: Initial Image: (44, 545)
  Resized Image: (132, 1635)
- OCR Space: API key: K8775698888957
-----
OCR Result
[675, 677]
Propeller Shaft Diameter 676.0 mm
Propeller Shaft Diameter 44 px
-----
Intermediate Shaft
-----
Preprocessing Image
- Extracting Image: Intermediate Shaft - Done
- Rotating Image: ROTATE_90_CLOCKWISE - Done
- Enhancing Details: Edge Sharpening - Done
  Noise Removal - Done
- Enlarge Image: Initial Image: (35, 652)
  Resized Image: (105, 1956)
- OCR Space: API key: K8775698888957
-----
OCR Result
[555, 555]
Intermediate Shaft Diameter 555.0 mm
Intermediate Shaft Diameter 35 px

```

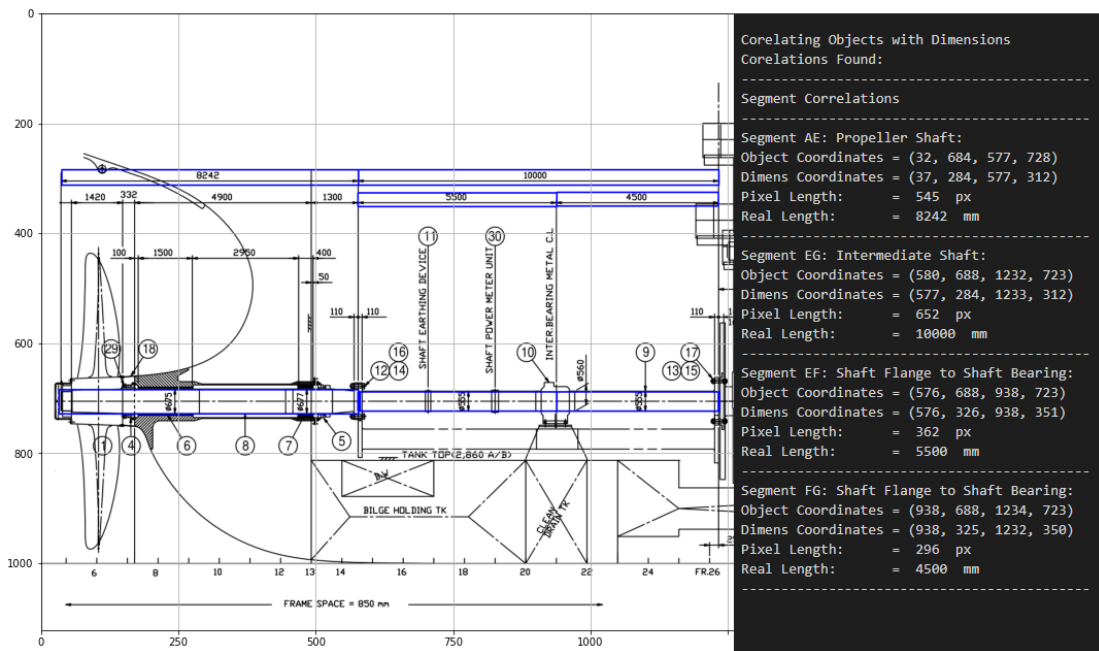
we calculate the average of the diameters and output the average diameter for each one of the shafts.

The last step is to calculate the real (in mm) lengths of the segments, which is the only information missing from completing the shaft's digital modeling. We employ the use of the trained Dimension Object Detection model, and we perform inference with it to detect all the available horizontal dimensions:



<p>Dimension 1</p> <p>Pixel Length = 93 px OCR Length = 1420 mm mm-px Ratio = 0.06549 mm/px</p>	<p>Dimension 6</p> <p>Pixel Length = 85 px OCR Length = 1300 mm mm-px Ratio = 0.06538 mm/px</p>
<p>Dimension 2</p> <p>Pixel Length = 98 px OCR Length = 1500 mm mm-px Ratio = 0.06533 mm/px</p>	<p>Dimension 7</p> <p>Pixel Length = 362 px OCR Length = 5500 mm mm-px Ratio = 0.06582 mm/px</p>
<p>Dimension 3</p> <p>Pixel Length = 193 px OCR Length = 2900 mm mm-px Ratio = 0.06655 mm/px</p>	<p>Dimension 8</p> <p>Pixel Length = 294 px OCR Length = 4500 mm mm-px Ratio = 0.06533 mm/px</p>
<p>Dimension 4</p> <p>Pixel Length = 322 px OCR Length = 4900 mm mm-px Ratio = 0.06571 mm/px</p>	<p>Dimension 9</p> <p>Pixel Length = 656 px OCR Length = 10000 mm mm-px Ratio = 0.0656 mm/px</p>
<p>Dimension 5</p> <p>Pixel Length = 540 px OCR Length = 8242 mm mm-px Ratio = 0.06552 mm/px</p>	<p>Dimension 10</p> <p>Pixel Length = 139 px OCR Length = 2144 mm mm-px Ratio = 0.0656 mm/px</p>
<p>Average mm to px Ratio (x axis) mm-px Ratio = 0.06563 mm/px</p>	

Our program now correlates the bounding boxes of the detected objects with the detected segments we have from the Key Entities detection model. We remember that if the bounding boxes of the dimensions and the segments (A-G) have common start and ending points (in pixels) (x_{first} , x_{last}). Searching and comparing the available bounding box dimensions with the already existing detected segment locations (adding a 10-pixel tolerance to smooth any bounding box errors) we may derive that there are certain geometries which match 1-1 with our dimensions which are the following below are:



The dimensions which are not intrinsically match the detected dimensions will either be calculated as a percentage the propeller/intermediate shaft and in case there are inadequate detection data we may use the horizontal mm/pixel ratio which is calculated from the dimension detection procedure:

Segments not correlated intrisically with a dimension	
AB, BC, CD, DE	
Segment AB	Segment CD
Pixel Length = 65 px	Pixel Length = 256 px
Percentage of AE = 12.06 %	Percentage of AE = 47.5 %
Length AB = 994 mm	Length CD = 3915 mm
Segment BC	Segment DE
Pixel Length = 123 px	Pixel Length = 95 px
Percentage of AE = 22.82 %	Percentage of AE = 17.62 %
Length BC = 1881 mm	Length DE = 1452 mm

We have finally completed the Shafts Arrangement Geometry and we now know all the necessary lengths and diameters for each segment. Our only last step to finalize this modeling is to ask the user to enter the Density of the shaft's material as well as the Young Modulus in

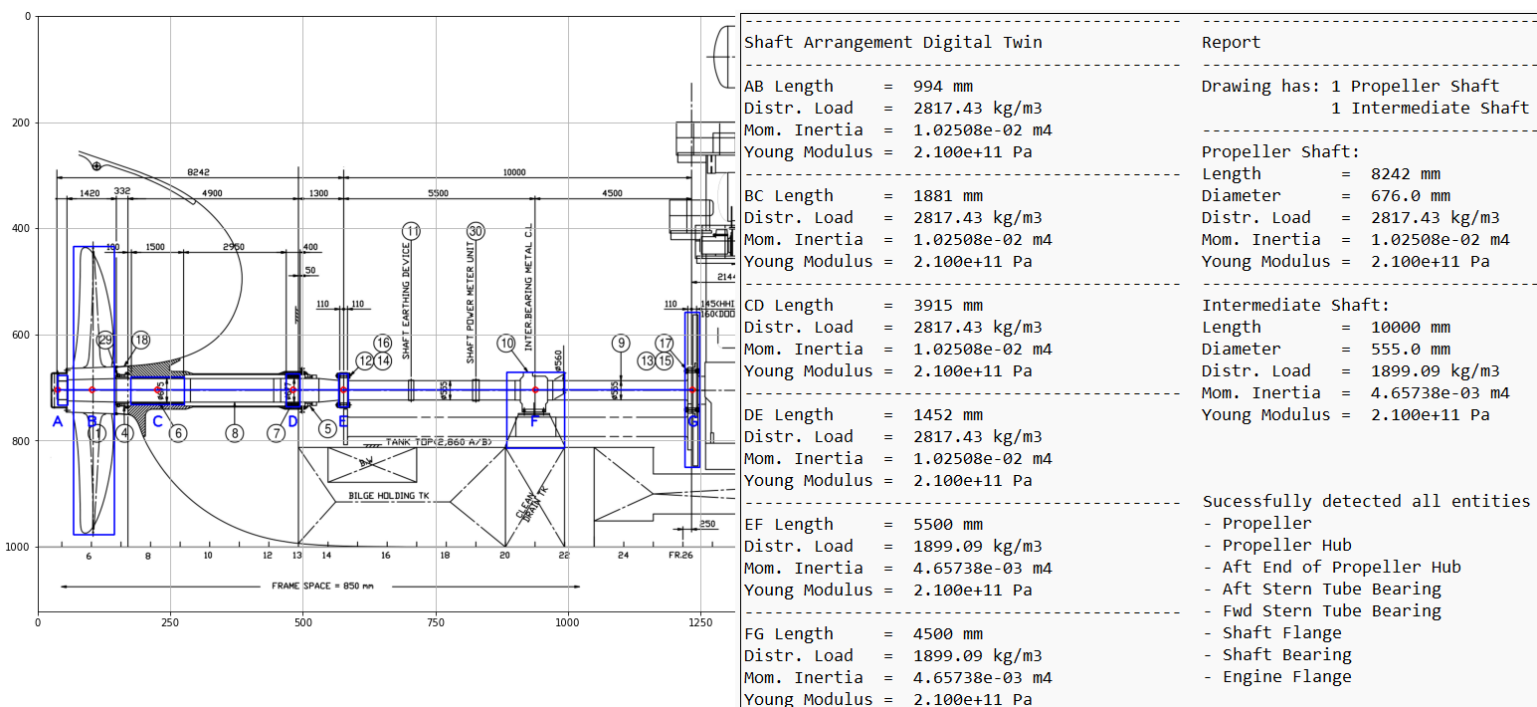
order to calculate for each segment its Weight Distribution and its Moment of Inertia using the known formulas below:

$$w = \frac{W}{L} = \frac{\rho V}{L} = \frac{\rho}{L} \left(\frac{\pi D^2}{4} L \right) \rightarrow w = \frac{1}{4} \rho \pi D^2$$

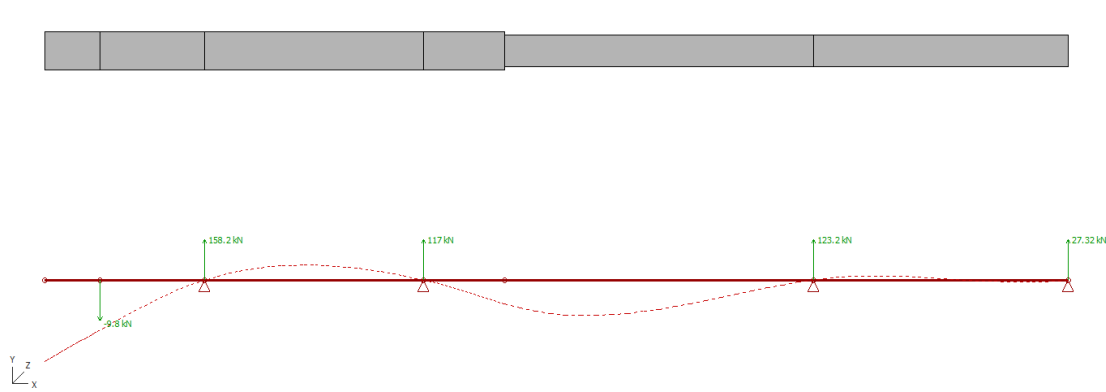
$$I = \frac{1}{4} \pi \left(\frac{D}{2} \right)^4 = \frac{1}{4} \pi \left(\frac{D^4}{16} \right) \rightarrow I = \frac{1}{64} \pi D^4$$

The segments which belong to the propeller shaft and the intermediate shaft respectively have the same diameter between them, thus the same weight distribution and moment of inertia.

We generate the final output report which contains the necessary length, weight and moment for each segment:



We can compare the results with the reality since the actual dimensions match the ones evaluated by our program. The Propeller Shaft is indeed 8,242 mm and the Intermediate Shaft is indeed 10,000 mm. The geometry which is outputted by our Object Detection is valid and it's according to the drawing above. The generated report file is ready to be imported at the NTUA Shaft Alignment and a great step has been made in order to automate the shaft alignment procedure when it comes to align vessels in bulk quantities. Assuming our propeller of Diameter D=8.3m (Z=3 blades) has a weight of 9800kg we may demonstrate the said model by importing the output file of our program into the Shaft Alignment Program. On this stage this will be performed manually and propose a future consideration of this being automatically imported on the program by having our program exported as a plugin to NTUA's software.

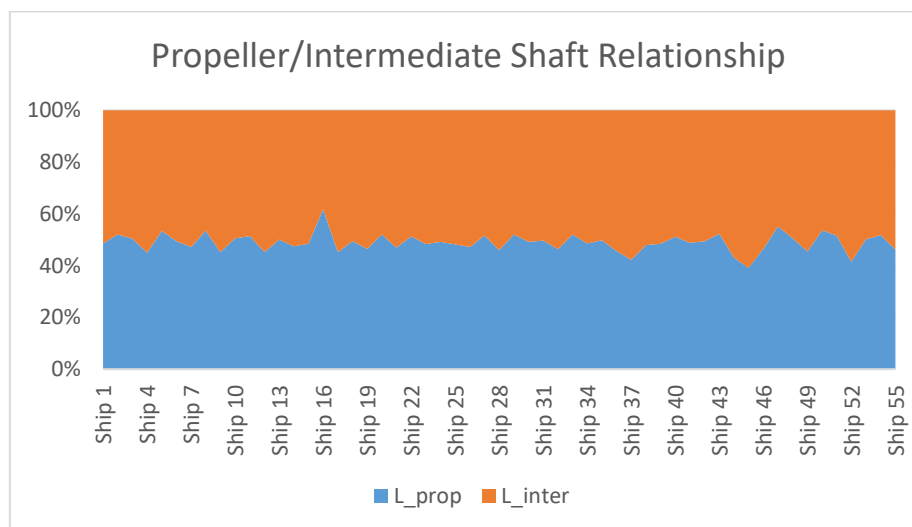


Figures 7.1 – Exported model into the Shaft Alignment Software by NTUA

7.3 Additional Results worth noting

We previously mentioned that according to NTUA Lab of Marine Engineering, there is a validation statistic between the length of the Propeller Shaft and the Length of the Intermediate Shaft which approximately estimates that if the dimensionless sum of the propeller and intermediate shaft is equal to 1, then the Propeller Shaft's Length is the 49.4% of the Total Shaft Length and the Intermediate Shaft's Length is the 50.6% of the Total Shaft Length with Standard Deviation of 2.2%

Upon performing digitalization on 55 vessels from our dataset we came into the conclusion that on average, the Propeller's Shaft Length takes $L_{PR} = 48,9\%$ and the Intermediate Shaft's Length takes $L_{PR} = 51.1\%$ of the total shaft arrangement length (the sum of the propeller and the intermediate shaft, not including the crankshaft length) with a standard deviation of 3%. This result comes in par with NTUA's distribution of 49.4% and 50.6% with only a 0.5% difference. Thus, we may derive that the analogy is correct with a deviation of 2-3%



8

Conclusion & Future Work

Marine engineers have a vital role in the design and maintenance of merchant vessels, with their ability to read and understand mechanical design drawings being a critical aspect of their field of work. The Shafting Arrangement Plan is a particularly important drawing for marine engineers, as it provides crucial information about the vessel's propulsion system. The current thesis aims to explore the design of an AI-driven software capable of understanding the structure, position and key dimensions of the most traditional mechanical components comprising a Propulsion Shafting System, and ultimately extracting automatically a draft digital twin geometry that can be imported in a Shaft Alignment software for further analysis. Such a tool could drive significant improvements in the efficiency and safety of maritime operations and could potentially revolutionize the field of marine engineering.

8.1 Synopsis and Conclusion

In the current thesis we successfully trained 3 important object detection YOLO models with great accuracy due to the YOLOv8's state-of-the art performance and novel libraries, as well as a large dataset of 100 different Shaft Arrangement Plan drawings which were preprocessed and augmented accordingly. Using Google Collab's GPU backend resources, we were able to perform transfer learning to YOLOv8x COCO pretrained model and we extract the necessary metrics to evaluate the model's performance. Each one of the 3 models scored a high Mean Average Precision metric, which signifies their ability to be highly accurate when it comes detecting the mechanical entities, while at the same time avoiding overfitting due to their great performance on the validation set and all that boast great Accuracy and Recall even if it's adjusted in a conservative confidence level. The developed Shaft Modeling software enabled the user to import shafting system models successfully and implement a combination of validations which mimic the traditional 'human' engineering way of thinking and acting, thus it was able to normalize results in the way a marine engineer would do. Associating horizontal dimensions and objects taking advantage of their explicit and quite predictable position and sequence is an important step into training machines thinking like humans. Encompassing OCR modules along with OCR preprocessing was necessary to digitize the Shaft

Arrangement's dimensions and associate them with certain detected bounding box features.

This way we were able to successfully:

- Detect the positions of the important mechanical entities which consist of the segments of our shaft arrangement (in pixels)
- Calculate the average diameters of the Propeller Shaft and the Intermediate Shaft as well as calculating a pixel to mm ratio for the vertical direction.
- Detect the positions of each dimension with bounding boxes and OCR the dimension features to extract their numerical values. Associating the dimension object lengths and values we may calculate an average mm to pixel ratio for the horizontal dimensions.
- Correlate the detected dimension positions/values with the positions of the segments we want to detect. Defining a small pixel tolerance (~10px) we may match the dimension objects with their real dimension value.
- Enable calculation of the other segments as a percentage of the propeller shaft or the intermediate shaft. In case of missing information, the user may utilize the mm to pixel ratio to calculate any segments with known pixel size accordingly.
- Manually added the known input of the Young Modulus and the Material density to calculate the Weight Distribution and the Young Modulus of each segment.
- The final report is generated in a file which may present the digital geometry of the shaft and can be imported in the Shaft Alignment software which is already developed.

8.2 Improvements using the developed software

Supposing a naval architecture office is assigned by an owner's company to check and perform shaft alignment on their fleet and the fleet consists of 150 vessels. Assuming each shaft arrangement modeling requires a minimum of 30 dimensions, several of which are manually calculated from the given dimensions, an engineer would have to manually enter to his Shaft Alignment Software an average of $150 \times 30 = 4500$ dimensions. This can be deemed as a time consuming and prone to mistakes process, whereas shaft alignment calculations have a very little tolerance for such mistakes. Minimizing the human error and time waste from the engineer's side, the developed software enables automating the shaft alignment process by extracting the whole shaft geometry in less than a few seconds. Such software enhances the engineer's productivity and efficiency while providing accurate and precise calculations. Such automation software presented in this thesis unlocks new pathways to further automating multiple tasks in the maritime industry and making marine engineering more efficient and focused on creativity where no AI can replace the human innovation and way of thinking.

8.3 Future work

This thesis is one of the first ones which employ the use of YOLOv8 as an object detection architecture and our results reveal how efficient and fast is comparing to its predecessors and other architectures available. Such progress reveals a new path to implement this thought processing and principles to expand the work presented and potentially make the modeling more efficient. Some recommended future scope of work would be:

- i) Experimenting with newer Object Detection Architectures when they are released. In case there more efficient networks which provide better results than YOLOv8 in the future, one could potentially use more than one architecture to compare how they perform in our Shaft Arrangement dataset.
- ii) Notably 100 original images were comprising our training dataset since it is exceptionally hard to find a vast number of unique drawings. Our dataset had to be augmented with several preprocessing methods in order to produce an accurate detection model. One future direction of work is to employ the use of Generative Adversarial Network (GANs). GAN networks are able to generate new samples using existing data. If we use the current images as input data to a properly developed GAN network, we can create a larger and more diverse shaft arrangement dataset which can improve the accuracy and generalizability of our detection model.
- iii) Additionally, our dataset was lacking vessels which consist of more than one Intermediate Shaft and/or have more than one Intermediate Shaft Bearings. Such vessels are large container ships or VLCC carriers whose engine room is large comparing to our dataset consisting of Bulk Carriers (mostly Kamsarmax and Suezmax vessels) and a few Crude Oil carriers. One could try to gather more of those vessels and try to generalize our digital shaft modeling software to encompass more than one intermediate shafts and bearings.
- iv) Performing a full integration of the Digital Shaft Modeling software into the Shaft Alignment software as an internal plugin which accepts drawings as an input file and exports the result data directly into the software without any external file generation.
- v) Additionally, we can attempt to generalize the concept of digitizing drawings to not only Shaft Arrangement plans but on further naval architecture drawings and implement those modeling algorithms in automating the ship design and evaluation process.

9

Bibliography

- [1] Ablameyko SV, Uchida S (2007) Recognition of engineering drawing entities: review of approaches. *Int J Image Graph* 07(04):709–733. <https://doi.org/10.1142/S0219467807002878>
- [2] Adam S, Ogier J, Cariou C, Mullot R, Labiche J, Gardes J (2000) Symbol and character recognition: application to engineering drawings. *Int J Doc Anal Recogn* 3(2):89–101. <https://doi.org/10.1007/s100320000033>
- [3] Bodic PL, Locteau H, Adam S, Heroux P, Lecourtier Y, Knippel A (2009) Symbol detection using region adjacency graphs and integer linear programming. In: 2009 10th international conference on document analysis and recognition. IEEE, pp 1320–1324. <https://doi.org/10.1109/ICDAR.2009.202>. <http://ieeexplore.ieee.org/document/5277721/>
- [4] Cordella L, Vento M (2000) Symbol recognition in documents: a collection of techniques? *Int J Docu Anal Recogn* 3(2):73–88. <https://doi.org/10.1007/s100320000036>
- [5] De P, Mandal S, Das A, Bhowmick P (2014) A new approach to detect and classify graphic primitives in engineering drawings. In: 2014 fourth international conference of emerging applications of information technology. IEEE, pp 243–248. <https://doi.org/10.1109/EAIT.2014.33>
- [6] Henderson TC (2014) *Analysis of engineering drawings and raster map images*. Springer
- [7] Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S et al (2017) Speed/accuracy trade-offs for modern convolutional object detectors. In: *IEEE CVPR*, vol 4
- [8] Hui J (2018) Object detection: speed and accuracy comparison (faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3).
- [9] Jin L, Zhou Z, Xiong S, Chen Y, Liu M (1998) Practical technique in conversion of engineering drawings to CAD form. In: *IMTC/98 conference proceedings. IEEE instrumentation and measurement technology conference. Where instrumentation is going (Cat. No. 98CH36222)*
- [10] Song J, Su F, Tai C-L, Cai S (2002) An object-oriented progressive-simplification based vectorization system for engineering drawings: model, algorithm, and performance. *IEEE Trans Pattern Anal Mach Intell* 24(8)

- [11] Karima M, Sadhal K, McNeil T (1985) From paper drawings to computer-aided design. IEEE Comput Graph Applications
- [12] Liu L, Ouyang W, Wang X, Fieguth P, Chen J, Liu X, Pietikainen M (2018) Deep learning for generic object detection: a survey. arXiv preprint arXiv:1809.02165
- [13] Meng Z, Fan X, Chen X, Chen M, Tong Y (2017) Detecting small signs from large images. In: 2017 IEEE international conference on information reuse and integration (IRI). IEEE
- [14] Kasturi R, Bow S, El-Masri W, Shah J, Gattiker J, Mokate U (1990) A system for interpretation of line drawings. IEEE Trans Pattern Anal Mach Intell
- [15] Pan SJ, Yang Q et al (2010) A survey on transfer learning. IEEE Trans Knowl Data Eng 22(10):1345–1359
- [16] Cordella L, Vento M (2000) Symbol recognition in documents: a collection of techniques? Int J Docu Anal Recogn 3(2):73–88. <https://doi.org/10.1007/s100320000036>
- [17] Roth PM, Winter M (2008) Survey of appearance-based methods for object recognition. Inst. for Computer Graphics and Vision, Graz University of Technology, Austria, Technical report ICGTR0108 (ICG-TR-01/08)
- [18] Ruo-yu Y, Feng S, Tong L (2010) Research of the structural-learning-based symbol recognition mechanism for engineering drawings. In: 2010 6th international conference on digital content, multimedia technology and its applications (IDC)
- [19] Smith R (2007) An overview of the Tesseract OCR engine. In: Ninth international conference on document analysis and recognition (ICDAR 2007), vol 2. IEEE, pp 629–633
- [20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in CVPR, 2016,
- [21] Uijlings JR, Van De Sande KE, Gevers T, Smeulders AW (2013) Selective search for object recognition. Int J Comput Vis <https://doi.org/10.1109/ICDAR.2009.202>. <http://ieeexplore.ieee.org/document/5277721/>
- [22] G. Renton, P. Heroux, B. Gauzere, and S. Adam, “Graph neural network for symbol detection on document images,” in ICDARW, vol. 1,
- [23] Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.
- [24] Rosenblatt, F. (1958). The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, Article 6.

- [25] Sharma, S., Sharma, S., & Athaiya, A. (2020). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 4(12), 310-316.
- [26] Solawetz, J. (2020, June 29, 2020). YOLOv5 New Version - Improvements And Evaluation. <https://blog.roboflow.com/yolov5-improvements-and-evaluation>
- [27] C. Moreno-Garcia, E. Elyan, and C. Jayne, "New trends on digitization of complex engineering drawings." *Neural Comput. Applic.*, vol. 31, no. 6, pp. 1695–1712, 2019
- [28] A. Rezvanifar, M. Cote, and A. Albu, "Symbol spotting for architectural drawings: State-of-the-art and new industry-driven developments," *Trans. Comput. Vis. Appl.*, vol. 11, no. 1, p. 2, 2019. 1
- [29] Lu, Z.: Detection of text regions from digital engineering drawings. *IEEE Trans. Pattern Anal. Mach. Intell.*
- [30] He, S., Abe, N.: A clustering-based approach to the separation of text strings from mixed text/graphics documents. *Proc. - Int. Conf. Pattern Recognition*
- [31] Fan, K.C., Liu, C.H., Wang, Y.K.: Segmentation and classification of mixed text/graphics/image documents. *Pattern Recognition. Lett.*
- [32] M. Shaw, D. Garlan: *Software architecture: perspective son an emerging discipline*. Prentice Hall, Englewood Cliffs
- [33] K. Tombre, C. Ah-Soon, P. Dosch, G. Masini, S. Tabbone: Stable and robust vectorization: how to make the right choices. In: *Proc. 3rd Int. Workshop on Graphics Recognition*, Jaipur (India),
- [34] E. Valveny, E. Marti: Application of deformable template matching to symbol recognition in hand-written architectural drawings. In: *Proc. 5th Int. Conf. on Document Analysis and Recognition*, Bangalore
- [35] K. Wall, P. Danielsson: A fast sequential method for polygonal approximation of digitized curves
- [36] Lewis, R., Sequin, C. Generation of 3D building models from 2D architectural plans. *Computer-aided Design*
- [37] Ah-soon, C. A constraint network for symbol detection in architectural drawings. In: Tombre, K., Chhabra, A.K. (Eds.), *Graphics Recognition*. Springer, Berlin.
- [38] Solawetz, J. (2023, January 25). What is Yolov8? the ultimate guide. Roboflow Blog. Retrieved March 4, 2023, from <https://blog.roboflow.com/whats-new-in-yolov8/>
- [39] Rath, S. (2023, January 23). Yolov8 ultralytics: State-of-the-art yolo models. LearnOpenCV. Retrieved March 4, 2023, from <https://learnopencv.com/ultralytics-yolov8/>