



# Automatic Detection of Elements in the Technical Drawings of Bridges by Deep Learning and Parametric Modeling

Scientific work to obtain the degree

**Master of Science (M.Sc.)**

at the Department of Civil, Geo and Environmental Engineering of the Technical University of Munich.

**Supervised by** Prof. Dr.-Ing. André Borrmann  
M. Saeed Mafipour, M.Sc.  
Andrea Carrara, M.Sc.  
Chair of Computational Modeling and Simulation

**Submitted by** Daniyal Ahmed [REDACTED]  
[REDACTED]  
[REDACTED]  
[REDACTED]

**Submitted on** 06. June 2022

## Abstract

Bridges, as essential structures of infrastructure of any country, require timely and efficient maintenance practices to ensure their longevity, structural durability, and safe functionality. Current bridge monitoring and inspection procedures are found to be ineffective, labor-intensive, and time-consuming in managing a large inventory of the bridges. Bridge owners and local infrastructure authorities often rely on traditional Bridge Management System (BMS) to maintain their bridges. However, traditional BMSs do not provide up-to-date information on the structural condition of the bridges, often resulting in high costs for the bridge owners as bridge maintenance practices are then characterized by reactive rather than predictive procedures. Therefore, emphasizing on Building Information Modeling (BIM) technology to effectively maintain and monitor bridges is often recommended. In the domain of BIM, a Digital Twin (DT) of a bridge is defined as a virtual representation of an existing bridge which continuously provides updated and useful information. While utilizing DT in maintenance of bridges provides numerous advantages to bridge owners, the DTs are not available for many of the existing bridges that were built several decades ago. On the other hand, most of the bridge owners have bridge legacy data in the form of 2D technical drawings. Creating DTs of bridges is a manual and time-consuming process. Therefore, 2D technical drawings of existing bridges could play a role in generating DTs of the bridges. Therefore, in this thesis, the task of utilizing 2D technical drawings in extracting useful information has been tackled with a particular focus on object detection of bridge elements in technical drawings by deep learning and parametric modeling techniques. You Only Look Once (YOLO), a deep learning model based on Convolutional Neural Network (CNN), has been trained on technical drawings of bridges which contained 1142 instances for a total of 14 different classes. The model was trained several times to arrive at optimum combination of hyperparameters. Once the well-performing hyperparameters were selected, the model was trained for even a large number of iterations until the mean Average Precision (mAP) and average loss values converge. It was concluded that YOLO object detector has provided reasonable performance in predicting detections on validation and test dataset with an mAP of 89.15%.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Bridges as Crucial Structures . . . . .	1
1.2	Causes of Structural Deterioration in Bridges . . . . .	1
1.3	A large Bridge Inventory: A boon or a bane? . . . . .	2
1.4	Bridge Maintenance and Inspection practices: A look into Standards and Guidelines . . . . .	3
1.5	Bridge Management System (BMS): Can we do better? . . . . .	7
1.5.1	What is a Bridge Management System? . . . . .	7
1.5.2	Shortcomings in Current Bridge Management Systems and a Brief Introduction to Building Information Modeling (BIM) and Digital Twin (DT) . . . . .	8
1.6	Implementation of Digital Twinning in AEC Industry: A look into Facts and Figures . . . . .	10
1.7	Problem Statement . . . . .	12
<b>2</b>	<b>Theoretical Background</b>	<b>13</b>
2.1	Building Information Modeling (BIM) . . . . .	13
2.2	Digital Twin (DT) . . . . .	14
2.3	Computer Vision . . . . .	17
2.4	AI, Machine Learning, and Deep Learning . . . . .	21
2.4.1	Artificial Intelligence . . . . .	21
2.4.2	Machine Learning . . . . .	21
2.4.3	Deep Learning . . . . .	22
2.5	Neural Networks . . . . .	23
2.5.1	Artificial Neural Networks (ANN) . . . . .	23
2.5.2	Convolutional Neural Networks (CNN) . . . . .	24
<b>3</b>	<b>Related Research</b>	<b>28</b>
3.1	Image processing . . . . .	28
3.2	Deep Learning . . . . .	30
<b>4</b>	<b>Research Gaps &amp; Objectives</b>	<b>36</b>
4.1	Research Gaps . . . . .	36
4.2	Research Objectives . . . . .	37
<b>5</b>	<b>Object Detection in 2D Technical Drawings</b>	<b>38</b>
5.1	Object detection . . . . .	38
5.2	YOLO . . . . .	39
5.3	Dataset . . . . .	41
5.3.1	Identifying Relevant Classes . . . . .	41

5.3.2	Preprocessing Data	47
5.3.3	Data Annotation	48
5.3.4	Counting Instances	51
5.4	Object Detection with YOLO: A Step-by-Step Guide	54
5.4.1	Training YOLOv4 on a Technical Drawings Dataset	54
5.4.2	Testing the trained YOLOv4 on Technical Drawings Dataset	63
5.5	Parametric Modeling and Data Augmentation	64
5.6	Hyperparameters	73
<b>6</b>	<b>Results</b>	<b>74</b>
6.1	Case 1: 416_416_32_16	74
6.2	Case 3: 608_608_32_16	74
6.2.1	Converting Predicted Bounding Boxes Coordinates into YOLO format	77
6.2.2	Scaling-up Predicted Bounding Boxes	80
6.2.3	Scene Text Detection	89
<b>7</b>	<b>Conclusion &amp; Future Works</b>	<b>93</b>
7.1	Findings and Conclusions	93
7.2	Future works	95
	<b>Bibliography</b>	<b>98</b>

# List of Figures

1.1	The structural condition, "good" or "fair", for the bridges across the United States between 2008 and 2019 (ASCE, 2021). For details on the description of structural ratings, please refer to table 1.2 . . . . .	3
1.2	Distribution of condition index of bridges across Germany, (HAARDT, 2008)	5
1.3	Recommended bridge actions to be taken based on National Bridge Inventory (NBI) ratings, adapted from (FEDERAL HIGHWAY ADMINISTRATION (FHWA), 2018). For details on the description of structural ratings, please refer to table 1.2 . . . . .	5
1.4	Basic modules in a bridge management system BMS (OMAR & NEHDI, 2018)	8
1.5	Percentage of Total Stock of Bridges in four Developed Countries of the world: Germany, Netherlands, USA, and Japan over the years 1925-2004, (REITSEMA et al., 2020) . . . . .	9
1.6	Percentage of usage of DT in 50% or more transport related infrastructure projects by countries, (JONES & LAQUIDARA-CARR, 2017) . . . . .	11
1.7	Percentage of users who author create their own models compared with the percentage of users who use models produced by others, (JONES & LAQUIDARA-CARR, 2017) . . . . .	11
2.1	BIM Life cycle, (FUGAS, 2019) . . . . .	13
2.2	Utilization of BIM based technology in different stages of a construction project, (BOTH, 2012) . . . . .	15
2.3	Data flow in a digital model, (KRITZINGER et al., 2018) . . . . .	16
2.4	Data flow in a digital shadow, (KRITZINGER et al., 2018) . . . . .	16
2.5	Data flow in a digital twin, (KRITZINGER et al., 2018) . . . . .	17
2.6	An example of object detection with YOLO algorithm, (MIHAJLOVIC, 2019) .	18
2.7	An example of semantic segmentation for an image showing street view and all the objects of interest segmented and assigned different colors according to their classes, (DWIVEDI, 2020) . . . . .	19
2.8	An example of instance segmentation for an image showing street view and all the instances of a particular class (for instance, pedestrian or car) assigned unique segments or colors. This is when two people or cars moving at different speeds or in different directions, particularly applicable in autonomous driving problem, (PULKIT, 2019) . . . . .	20
2.9	A summary of computer vision tasks (a) semantic segmentation, (b) image classification with localization, (c) object detection, and (d) instance segmentation, (LI et al., 2017) . . . . .	20
2.10	Venn diagram displaying fields of Artificial Intelligence (AI), machine learning, and deep learning with respect to each other. Deep learning is a subset of machine learning which is in turn the subset of Artificial Intelligence (AI) .	21

2.11	A general architecture of a simple neural network displaying the input layer, the hidden layers, as well as the output layer, (SHUKLA, 2019) . . . . .	24
2.12	A convolution operation between a local receptive field of an image and the filter matrix (stride=2) to create a feature map (output), adapted from (REYNOLDS, 2019) . . . . .	26
2.13	A convolution operation between a local receptive field of an image and the vertical edge detector filter matrix (stride=2),(REYNOLDS, 2019) . . . . .	26
2.14	An example of ReLU activation function being applied to each of the output of neurons and then a max pooling of fxf where f=2 and stride =2 , adapted from (REYNOLDS, 2019) . . . . .	27
2.15	A typical CNN architecture showing convolution and ReLU hidden layers followed by pooling and so on. The network architecture can be divided into two categories: feature extraction (hidden layers) and classification, adapted from (MATHWORKS, n.d.) . . . . .	27
3.1	Creation of 3D models from 2D floor plans, (GIMENEZ et al., 2015) . . . . .	28
3.2	Fundamental steps in creation of 3D models from 2D scanned floor plans: (a) an original 2D scanned floor plan, (b) after denoising and text extraction, (c) graphical symbol recognition (such as doors, windows) and 2D geometry creation, and finally, (d) 3D model extrusion, (YIN et al., 2008) . . . . .	30
3.3	Image tiling process as described by Dosch and his colleagues, (DOSCH et al., 2000) . . . . .	31
3.4	An example of high-level element detection in scanned process and instrumentation (P&I) diagrams, (NURMINEN et al., 2019) . . . . .	31
3.5	A CNN architecture for symbol detection on process and instrumentation (P&I) diagrams, (MANI et al., 2020) . . . . .	32
3.6	An example of source symbol detected (red), connected symbols detected (green), and connection detected between symbols (blue) through lines traversed by depth-first search (graph-based approach), (MANI et al., 2020)	33
3.7	An example of symbol and text detection, (MANI et al., 2020) . . . . .	33
3.8	Object detection and similarity assessment framework as proposed by (VAN DAELE et al., 2019) . . . . .	34
3.9	CNN architecture ResNet-50 used for binary classification of 2d drawings. If a pair of drawings is similar then assign it to a class 'same', other 'different' (VAN DAELE et al., 2019) . . . . .	35
4.1	An example of a 2D architectural floor plan (left), (YIN et al., 2008) and a 2D technical drawing of a bridge (right) . . . . .	36
5.1	An example of a technical drawing showing different views of abutment including wing wall and retaining wall . . . . .	38
5.2	Intersection over Union (IOU) . . . . .	40
5.3	Several overlapping bounding boxes, (REDMON et al., 2016) . . . . .	40
5.4	An example of class deck_t_shaped_cross_section (class ID = 0) . . . . .	42

5.5	An example of class <code>deck_beam_shaped_cross_section</code> (class ID = 1)	42
5.6	An example of class <code>deck_plain_cross_section</code> (class ID = 2)	42
5.7	An example of class <code>deck_top_view</code> (class ID = 3)	43
5.8	An example of class <code>deck_t_shaped_cross_section</code> (class ID = 4)	43
5.9	Examples of class <code>abutment_retaining_wall</code> (class ID = 5). This class also includes the cross-section of the wing-wall since it is actually a part of the retaining wall, and looks very similar	43
5.10	An example of class <code>abutment_retaining_wall_cross_section</code> (class ID = 6)	44
5.11	An example of class <code>abutment_top_view</code> (class ID = 7)	44
5.12	An example of class <code>bridge_top_view</code> (class ID = 8)	44
5.13	An example of class <code>bridge_side_view</code> (class ID = 9)	45
5.14	An example of class <code>foundation_top_view</code> (class ID = 10)	45
5.15	An example of class <code>foundation_side_view</code> (class ID = 11)	45
5.16	An example of class <code>frame</code> (class ID = 12)	46
5.17	An example of class <code>table</code> (class ID = 13)	46
5.18	Graphical user interface of Labelling tool with one of the technical drawings of a bridge <code>bw003_03.JPG</code> opened, and also showing some of the relevant features in the tool	49
5.19	A screenshot of a dataset folder showing data labels in <code>.txt</code> files along with their images in <code>.JPG</code> format	49
5.20	The bounding box information stored in a <code>.txt</code> file for a particular technical drawing	50
5.21	A sample image of bridge <code>bw003</code> from the technical drawings dataset to show bounding box coordinates and image size in pixels	51
5.22	A snippet of the output console where the overall statistics of all of the 15 bridges have been computed	53
5.23	A snippet of the output console where the statistics of <code>bw056</code> bridge have been computed	53
5.24	A snippet of the output console where the statistics of <code>bw003</code> bridge have been computed	54
5.25	A bar chart displaying the total number of instances for each of the classes for a total of 15 bridges	54
5.26	Enabling GPU acceleration in Google Colab Pro	55
5.27	Modifications in <code>detector.c</code> file for iterations to be saved at every 500th iteration	56
5.28	The contents of a <code>backup</code> folder after training	58
5.29	The contents of a <code>YoloV4</code> folder before training	58
5.30	The names of classes in <code>obj.names</code> file	59
5.31	Content of a <code>obj.data</code> file	59
5.32	Content of a <code>yoloV4-obj.cfg</code> file	61
5.33	Content of a <code>train.txt</code> (left) and <code>test.txt</code> (right) files. Please, note only few lines are shown here	62

5.34	Content of a <code>image.txt</code> file for running detections on multiple test images at once.	63
5.35	Parametric Modeling tab in AutoCAD	64
5.36	An example of parametric model of class "frame" with geometric and dimensional constraints	65
5.37	An example of parametric model of class "frame". It can be noticed that the dimensions have been changed easily to create another unique instance of the same class. Some of the changes are shown in the circles	65
5.38	To convert dynamic dimensions into annotated dimensions for showing dimensions on the synthetic technical drawings	66
5.39	An example of parametric model of class "deck_t_shaped_cross_section" with geometric and dimensional constraints	66
5.40	An example of parametric model of class "deck_t_shaped_cross_section" showing modifications performed to create another instance easily of the same class	67
5.41	An example of parametric model of class "bridge_side_view" with geometric and dimensional constraints	67
5.42	An example of parametric model of class "bridge_side_view" with geometric constraints	67
5.43	An example of parametric model of class "bridge_side_view" with geometric constraints	68
5.44	Parametric models of all of the insufficient classes are shown in this diagram with geometrical and dimensional constraints hidden	68
5.45	An example of annotated synthetic 2D technical drawing created using parametric modeling in the Labellmg environment	69
5.46	An example of synthetic 2D technical drawing with labeled bounding boxes in Labellmg data annotating tool	70
5.47	Total number of synthetic instances for each of the classes as well as overall total	71
5.48	After adding the synthetic data to the original dataset, the total number of instances for all of the 14 classes equals to 1213 instances	71
5.49	Revisiting the bar chart (also included earlier as figure 5.25 in section 5.3.4) displaying the total number of instances for each of the classes for a total of 15 bridges	72
5.50	A bar chart showing the number of instances of each of the classes after labeling synthetic data from parametric modeling	72
6.1	A loss and convergence diagram with respect to number of iterations	75
6.2	A loss and convergence diagram with respect to number of iterations	76
6.3	A console output showing the average precision values for each of the classes as well as the mean average precision (mAP) for the best weights received for the highlighted hyperparameters in table 6.2	77

6.4	The content of <code>bbcoordinates.txt</code> file with predicted bounding box coordinates (yellow rectangles) and name of validation image shown in blue circle . . . . .	78
6.5	The content of "resized_valid_results" folder after running the test detector on validation images . . . . .	78
6.6	Object Detection results on a bridge technical drawing named as BW_43_21	81
6.7	Object Detection results on a bridge technical drawing named as BW_51_23, showing the classes "foundation_top_view" and "foundation_side_view" identified . . . . .	81
6.8	Object Detection results on a bridge technical drawing named as BW012-1_3, showing the classes "abutment_top_view", "abutment_retaining_wall_cross_section", "abutment_wing_wall", "abutment_retaining_wall", and "table" identified . . . . .	82
6.9	Object Detection results on a bridge technical drawing named as BW067-1_17, showing the classes "abutment_top_view", "abutment_retaining_wall_cross_section", "abutment_wing_wall", "abutment_retaining_wall", "deck_top_view", and "table" identified . . . . .	82
6.10	Object Detection results on a bridge technical drawing named as BW_56_30, showing the classes "deck_top_view", "abutment_retaining_wall", and "frame" identified . . . . .	83
6.11	Object Detection results on a bridge technical drawing named as BW_56_30, showing the classes "deck_top_view", "deck_tshaped_cross_section", and "table" identified . . . . .	83
6.12	The folders structure for subsequent postprocessing steps . . . . .	84
6.13	The content of output folder . . . . .	85
6.14	The original sized images corresponding to the resized image in figure 6.6 . . . . .	86
6.15	The original sized images corresponding to the resized image in figure 6.7 . . . . .	86
6.16	The content of output folder . . . . .	87
6.17	The content of crops folder . . . . .	87
6.18	The content of <code>scenetext</code> folder after running Character-Region Awareness for Text Detection (CRAFT) . . . . .	90
6.19	The content of the subfolder of a cropped region after running Character-Region Awareness for Text Detection (CRAFT) displaying further the cropped regions extracted from bounding boxes around the textual information . . . . .	90
6.20	Scene text detection performed on cropped image showing the class "abutment_wing_wall" . . . . .	91
6.21	Scene text detection performed on cropped image showing the class "abutment_top_view" . . . . .	92
7.1	A technical drawing in Labelling tool showing reinforcement details . . . . .	96
7.2	A console output showing the best mAP value and iteration number . . . . .	96

# List of Tables

1.1	Description of bridge condition rating in Germany, (HEARN, 2007) . . . . .	4
1.2	Description of NBI ratings for bridge condition, Federal Highway Administration (FHWA) (FHWA, 1995) . . . . .	6
5.1	Identified classes for object detection in technical drawing of bridges . . . . .	41
5.2	Different combinations of hyperparameters tested in this thesis . . . . .	73
6.1	Hyperparameters for the first case have been highlighted in the table below	74
6.2	Hyperparameters for the second case have been highlighted in the table below . . . . .	75
7.1	Evaluation parameters for third_run case . . . . .	95



# List of Algorithms

5.1	The script below resizes images in the dataset while maintaining the aspect ratios . . . . .	47
5.2	The script below counts the number of instances for each of the classes after data annotation process . . . . .	51
5.3	clone, build, make <code>darknet</code> from AlexeyAB github repository . . . . .	56
5.4	Helper functions to display object detection results on images within Colab notebook . . . . .	56
5.5	Mounting Google Drive into Cloud VM . . . . .	57
5.6	Splitting dataset into training and validation data . . . . .	59
5.7	Moving custom dataset and files from Google Drive to <code>darknet</code> folder . . . . .	61
5.8	This command starts the training for custom object detection . . . . .	62
5.9	Modifications in <code>yolo4-obj.cfg</code> file for testing the detector . . . . .	63
5.10	Execute the following code to run detections on multiple images and save the results of predicted bounding boxes in a <code>.txt</code> file . . . . .	63
6.1	Execute the following code to run detections on multiple images and display the predicted bounding boxes on technical drawings in the console output . . . . .	75
6.2	Execute the following code to run detections on multiple images and display the coordinates of predicted bounding boxes in the console as well as export them to a <code>.txt</code> file . . . . .	76
6.3	Execute the following code to convert predicted bounding boxes coordinates into YOLO format . . . . .	77
6.4	Execute the following code to scale-up the bounding boxes to be displayed on the original sized images and then crop out the regions enclosed by the predicted bounding boxes . . . . .	85
6.5	Execute the following code to performed scene text detection using Character-Region Awareness for Text Detection (CRAFT) . . . . .	90

# Acronyms

<b>AASHTO</b>	American Association of State Highway and Transportation Officials
<b>AEC</b>	Architecture Engineering and Construction
<b>AI</b>	Artificial Intelligence
<b>ANN</b>	Artificial Neural Network
<b>ASCE</b>	American Society of Civil Engineers
<b>BIM</b>	Building Information Modeling
<b>BMS</b>	Bridge Management System
<b>CAD</b>	Computer-aided Design
<b>CNN</b>	Convolutional Neural Network
<b>COCO</b>	Common Objects in Context
<b>CRAFT</b>	Character-Region Awareness for Text Detection
<b>DIN</b>	Deutsches Institut für Normung
<b>DT</b>	Digital Twin
<b>DWG</b>	AutoCAD Drawing Files
<b>DXF</b>	Data Exchange Format
<b>EAST</b>	Efficient and Accurate Scene Text Detector
<b>FHWA</b>	Federal Highway Administration
<b>GCRs</b>	General Condition Ratings
<b>ICT</b>	Information and Communication Technology
<b>IOU</b>	Intersection Over Union
<b>IoU</b>	Intersection over Union
<b>LMI</b>	locally mounted instrument
<b>mAP</b>	mean Average Precision
<b>MBE</b>	The Manual for Bridge Evaluation
<b>MR&amp;R</b>	Maintenance Repair And Rehabilitation
<b>NASA</b>	National Aeronautics and Space Administration
<b>NBI</b>	National Bridge Inventory
<b>NBIS</b>	National Bridge Inspection Standards
<b>NDT</b>	Non-Destructive Testing
<b>O&amp;M</b>	Operation and Maintenance
<b>OCR</b>	Optical Character Recognition
<b>P&amp;I</b>	process and instrumentation
<b>PCD</b>	Point Cloud Data
<b>ReLU</b>	Rectified Linear Unit
<b>RPN</b>	Region Proposal Network
<b>UAVs</b>	Unmanned Aerial Vehicles
<b>YOLO</b>	You Only Look Once

# Chapter 1

## Introduction

*This chapter begins with the importance of bridges to a society and economy as a whole, and then continues to highlight some of the causes of their structural degradation in section 1.2; therefore, requiring regular inspection and maintenance throughout its entire functional life to ensure their safe operations. Moreover, the issue of aging bridges in developed countries and excessive costs involved in their repair and restoration have been emphasized in section 1.3. Section 1.4 elaborates on the standards and guidelines used by different infrastructure authorities to measure, collect, record, and utilize bridge inspection data. Section 1.5 illustrates how BMS serves as a decision-making tool for the bridge owners and also highlights the shortcomings of traditional bridge management practices. Furthermore, section 1.5 also outlines utilizing DT models in the domain of BIM-based methodologies will yield substantial benefits to the bridge managers and decision-makers. Section 1.6 summarizes the trends in transportation infrastructure sector towards implementation of DT. To conclude this introductory chapter, section 1.7 introduces the problem statement of this thesis.*

### 1.1 Bridges as Crucial Structures

Bridges, as major links between nation's transportation network, play a key role in deciding how people and goods move from one point to another; hence, ensuring their long-term safety and performance is one of the most important tasks of infrastructure development sector of any country. Being an integral part of large-scale infrastructure of a country, bridges provide delivery of goods and services that are essential in supporting economic growth and social development of a region (GIL & BECKMAN, 2009).

### 1.2 Causes of Structural Deterioration in Bridges

Over time, the structural condition of a bridge deteriorates due to physical, material, and environmental factors. KHATAMI and SHAFEI (2021) investigated that several environmental factors, such as relative humidity, extreme temperatures, and excessive precipitation have detrimental effects on the strength and durability of a bridge, resulting in chloride penetration process which leads to corrosion initiation and cracks formation. In addition to weather conditions, KIM and YOON (2010) explored that other factors, such as the age of a bridge, the volume of traffic flow, and the type of structural system are also responsible for structural degradation of a bridge. Moreover, both group of authors have stressed upon the efficiency of bridge maintenance and repair actions to lower the costs involved in

bridge monitoring and rehabilitation. Considering adverse environmental factors and long duration of bridge inspection process to predict structural deficiency and longevity, there is a pressing need to establish an effective and well-integrated [BMS](#) to ensure public safety.

### **1.3 A large Bridge Inventory: A boon or a bane?**

Although bridges play an indispensable role in connecting communities, promoting economic and social development, and delivering goods and services, countries now face a challenge of accumulation of large inventory of old bridges. Bridges are critical structures in the infrastructure system of any country, and they require regular and timely maintenance procedures to ensure its safe functioning. A significant proportion of the bridges that are currently present in the regions of developed countries were built several decades ago; therefore, they require regular monitoring and inspection to maintain their structural integrity and serviceability. Most of the developed countries have a large inventory of bridges that is subjected to ageing and degradation; therefore a significant amount of time and investment is needed to rehabilitate the existing bridges. This is also one of the most concerning issues for public and private interests of the developed nations (MCCLURE & DANIELL, 2010).

The average operational life of large-scale infrastructure facilities, such as airports and bridges is more than 40-50 years (GIL & BECKMAN, 2009). This does not necessarily imply that bridges with average age of more than 50 are functionally obsolete, but this does suggest that developed countries face infrastructure restoration crisis since most of the bridges were erected during the infrastructure boom between 1960s and 1980s (REITSEMA et al., 2020). Figure 1.5 from REITSEMA et al. (2020) clearly shows a percentage increase in bridge inventory, during the construction boom between 1960s and 1980s, in the Netherlands, Germany, Japan, and the USA. This indicates that, as time passes, more and more bridges would require repair activities causing enforcement of weight limitations on the bridges and disruption in traffic flows (KIM & YOON, 2010), posing detrimental effects on the economy of a region (REITSEMA et al., 2020). According to a report by CEBR (2014), the direct and indirect costs related to traffic congestion is expected to rise from \$ 33.4 million in 2013 to \$ 43.8 million in 2030; however, when combining the figures with those of other three developed countries (France, UK, USA), the total costs are expected to increase from \$200.7 billion in 2013 to a \$293.1 billion by 2030. Therefore, there is an increasing demand for optimization of bridge management practices by local transportation sector to minimize traffic hindrance and congestion.

According to infrastructure report of American Society of Civil Engineers ([ASCE](#)), 42% of America's more than 617,000 bridges are at least 50 years old (ASCE, 2021). ASCE (2021) report further highlights that among these bridges, currently, 7.5% are structurally deficient, and it requires an increase in the spending by 58% from the current budget allocation to \$22.7 billion per year in order to improve the condition depicted by figure 1.1, so that the rate of repair becomes greater than the rate of deterioration. Whereas,

## Bridge Conditions by Year



Figure 1.1: The structural condition, "good" or "fair", for the bridges across the United States between 2008 and 2019 (ASCE, 2021). For details on the description of structural ratings, please refer to table 1.2

the bridge stock in Germany consists of 38,288 bridges with current maintenance needs accumulating to €450 million annually (HAARDT, 2008).

In its report, ASCE (2021) highlighted the fact that in the US, owing to the low annual rate of just 0.1% in reduction of structurally deficient bridges over the past two years, the number of bridges with "fair" rating is now more than the ones with "good" rating. This is due to inherent problems associated with traditional practices in the US construction industry leading to inefficient rehabilitation and replacement actions, higher costs, and lower productivity (SACKS et al., 2018). As a consequence of the slow progress of bridge rehabilitation work in the US, the number of bridges with deteriorating conditions is exceeding the number of the ones that are structurally sound.

## 1.4 Bridge Maintenance and Inspection practices: A look into Standards and Guidelines

Different countries practice bridge monitoring and inspection procedures based on well formulated guidelines and standards. In Germany, for instance, the inspection of bridges is performed by certified civil engineers who serve as inspection program managers and team leaders, whereas inspectors are usually well-trained technicians (HEARN, 2007). Moreover, the inspection team perform inspections according to the German Standards DIN 1076 (DEUTSCHES INSTITUT FÜR NORMUNG (DIN), 1999) which serve as guidelines and classify inspections into different stages: main inspections, simple inspections, inspections on special occasions, inspections according to special regulations, and regular observations. The results of the inspections are collected and recorded according to the guidelines (RI-

EBW-PRÜF, 2003), and the data is then used to update German bridge data inventory, SIB-Bauwerke (Road Information Database – Structures) (SIB-BAUWERKE, 2013). After a thorough inspection, each of the deteriorated parts is then assigned a four-scale rating which is mentioned in detail in the following paragraph. In addition, table 1.1 shows detailed description of these ratings.

Federal or state authorities often assign evaluation criteria to the existing structural condition of a bridge. After a thorough condition assessment analysis of the bridges, their current structural conditions drive the necessity to assign different level of priorities in their maintenance (SPENCER et al., 2019). In Germany, for instance, a four-stage rating is assigned to a bridge after careful analysis by a certified bridge inspector. Visible damages, cracks, and other structural deterioration of a bridge are assessed with regards to their effects on traffic safety, stability, and durability (HAARDT, 2008). Considering these factors, a condition index is derived in a range of 1.0 to 4.0 where 1.0 being "very good" condition and 4.0 being "insufficient condition" (HAARDT, 2008). The results of condition analysis of bridges across Germany are summarized in figure 1.2. According to figure 1.2, over 70.8% of the bridges have a satisfactory condition index of above 2.0, whereas 6.0 % are those with critical or inadequate structural conditions (HAARDT, 2008). This suggests that there is a significant proportion of bridges in Germany that require maintenance actions to preserve their structural integrity, stability, and durability.

Table 1.1: Description of bridge condition rating in Germany, (HEARN, 2007)

Grade	Description
1.0-1.4	Very good structural condition Continue normal maintenance
1.5-1.9	Good structural condition, but may have less long-term durability Continue normal maintenance
2.0-2.4	Satisfactory structural condition, but may have less long-term durability Continue normal maintenance and consider a plan for repair
2.5-2.9	Unsatisfactory structural condition Traffic safety may be affected Structure is not sufficiently durable Continue normal maintenance and plan for repair Restrictions on traffic use or load may be needed
3.0-3.4	Critical structural condition Traffic safety is affected Structure is not durable Immediate repair is needed Restrictions on traffic use or load are needed
3.5-4.0	Inadequate structural condition Traffic safety is not adequate Structure is not durable Immediate repair or rehabilitation is needed Restrictions on traffic use or load are needed

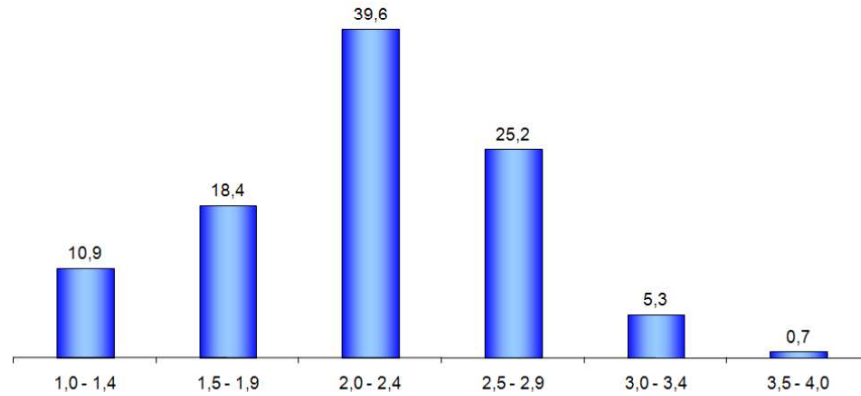


Figure 1.2: Distribution of condition index of bridges across Germany, (HAARDT, 2008)

In the US, The Manual for Bridge Evaluation (MBE) (AASHTO, 2018) provided by the American Association of State Highway and Transportation Officials (AASHTO) serves as a guideline for bridge owners to assess the condition of existing highway bridges based on standard inspection procedures that are also compliant with National Bridge Inspection Standards (NBIS) (FHWA, 2022b). Moreover, In the US, Federal Highway Administration (FHWA) is the responsible authority to conduct safety inspections of existing bridges across the country, and upon judging the condition, regularly updates its database, National Bridge Inventory (NBI) (FHWA, 2022a) which includes all the bridges and tunnels across the US.

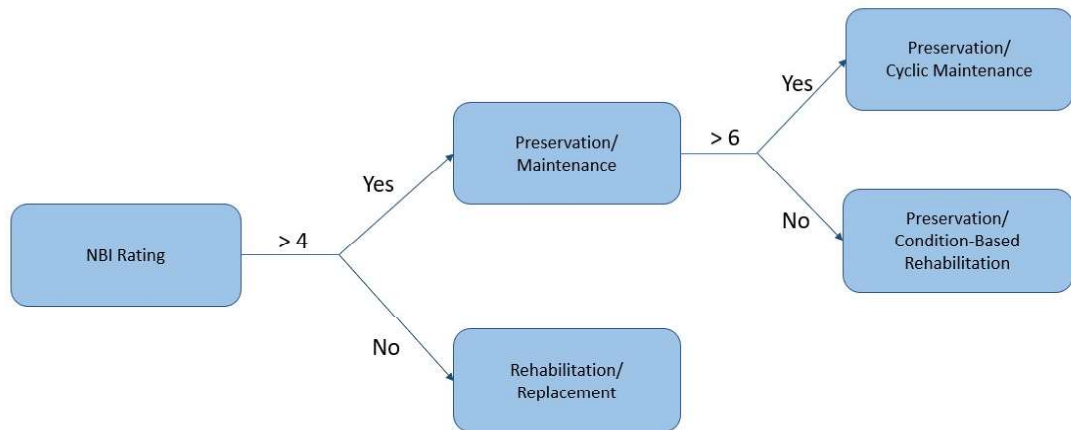


Figure 1.3: Recommended bridge actions to be taken based on NBI ratings, adapted from (FHWA, 2018). For details on the description of structural ratings, please refer to table 1.2

As mentioned earlier in section 1.3, there is an increasing backlog of existing bridges requiring repair or replacement actions. Infrastructure development authorities often

face the problem of proper allocation of available funds; thus, they need careful asset management practices to effectively deploy resources to the degraded structures. In the United States, for instance, **NBI** General Condition Ratings (**GCRs**) are used to compare and evaluate the structural condition of an existing bridge with respect to its as-built condition (**FHWA**, 2018). Federal authorities use these **NBI** ratings to prioritize investment for bridges (table 1.2), with regards to their structural condition, serviceability, and importance to public (**AASHTO**, 2008). These **NBI** ratings are based on annual or biennial visual inspection of bridges and ranges from 0 to 9 (**FHWA**, 2018). Detailed description of bridge condition ratings is provided in table 1.2 which is extracted from **FHWA** (**FHWA**, 1995).

Table 1.2: Description of **NBI** ratings for bridge condition, **FHWA** (**FHWA**, 1995)

Rating	Condition	Description
9	Excellent	New condition, no noteworthy deficiencies
8	Very good	No repair needed
7	Good	Some minor problems, minor maintenance needed
6	Satisfactory	Some minor deterioration, major maintenance needed
5	Fair	Minor section loss, cracking, spalling, or scouring for minor rehabilitation; minor rehabilitation needed
4	Poor	Advanced section loss, deterioration, spalling or scouring; major rehabilitation needed
3	Serious	Section loss, deterioration, spalling or scouring that have seriously affected the primary structural components
2	Critical	Advanced deterioration of primary structural elements for urgent rehabilitation; bridge maybe closed until corrective action is taken
1	Imminent failure	Major deterioration or loss of section; bridge may be closed to traffic, but corrective action can put it back to light service
0	Failed	Out of service and beyond corrective action

From table 1.2, an existing bridge is termed as "structurally deficient" if its condition rating is equal to or less than 4 for its bridge deck, superstructure, and substructure (**SRIKANTH & AROCKIASAMY**, 2020). Regular inspections are conducted to collect, maintain, and update the **NBI** database and corresponding ratings of the bridges. These ratings are then used to decide on recommended bridge management action (preservation/maintenance, rehabilitation, or replacement) so that federal funds can be allocated. Please, refer to figure 1.3 for an understanding on proper bridge actions according to **NBI** ratings. Taking limited availability of financial resources into consideration, responsible authorities have to ensure efficient allocation of resources so that maximum benefits could be achieved (**HAARDT**, 2008).

While **NBI GCRs** ratings assist bridge owners and decision-makers to allocate investments funds for management of their assets, these ratings are generally qualitative, and are too broad in deciding on a particular bridge maintenance and restoration activity (**FHWA**,



2018). This also means that a conservative bridge maintenance approach would result in excessive repairing costs due to unnecessary actions, whereas, negligence or delay in maintenance would lead to unavoidable disastrous consequences in the long run (RASHIDI et al., 2016). The location and type of the defects are not taken into account while devising these ratings which can have detrimental effects on the overall performance of a bridge in the long run. In such cases, the utilization of digital technology, such as generation of DT, which is a detailed information rich virtual representation of an existing physical structure, is often encouraged. Further details on DT are provided in section 2.2. Deploying DT in bridge management will not only allow decision-makers to optimize their bridge maintenance practices, but will also assist them in implementing correct reparation or restoration actions, while saving limited resources.

Considering the huge amount of data in NBI database, bridge owners and local infrastructure authorities require BIM based BMS for an efficient asset management and resource allocation practices. An efficient BMS does not only allow authorities to have a hierarchical view on the condition of the bridges, but it also enables them to prioritize investments to the bridges that need more attention with regards to their structural deficiency, importance, and functionality.

## 1.5 Bridge Management System (BMS): Can we do better?

### 1.5.1 What is a Bridge Management System?

BMS was briefly mentioned at the end of section 1.2 as a mean to manage bridges. Considering high costs involved in the maintenance of aging bridges, infrastructure development and maintenance authorities utilize BMS to manage local bridges throughout their entire life cycle. As limited funds are available, BMS assist bridge owners and local decision making authorities to maximize the safety, serviceability, and functionality of bridges (OMAR & NEHDI, 2018). Besides ensuring the safety of public traveling on the existing bridges, over the recent years, the scope of BMS has also been broadened to include the aim of maximizing the effect of limited maintenance funds to secure future investment in bridges (HURT & SCHROCK, 2016). Figure 1.4 depicts the general architecture of any BMS with its respective modules. A brief description of the functions of each of the modules is given below:

- **Inventory Module:** This is where the bridge inventory database is stored. This module is of paramount importance to any BMS, and most BMS are just restricted to this module (PREGNOLATO, 2019). All further actions and decisions of BMS depend on this module (SABACK DE FREITAS BELLO et al., 2021a).
- **Inspection Module:** This component of BMS is related to the inspection of damages and evaluates the existing structural condition of the bridges.



Figure 1.4: Basic modules in a bridge management system **BMS** (OMAR & NEHDI, 2018)

- **Maintenance Repair And Rehabilitation (MR&R):** This module assists in short and long term plans for intervention.
- **Optimisation Module:** This module combines all the previous modules to determine the most cost effective maintenance strategies.

### 1.5.2 Shortcomings in Current Bridge Management Systems and a Brief Introduction to Building Information Modeling (BIM) and Digital Twin (DT)

Data collection for a traditional **BMS** inventory is based on visual inspection by bridge inspectors who then record visible damages and cracks on key components of a bridge, such as girders, deck, slabs, and piers. This means that the data from visible inspection is highly subjective and primarily qualitative since it depends on the judgement and expertise of an individual inspector (OMAR & NEHDI, 2018; ZHU et al., 2010). Performing a condition assessment of a bridge is a labour-intensive, cumbersome, expensive, and sometimes a dangerous task (SPENCER et al., 2019; ZHU et al., 2010); hence, bridge managers should explore new and improved techniques for collection of high quality data, while simultaneously considering cost limitations involved in bridge inspection (HEARN et al., 2000). Given the challenges involved in visual inspection and the impossibility of early detection of invisible weakness in structure, innovative Non-Destructive Testing (**NDT**) techniques could be employed to detect the actual causes of damages (LEHMANN, 2020). This also helps in implementation of preventive measures, rather than reactive measures, which reduces the overall cost of maintaining bridges.

As visual inspection is a time intensive and a difficult task which results in increasing backlog of maintenance activities and traffic disruption, CHAN et al. (2015) explored in their research that Unmanned Aerial Vehicles (**UAVs**) could potentially be employed for bridge inspection procedures which will not only help in lowering the overall inspection costs, but also assist in reducing traffic disruption. However, the research has also highlighted several critical obstacles that might hinder implementation of **UAVs** in bridge inspection. They are: stability and control, accuracy of data collected, safety to public, and challenges in wider deployment of **UAVs**. While there are potential benefits in integrating innovative technological ideas, such as **UAVs**, in bridge condition assessment practices, several challenges still remain.

So far, several bridge data collection techniques for **BMS** have been explored, and it can be summarized that, while aforementioned techniques entail benefits in bridge inspection,

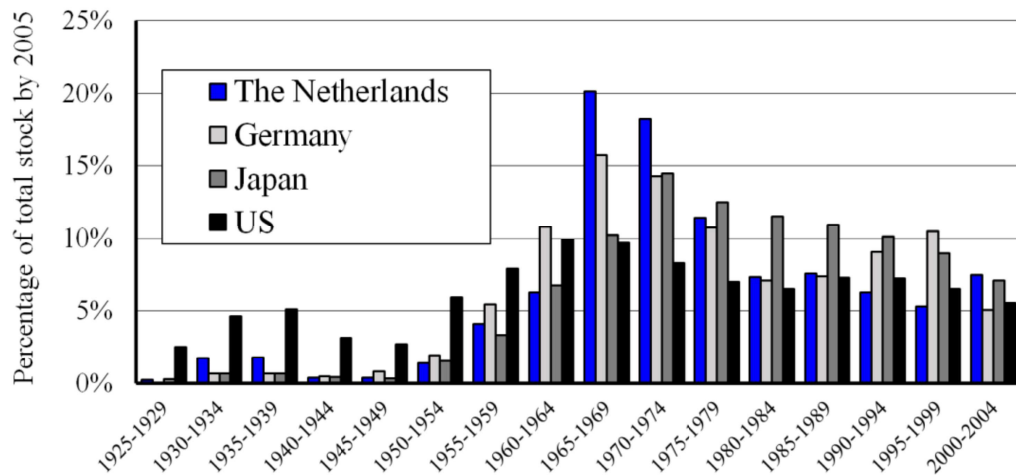


Figure 1.5: Percentage of Total Stock of Bridges in four Developed Countries of the world: Germany, Netherlands, USA, and Japan over the years 1925-2004, (REITSEMA et al., 2020)

challenges still remain as such techniques are time-consuming and cumbersome to implement. To exacerbate the problem, traditional methods of collecting and exchanging information in BMS do not provide an integrated and single unit for decision making. Data in BMS is usually stored in the form of loosely integrated print versions or text files (HOU et al., 2018; SHIM et al., 2017). As most of the information is conveyed in 2D (on paper, text files, or other electronic files), the bridge owners often find it challenging to interpret and relay the information to facility managers, and thus, this task is , time-consuming and prone to errors and omissions, resulting in delay in maintenance actions and an increase in subsequent costs (MIGILINSKAS et al., 2013; SACKS et al., 2018).

Sometimes, lack of version control and duplication of information is a common occurrence; thus, creating an information chaos. Moreover, the data do not meet the standard of information needed for reliable decision-making; thus, challenging the data interpretation abilities of decision makers to provide useful information. Therefore, efforts have been made in Architecture Engineering and Construction (AEC) industry to implement digital collaborative methods to reduce costs and increase efficiency in Operation and Maintenance (O&M) of the structures. In this domain, BIM, which is a digital description of a structure, has been demonstrated to have an immense value in enriching the bridge owners with useful information so that sound decisions are made for the maintenance of their assets.

Since most of the existing BMSs surveyed by MIRZAEI (2014) do not include geometric representation of bridges, ISAILOVIĆ et al. (2020) suggests that if BMSs are enriched with BIM representation of bridges, significant value will be added to the quality of existing BMSs. Furthermore, most of the existing BMSs do not fully integrate the data related to the structural condition of a bridge at the time of its completion; hence, it does not allow comparison between the condition upon completion and condition upon inspection. Integrating condition assessment data of a bridge at the time of its completion and

recording data generated from its inspections will assist authorities in predicting future structural degradation more accurately and planning maintenance activities efficiently.

In his research, HAARDT (2008) inspected that German BMS, includes bridge data with regards to existing damages based on visual inspection; hence, bridge maintenance activities involve a high cost as they are characterized by reactive procedures. Whereas, he proposed that an optimized life cycle oriented BMS, which is active throughout the entire functional life cycle of a structure, should be based on predictive and risk-based inspection procedures. This necessitates the need for an integrated and systematic bridge management system based on collaborative DT technology to improve the performance of bridge management and maintenance operations; thus, reducing consequential costs.

Therefore, emphasizing on BIM, a technology for modeling the bridge DT, as a possible solution to reduce the O&M related costs throughout the entire life cycle of a structure is often suggested. In the domain of BIM, a geometric DT is defined as a virtual information-rich 3D model of an existing bridge. The DT model could be generated and provided with all the required information based on extracted data from 2D technical drawings (as-designed DT), parametric modeling (as-designed DT, as-built DT, (SABACK DE FREITAS BELLO et al., 2021b)), inspection and condition assessment of the bridge (as-is DT) and bridge management systems (as-is DT). More details on BIM and DT are provided in the 2.1 and 2.2 respectively.

## 1.6 Implementation of Digital Twinning in AEC Industry: A look into Facts and Figures

Realizing the substantial benefits of transformation to digital technology, countries around the world are making efforts in DT generation of their infrastructure to have a better understanding and control over the management of their assets.

From figure 1.6, it can be seen that, during the span of two years (from 2015 to 2017), there has been an almost 50% increase in the usage of DT in more than 50% of the projects related to transportation infrastructure in advanced countries, such as France, Germany, the UK, and the US (JONES & LAQUIDARA-CARR, 2017). According to the 2017 report by Dodge Data and Analytics, among the heavy DT users surveyed, 83% reported an positive impacts on their projects from the use of it, and 73% claimed that they have not even realize half of the benefits that they believe could be received from implementing DT across their projects (JONES & LAQUIDARA-CARR, 2017). Heavy users of DT were defined as engineers and contractors who utilize BIM and DT on more than 50% of their infrastructure projects. This shows an increased confidence in innovative digital technologies among engineers and contractors in the transportation infrastructure sector.

Moreover, DT provides advantages to both category of users: the ones who create their own models, and the ones who use the models created by others. Figure 1.7 shows the

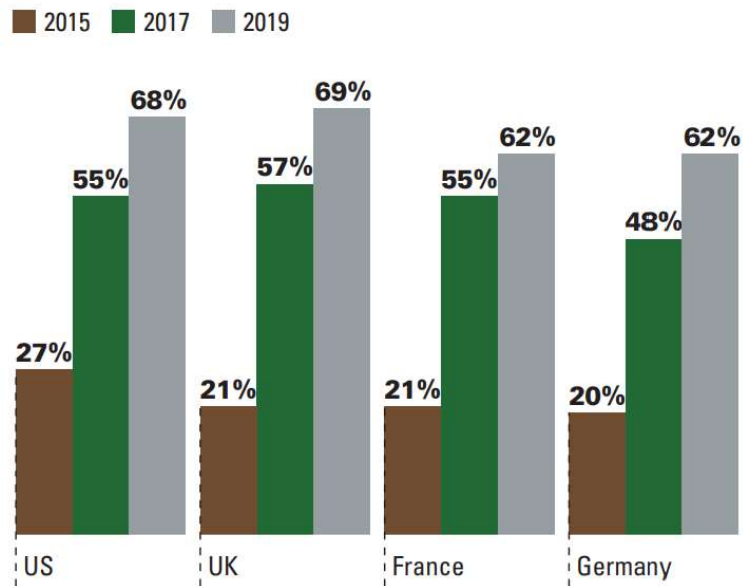


Figure 1.6: Percentage of usage of DT in 50% or more transport related infrastructure projects by countries, (JONES & LAQUIDARA-CARR, 2017)

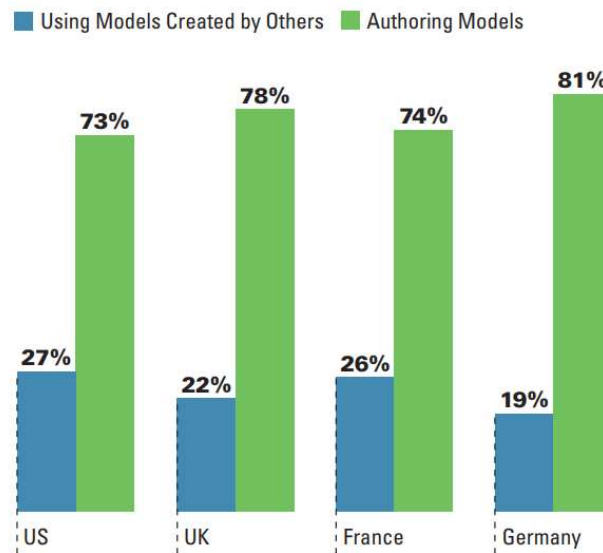


Figure 1.7: Percentage of users who author create their own models compared with the percentage of users who use models produced by others, (JONES & LAQUIDARA-CARR, 2017)

percentage of engineers and contractors who create their own DT compared with the percentage of users who use models created by others. It can be seen that 75% of the users do create their own models to realize full advantages of DT and gain competitive edge in the market with higher return on their capital investments (JONES & LAQUIDARA-CARR, 2017; SACKS et al., 2018). As DT users have a competitive edge in securing infrastructure projects, the companies that utilize DT created by others also experienced

an improvements in areas, such as interdisciplinary communication, timely release of deliverables, and better design of visualizations (JONES & LAQUIDARA-CARR, 2017).

## 1.7 Problem Statement

While utilizing DT in operation, maintenance, and inspection of the the bridges yields many benefits to the local authorities, the models are not readily available for most of the existing aging bridges that were built several decades ago (GIMENEZ et al., 2015). Despite its information rich features, DTs are not easy to generate and require extensive and time-consuming manual process. Also since, 2D technical drawings are present for existing bridges, in this thesis, the process of detecting bridge elements in technical drawings of the bridges has been explored and demonstrated. Therefore the problem statement can be defined as follows:

*"Automatic Detection of Elements in the Technical Drawings of Bridges by Deep Learning and Parametric Modeling"*

This problem statement can now be divided into parts to provide more details.

*"Automatic Detection of Elements..."*

Since the task of generating DTs is a time-consuming, error prone, and a manual process, efforts have been put into this thesis to automate or at least semi-automate the task of detecting bridge elements in technical drawings. This object detection task would then serve as a basis in DT generation process since all the relevant information regarding a particular bridge element would be extracted.

*...in the Technical Drawings of Bridges...*

In this thesis, the concept of utilizing 2D technical drawings of bridges to tackle DT generation process for existing bridges, has been used because the physical features of a bridge change gradually, and the updates in DT's features of such structures are not very frequent. Moreover, bridge owners possess bridge legacy data in the form of 2D technical drawings of the existing bridges; hence, such drawings could be utilized in extracting useful bridge information.

*...by Deep Learning...*

Deep learning models have seen an increase in popularity in many computer vision tasks, such as object detection, image segmentation, and so on. Particularly, deep learning models based on CNN, such as YOLO object detector, have delivered reasonable performances in several object detection related tasks. Hence, in this thesis, YOLO object detector model will be trained to detect bridge elements in technical drawings.

*...and Parametric Modeling"*

And finally, in this thesis, parametric modeling techniques in Computer-aided Design (CAD) have been utilized for data augmentation purposes to create synthetic technical drawings.



## Chapter 2

# Theoretical Background

*This chapter will assist the reader in reviewing the theoretical background and concepts to have a clear understanding on the terminologies, techniques, and processes used in thesis to automate the detection of elements in the 2D technical drawing of bridges by deep-learning and parametric modeling techniques.*

### 2.1 Building Information Modeling (BIM)

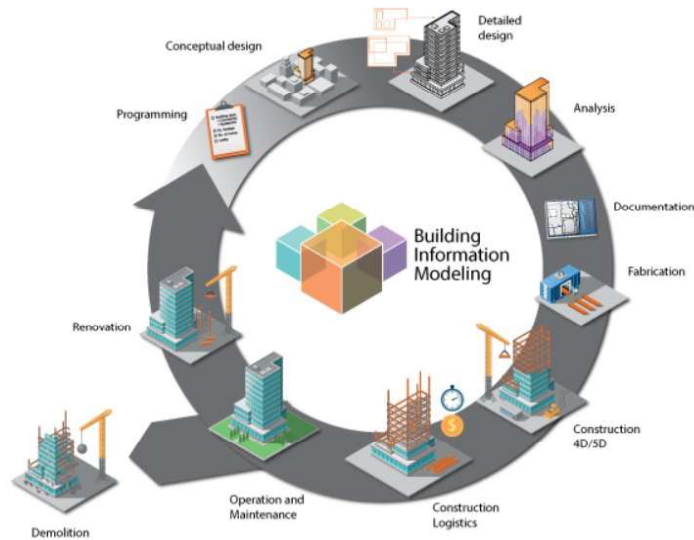


Figure 2.1: BIM Life cycle, (FUGAS, 2019)

Within recent years, **BIM** has gained tremendous popularity in **AEC** industry due to its digital description of a structure, its usefulness in simulation of infrastructure projects in a virtual environment, and its effectiveness in management of different construction projects. **BIM**, as an integrated and collaborative methodology, is centered on digital 3D-model, **DT**, which contains information needed to manage a structure and to optimize cooperation between stakeholders at various stages of structure's life cycle (ROCHA et al., 2020). With the advancements in Information and Communication Technology (**ICT**), there has been a major shift in the proliferation of **BIM** in industrial and academic sectors to manage project design and data in a digital format throughout entire life cycle of a structure (SUCCAR, 2009). Reaching beyond the scope of **CAD** and visualization, a **BIM** model does not only contain geometrical and topological information of a structure but also technical information as well as semantic data that can be queried and analysed,

allowing for collaborative multidisciplinary communication and management between all the stakeholders of a building (GIMENEZ et al., 2015).

As one of the most promising developments in AEC industry, BIM assists infrastructure owners and managers to effectively manage their projects to formulate maintenance activities within cost limitations. In their research, BRYDE et al. (2013) have underlined potential benefits of utilizing BIM technology, and have also analysed the performance of the construction projects that have used BIM throughout their entire project phases. Among the benefits explored, the most frequent gain was the reduction in overall cost and the ability to have a better management of an infrastructure project throughout its different design and development phases.

Realizing the benefits of integrating BIM in its AEC industry with respect to increased productivity, improved multidisciplinary communication, and reduced transaction costs, the German Federal Ministry of Transport and Digital Infrastructure (Bundesministerium für Verkehr und digitale Infrastruktur) has devised a comprehensive plan to outline digital transfer of its current infrastructure to BIM based technology (BORRMANN et al., 2020). Furthermore, the ministry requires the implementation of BIM in all new public infrastructure projects starting from 2020 (BORRMANN et al., 2020). According to a survey by BOTH (2012), in Germany, the usage of BIM is primarily limited to the planning phase of a building. From figure 2.2, it can be interpreted that a good proportion of surveyed engineers, building owners, and construction companies utilize BIM technology in early design and planning stages, whereas, only a few use it in inspection/management (facility management: 22.4%) stages of project life cycle. The survey also found a strong correlation between efficient management of infrastructure projects and the degree of application of BIM in different phases of a project.

## 2.2 Digital Twin (DT)

Initially utilized by National Aeronautics and Space Administration (NASA), the concept of DT was first born in the aerospace industry to replicate the development life cycle of aerial vehicles (NEGRI et al., 2017). A DT is a virtual representation of a physical system or a structure that spans throughout its entire life cycle (PARROTT & WARSHAW, 2017). Some of the key capabilities of DT are mentioned below:

- it allows bidirectional exchange of data from its physical counterpart to itself and vice versa.
- it updates from real-time data produced from sensors attached to its physical counterpart
- it leverages computational techniques and multi-physics simulations to optimize the performance, monitoring, and management of physical entities throughout their life cycle



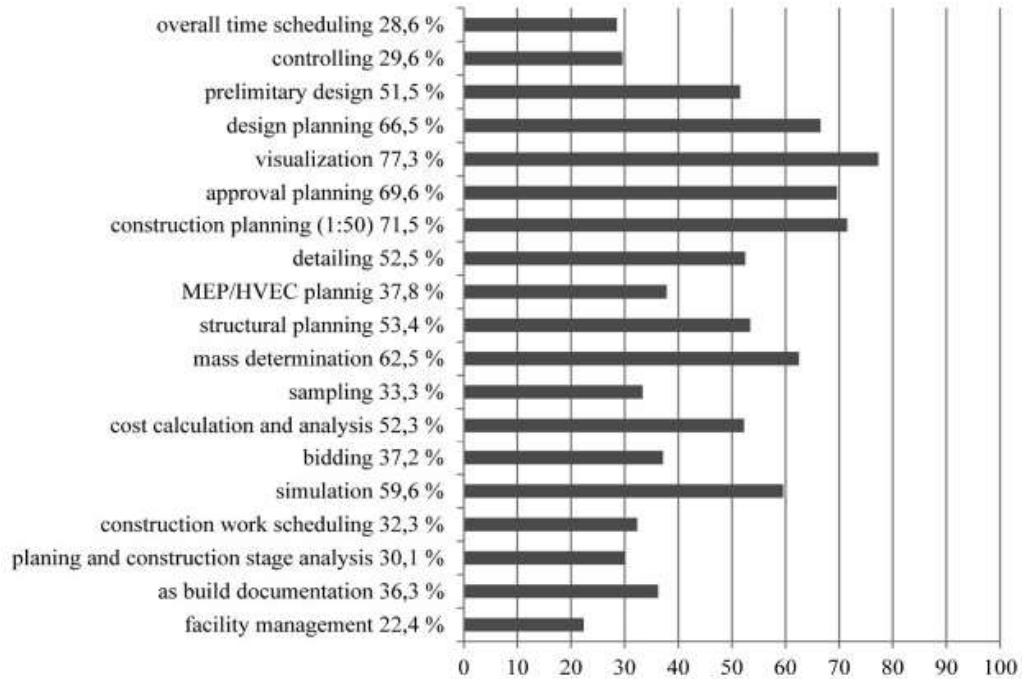


Figure 2.2: Utilization of BIM based technology in different stages of a construction project, (BOTH, 2012)

In the domain of BIM of a bridge, DT can be defined as a digital geometric-semantic model that provides an accurate replication of an existing physical bridges in an integrated common environment that serves a means of bringing all the stakeholders together for effective decision-making purposes (SACKS et al., 2018). As bridge owners face enormous challenges in collecting, structuring, and managing the data needed for bridge reparation or restoration works, traditional BMSs have been less effective in reducing the consequential costs associated with maintaining the inventory of bridges. Thus, DTs of existing bridges could possibly allow an improvement in bridge inspection, monitoring, and management. As DT can update itself using the real-time data from the physical environment via attached sensors, bridge inspection teams are able to now receive an early understanding on the structural degradation caused by adverse environmental affects, traffic loads, and aging (KOCH et al., 2014).

Although efforts have been made to devise a general definition of DT, there has been no consensus over the scope or features of a DT (CIMINO et al., 2019; KRITZINGER et al., 2018). Therefore, sometimes it is possible to confuse the DT with other simulation types. Often the digital replica of existing structure is modeled manually and is not connected with its physical counterpart for data exchange, on the other hand, there is a synchronized real time data exchange between physical and digital representation. KRITZINGER et al. (2018) proposed three broad categories of DT based on their data integration level between physical entity and digital counterpart to make a clear comparison between them.

- **Digital Model:** A digital model is a virtual representation of an existing physical structure or a system which does not use any form of automated data exchange

between physical system and its digital counterpart. Hence, a change in the properties of a physical system will have no direct effect on the digital representation and vice versa which means that the changes have to be updated manually between the two. Figure 2.3 depicts only the manual data flow between physical object and digital model.

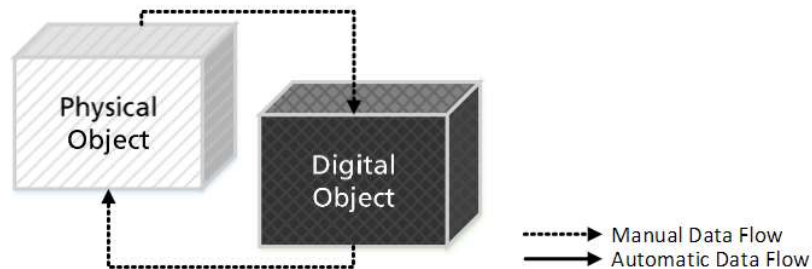


Figure 2.3: Data flow in a digital model, (KRITZINGER et al., 2018)

- **Digital Shadow:** This category inherits all the features of the digital model described above, in addition to the one-way automated data flow from physical object to its digital representation. When this automated uni-directional data flow from physical object to digital representation exists, it is called a digital shadow. Any change in the physical object will lead to a change in virtual representation and not in the opposite direction.

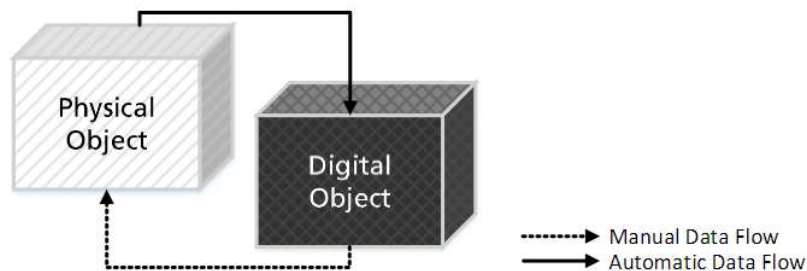


Figure 2.4: Data flow in a digital shadow, (KRITZINGER et al., 2018)

- **Digital Twin:** If a digital representation of physical system further inherits all the properties of a digital shadow, in addition to allowing bi-directional data exchange between physical structure and its digital counterpart, then it is called digital twin. In such a case, a change in the physical object will lead to a direct change in its digital representation and vice versa.

Thereafter, in this thesis, the author will refer to digital twin (DT) only and further explores on the types of DT in the domain of construction industry.

As bridge owners and decision-makers are facing an increasing burden of maintaining existing bridges, both industry as well as academics are searching for digital technology

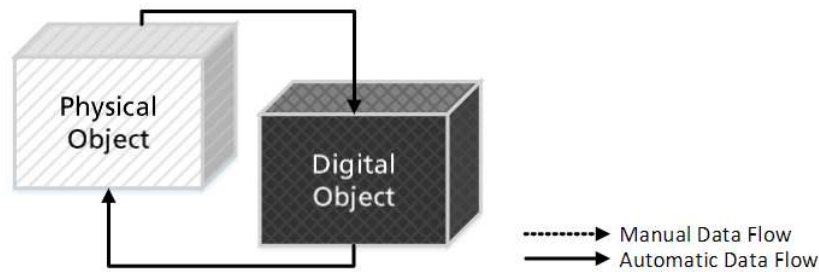


Figure 2.5: Data flow in a digital twin, (KRITZINGER et al., 2018)

integrated inspection solutions According to KOCH et al. (2014), **DT** can be defined at three different phases of structure's life cycle:

- **As-Designed:** This type of model contains detailed designed configuration of a product (such as performance requirements and functional requirements) before the start of construction, and is created by the design team based on the data generated by suppliers and sub-contractors.
- **As-Built:** This **DT** is generated after the completion of a construction project, and it reflects the actual state of the bridge with respect to its designed state. This model is produced by the general contractors.
- **As-is:** This model is the produced by bridge management agencies upon regular inspections and it could lead to improvements in **O&M** of the bridges. This virtual model replicates architectural, mechanical, and structural features of a bridge with a high degree of accuracy. If its capabilities are fully realised, then this model would allow decision-makers to predict future structure deterioration more accurately and take intervention measure timely.

Among the above categories, **DT** is utilized the highest during the design phase of any construction project (as-designed **DT**), followed by less utilization upon completion of the project (as-built **DT**), and almost no usage for maintenance stage in the construction industries of France, Germany, the UK, and the US (KOCH et al., 2014).

## 2.3 Computer Vision

Computer vision is an interdisciplinary scientific field that deals with automation of extracting and interpreting useful information from digital images or videos to extract high-dimensional data which can be beneficial in understanding the physical world to take action or make decision based on it (SPENCER et al., 2019). From an engineering point of view, computer vision aims to artificially imitate the tasks of human visual system in terms of performance or sometimes even surpass it (HUANG, 1996). The field of computer

vision is not new. Efforts have been made in 1960s to extract useful information from the images. For instance, in 1963, Larry Roberts at MIT made considerable efforts to extract object shape information from the 2D perspective view of blocks using edges and other geometrical primitives (ROBERTS, 1963). In recent years, several applications of computer vision have started to emerge including license plate number recognition, facial recognition, pedestrian detection, and vehicle detection to name a few. Due to a broad scope of applications, there has been a merge of computer vision with other closely related fields, such as image processing, 3D pose estimation, computer graphics, and photogrammetry (HUANG, 1996).

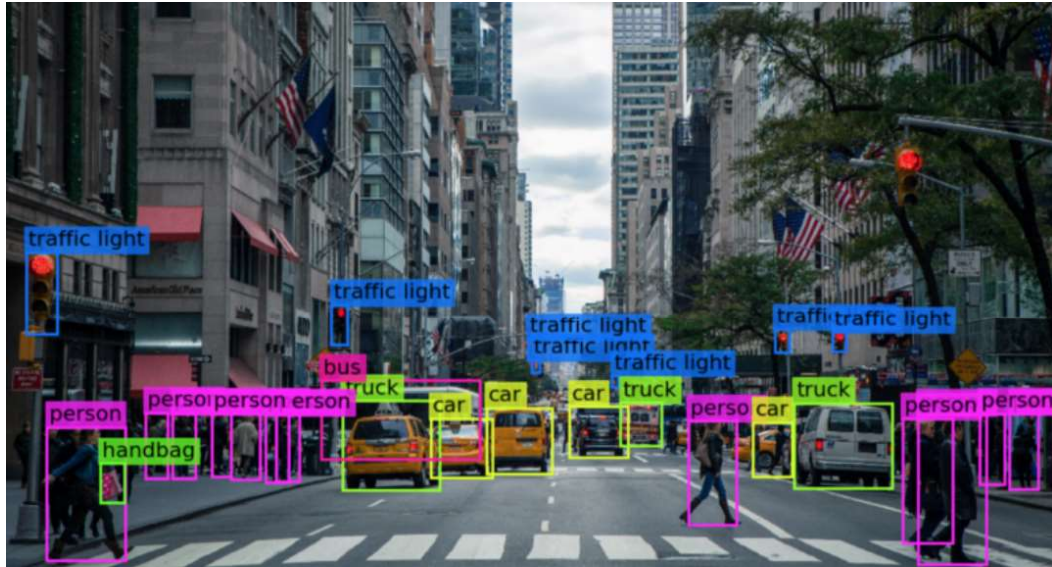


Figure 2.6: An example of object detection with YOLO algorithm, (MIHAJLOVIC, 2019)

Computer vision technology utilizes machine learning and deep learning to train computers for object detection, facial recognition, or vehicle and pedestrian detection in autonomous vehicles. Figure 2.6 shows a classic example of object detection in an image. Most of the computer vision tasks involve image classification, object detection, and image segmentation.

- **Image classification:** For image classification problem, the algorithm tries to classify the entire image as one of the classes. For instance, is this the image of a cat or not?
- **Image classification with localization:** Another category of image classification is the one that also involves localization. For this task, the algorithm not only tries to classify the entire image as one of the classes, but also it tries to find the location of that object with respect to the image.
- **Object detection:** This involves detecting multiple images in an image, for example, a dog, a cat, and a duck. Besides detecting what object is present in an image, it also involves finding the location of those objects and draw a resulting bounding box around the object of interest. The bounding boxes coordinates (in YOLO format)

contains information on the normalized coordinates x center, y center, bounding box width, and bounding box height.

- **Image segmentation:** This task of computer vision consists of two categories: (a) instance segmentation, and (b) semantic segmentation. This task involves classifying each of the pixels of the image as one of the classes, in other words, image is segmented into different segments and represented by a unique color based on the identified class.
  - **Semantic segmentation:** Semantic segmentation involves creating unique segments for every type of the object (for instance, this involves assigning unique color to every pixel of a particular object of interest). As shown in figure 2.7, as an example of instance segmentation, all the pedestrians are represented by red color, all the cars are represented by blue, all the traffic lights are represented by yellow, pavements by pink, and so on. Moreover, semantic segmentation does not only depend on the data, but it also depends on the type of problem. For instance, for pedestrian detection problem, a person's whole body would belong to one segment, whereas for action recognition system it would be necessary to assign different parts of the body to different classes (GHOSH et al., 2019).



Figure 2.7: An example of semantic segmentation for an image showing street view and all the objects of interest segmented and assigned different colors according to their classes, (DWIVEDI, 2020)

- **Instance segmentation:** It can be defined as a refined version of semantic segmentation. In instance segmentation, unlike semantic segmentation, different objects of the same class could belong to differently colored segments. This is a useful technique, particularly in autonomous driving, where a self-driving car should keep track of every instance of a class. For example, car 1 and car 2 might be moving in opposite directions or with different speeds. Figure 2.8 displays an example of instance segmentation.





Figure 2.8: An example of instance segmentation for an image showing street view and all the instances of a particular class (for instance, pedestrian or car) assigned unique segments or colors. This is when two people or cars moving at different speeds or in different directions, particularly applicable in autonomous driving problem, (PULKIT, 2019)

All of above described tasks of computer vision can be summarized into figure 2.9 for better understanding and their comparison. From figure 2.9, (a) is an example of semantic segmentation where different objects are segmented, (b) shows image classification with localization, (c) displays an example of object detection with respective bounding boxes detected, and finally (d) is an example of instance segmentation where different objects of the same class are marked with different segments, in this case dogs.

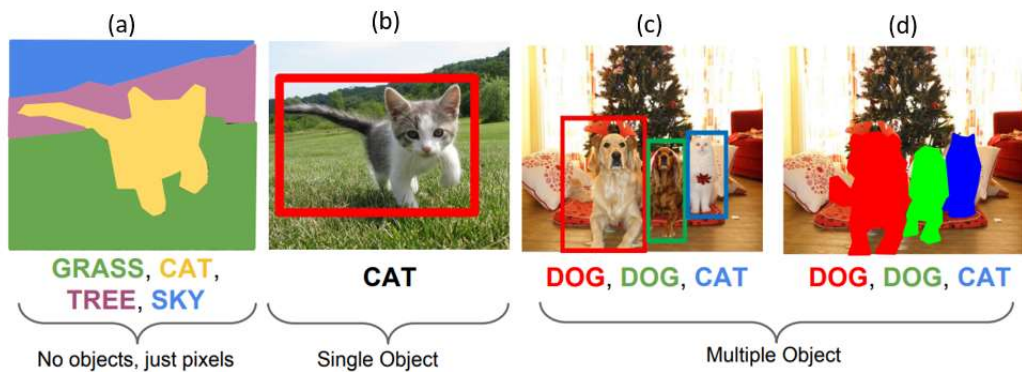


Figure 2.9: A summary of computer vision tasks (a) semantic segmentation, (b) image classification with localization, (c) object detection, and (d) instance segmentation, (LI et al., 2017)

## 2.4 AI, Machine Learning, and Deep Learning

### 2.4.1 Artificial Intelligence

Artificial intelligence is defined as a field of study of intelligent agents which refers to any system that "operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals" (RUSSELL, 2010). In other words, an intelligent agent is a system which take actions based on its environmental perceptions to maximize the outcome or in case of uncertainty maximize the expected outcome without or with minimal human intervention, often outperforming human capabilities.

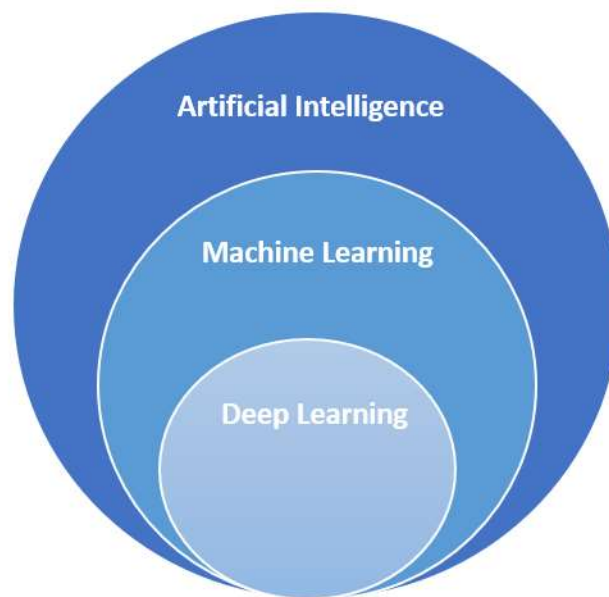


Figure 2.10: Venn diagram displaying fields of Artificial Intelligence (AI), machine learning, and deep learning with respect to each other. Deep learning is a subset of machine learning which is in turn the subset of AI

### 2.4.2 Machine Learning

Machine learning, as part of artificial intelligence, is found at the intersection of the fields of computer science and statistics. It is a field that deals with understanding and building models that predict outcomes based on a given data and improves the performance of the model with experience without being explicitly programmed to do so. Modern definition of machine learning was provided by Tom Mitchell as "A computer program is said to learn 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" (MITCHELL, 2006). Machine learning algorithms can be classified into three broad categories: supervised learning, unsupervised learning, and reinforcement learning.

- **Supervised Learning:** In supervised learning a labeled dataset (ground truth) is provided to the machine learning model which then learns and trains on this dataset to predict the outcomes or classify data accurately. The dataset includes pairs of inputs and their labeled outputs or target values which is then used to train a machine learning algorithm. Once the training is successfully completed, the model could then be used to predict the output values for the unseen input dataset (JANIESCH et al., 2021). Supervised learning further includes two categories: regression and classification. Regression includes predicting results with continuous output, for instance, predicting housing prices based on features such as house area, house location, year built, and so on. Whereas, classification problems include predicting results with discrete output, for instance, predicting if the tumor is malignant or benign.
- **Unsupervised Learning:** In unsupervised learning, a dataset is not labeled with targeted output; hence, machine learning models extract patterns or cluster the data based on shared similar features within the dataset (JANIESCH et al., 2021). An example of unsupervised learning would be cluster customers based on similar buying patterns to enhance targeted customer service. Unlike supervised learning where dataset contains both input and their labeled outputs, the dataset for unsupervised learning problems only includes the input data.
- **Reinforcement Learning:** Unlike supervised learning and unsupervised learning, input and output data is not provided. Reinforcement learning deals with specifying a goal and current state of the system, and a list of permissible actions and environmental constraints for an agent which then based on trial-and-error tries to maximize the reward for itself (JANIESCH et al., 2021). Various applications of reinforcement learning is found in robotics, autonomous driving, and gaming industry.

### 2.4.3 Deep Learning

Deep learning is a subset of machine learning which is further a subset of artificial intelligence. Deep learning is inspired by the structure of human brain. In the domain of deep learning, this structure is called Artificial Neural Network (ANN), and it involves feeding in a vast amount of data to multilayered deep neural networks to adapt and learn to perform tasks, such as speech recognition, object detection, image segmentation and so on.

As deep learning algorithms require a lot of computational power, with advancements in such technology, data availability, and new programming frameworks, deep learning models are becoming increasingly popular in discovering patterns in high-dimensional data. Therefore, deep learning is a promising development for computer vision tasks mentioned earlier (LECUN et al., 2015). Often times for such tasks, deep learning models have shown superior performances than those shown by traditional data analysis techniques and machine learning models (JANIESCH et al., 2021). Moreover, in several controlled applications, deep learning algorithms based on CNN have even outperformed humans



in terms of accuracy and speed. For instance, in a study performed by Madani and his colleagues, a trained deep learning model classified different views in echo-cardiograms much faster and more accurately than a human expert (MADANI et al., 2018). Figure 2.10 shows a Venn diagram and relative position of the fields of AI, machine learning, and deep learning.

## 2.5 Neural Networks

### 2.5.1 Artificial Neural Networks (ANN)

Artificial Neural networks (ANN) are inspired by functioning of human brain and imitate biological neurons in sending signals to one another. Neural networks have the capability to extract intricate properties and information from the high-dimensional data provided that enough data and sufficient number of hidden layers are available (VAN DAELE et al., 2019). With recent advances in the availability of computational power, computer vision tasks have seen an increase in the usage of ANN and CNN. Figure 2.11 illustrates the general architecture of a simple neural network showing the input layer, the hidden layers, and the output layer. The output of each of the nodes in the hidden layers and subsequently the output layer is calculated based on the following equation:

$$y_n = \sigma_n(\mathbf{w}_n^T \mathbf{x}_n + \mathbf{b}_n) \quad (2.1)$$

where  $\mathbf{x}_n$  is an input vector to the node  $n$ ,  $\mathbf{w}_n$  is the weight vector,  $\mathbf{b}_n$  is the bias vector,  $y_n$  is the scalar output from the node  $n$ , and  $\sigma_n$  is nonlinear activation function, such as sigmoid, Rectified Linear Unit (ReLU), Leaky ReLU, or tanh. NWANKPA et al. (2018), in their paper, compiled a list and provided description of the most common activation functions and their trends and popularity within deep learning community. Activation function, as the name suggests, decides which neuron would be activated based on the input; hence, the choice of activation function also determines the efficiency and performance of a neural network (NWANKPA et al., 2018). Various architectures for ANN exist, such as deep feed-forward neural networks, recurrent neural networks, generative adversarial networks, CNN and so on (LECUN et al., 2015).

The number of input nodes, output nodes, and hidden layers depends on the problem at hand. The number of input neurons in the input layer serves as a hyperparameter whose optimum value is chosen based on trial and error process. The number of hidden layers, as a design parameter, determines the capability of a neural network to solve complex problems. Neural networks with a higher number of internal hidden layers are more reliable and more accurate in discovering patterns in a complex problem. On the other hand, fewer hidden layers result in model having higher errors and being able to learn accurately. However, too many hidden layers result in an overfitting problem where the network seems to have a high accuracy on the training dataset yielding low training error, but providing poor generalization performance under circumstances of a new or

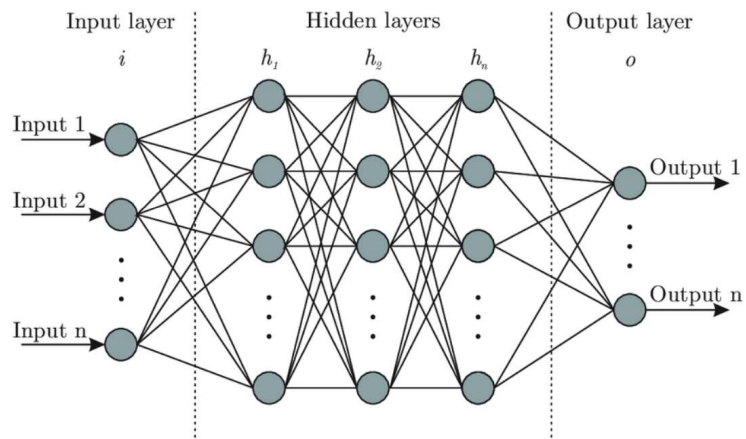


Figure 2.11: A general architecture of a simple neural network displaying the input layer, the hidden layers, as well as the output layer, (SHUKLA, 2019)

unseen data. Therefore, the decision for an optimum number of hidden layers and the number of neurons in each hidden layer is quite a challenging task.

KARSOLIYA (2012) investigated methods to approximate the number of hidden layers and number of neurons in each of the hidden layers in a neural network, and concluded that the numbers do not only depend on the number of inputs and outputs, but also on the amount of training data and the complexity of a classification problem. Although increasing the number of hidden layers and hidden neurons per layer results in increase in accuracy, it also leads to overfitting issue. Thus, it is challenging to arrive at a number which will produce reliable and accurate results in any general setting, and thus, it is usually determined based on trial and error (KARSOLIYA, 2012).

While ANN are capable of extracting useful information from a given data and learn from it, there are some limitations of such networks in computer vision tasks, such as object detection. For instance, a simple ANN with an input layer, one hidden layer, and an output layer works well for an image with a hand-written digit. However, for object detection task, a more complex detailed deep neural network is required. Hence, there are limitations of using ANN for a high-dimensional dataset such as those involved in computer vision tasks. Firstly, a high computational power would be needed for image classification and object detection tasks since regular neural networks with fully-connected hidden layers do not scale well to larger images. This involves, depending on the size of the input image, computation of a large number of weights sometimes in the order of  $10^6$  (LI et al., 2021). Secondly, it is sensitive to location of an object in an image, in other words, the image classification task is centered around locality.

## 2.5.2 Convolutional Neural Networks (CNN)

Although there are several categories of neural networks, in this thesis, particular emphasis is on the Convolutional Neural Networks (CNN) as they have seen an increasing popularity in image classification and object detection tasks. One of the first architectures of CNN

was proposed in 1990s by Yan LeCun known as LeNet (LECUN et al., 1998). As mentioned earlier that fully-connected neural networks involves massive computations as the network grows with the size of the image, and thus, requires a huge dataset to avoid overfitting issues.

One of the key features that distinguishes CNN from other neural networks is its convolution layers through which the input image is passed. Considering equation 2.2, the difference with regards to a simple ANN architecture is that in a CNN every node applies a convolution operator denoted by  $*$  on parts of its input, and then a nonlinear activation function. As a result, the learned features become increasing complex as the input is passed through the series of such convolutional layers until the captured features combine to predict an object of interest in the input image. For instance, when an image is provided as an input to CNN, the network detects the raw pixel values from the image in the first layer, then it detects the edges from raw pixels in the next layer. It then detects simple features based on the combination of the edges in the previous layer, and finally, based on the features detected in previous layers, it detects higher level features to predict or classify an object of interest. Equation 2.1 modified for CNN architecture will become:

$$y_n = \sigma_n(\mathbf{W}_n * x_n + \mathbf{b}_n) \quad (2.2)$$

where  $*$  denotes convolution and  $\mathbf{W}_n$  is the convolutional kernel(or a filter).

Within the scope of CNN, concepts such as, local receptive fields, shared weights and biases, and activation and pooling are important in understanding the architecture of such neural networks.

- **Local Receptive Fields:** As opposed to a typical neural network where each of the neurons in the input layer is fully connected to the neurons in the hidden layer, in CNN only a small region of input layer (or an input image) is connected to subsequent hidden layer. This region is called local receptive field. This patch or region of the input image is translated across the image and undergoes convolution operation with a filter matrix to create a feature map from input layer to the subsequent hidden layer. Figure 2.12 shows this convolution operation between local receptive field in of input image and filter matrix to create a feature map.
- **Shared weights and biases (Parameter Sharing):** Unlike a typical neural network, in CNN each of the hidden neurons in a given layer has same weights and biases attached to it. This deals with the locality problem in a typical neural network. Having same weights and biases for all hidden neurons in a given layer, allows the model to detect edges or blobs to create a feature map no matter where the edges are present in the image. Figure 2.13 shows a convolution operation between a patch of input image and vertical edge detector filter to detect vertical edges. These vertical edges are then combined to form geometrical primitives in subsequent layers to create complex feature maps.

- **Activation and Pooling:** The output of each neuron is then passed through an activation function (in this case [ReLU](#)) as shown in figure 2.14. This step is further processed with pooling operation, in this case max pooling, to reduce the dimensionality of the feature map by extracting more prominent features. Hence, max pooling with convolution assists in reduced dimensions and computation, position invariant feature detection, and reduced overfitting as there are less parameters.

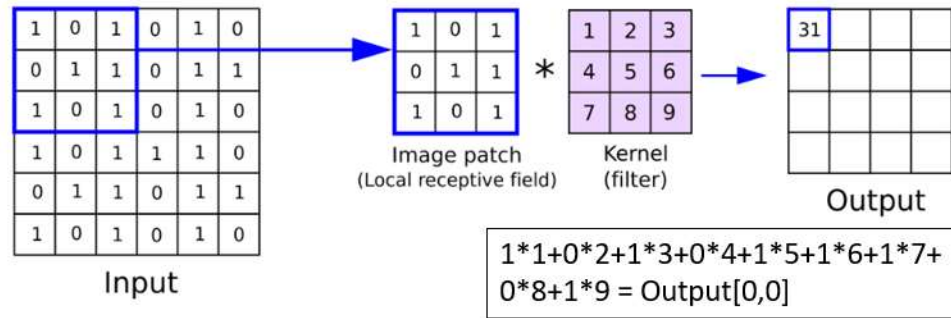


Figure 2.12: A convolution operation between a local receptive field of an image and the filter matrix (stride=2) to create a feature map (output), adapted from (REYNOLDS, 2019)

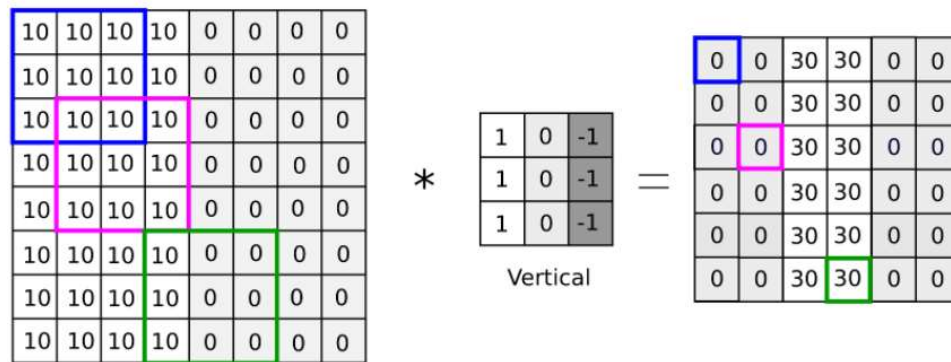


Figure 2.13: A convolution operation between a local receptive field of an image and the vertical edge detector filter matrix (stride=2),(REYNOLDS, 2019)

Figure 2.15 shows a typical network architecture of a [CNN](#) including a convolutional and ReLU activation hidden layer followed by pooling layer and so on. There are many such layers inside a [CNN](#). Towards the end of [CNN](#), there is a fully-connected dense neural network connecting all the neurons of the last hidden layer to the output layer for classification task. The first part of the [CNN](#) deals with feature extraction using convolution operations, activation function, and pooling operations. A [CNN](#) will learn the filters or kernels during training process and update it iteratively through backward propagation based on an error function to determine the optimum values inside these filters (GHOSH et al., 2019). However, the number and the size of each these filters are used as hyperparameters, but the exact values of these filters are learned during the training process.

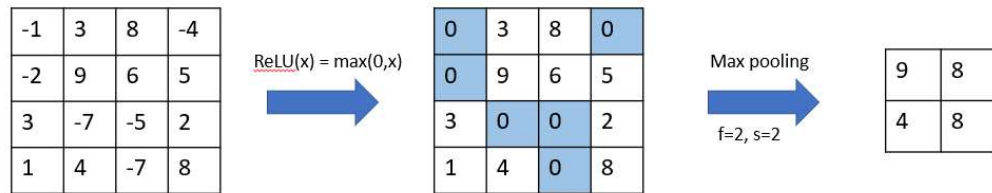


Figure 2.14: An example of ReLU activation function being applied to each of the output of neurons and then a max pooling of  $f \times f$  where  $f=2$  and stride =2 , adapted from (REYNOLDS, 2019)

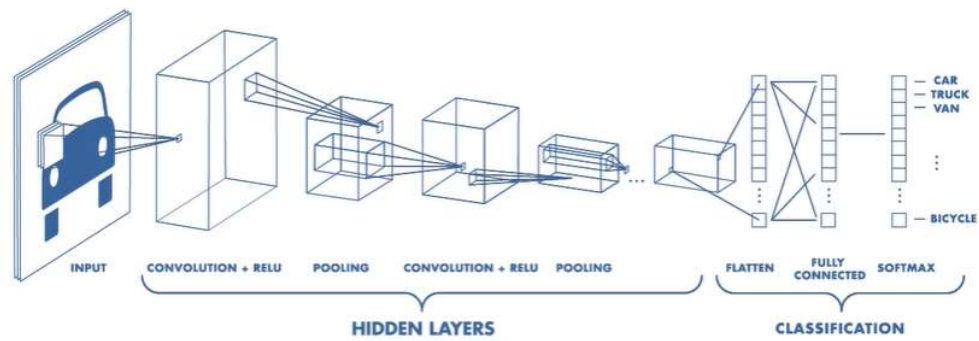


Figure 2.15: A typical CNN architecture showing convolution and ReLU hidden layers followed by pooling and so on. The network architecture can be divided into two categories: feature extraction (hidden layers) and classification, adapted from (MATHWORKS, n.d.)

## Chapter 3

# Related Research

As mentioned earlier in section 1.6, the transportation infrastructure sector, around the world, is experiencing a rising trend towards implementation of *BIM*-based collaborative technologies to create digital representation of their assets. Infrastructure undergoes aging process and is subjected to an increasing demand as well as natural hazards. These adverse factors, combined with a boom in digital innovation and technology, are motivating infrastructure owners and decision-makers to commence digital transformation of their assets. This has encouraged the development of techniques to generate *DT* of the existing infrastructure, such as bridges. While 2D technical drawings can be used to generate 3D *DT* models, this process is manual, labor-intensive, and prone-to-error, and thus, it could benefit significantly from automation. This chapter explores the related work performed by other researchers. Particularly, the concept of automated generation of *DT* from 2D technical drawings of bridges or other infrastructure will be investigated.

### 3.1 Image processing

Even before the recent advancements in computational power, computer vision technology and deep learning techniques, digitization of paper-based or scanned engineering drawings has always been an active research area that has amassed substantial interest. Considerable efforts have been made in the past to convert paper-based engineering drawings to intelligent form, such as generating 3D models.

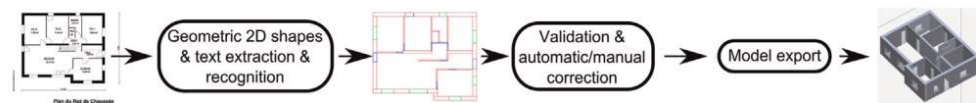


Figure 3.1: Creation of 3D models from 2D floor plans, (GIMENEZ et al., 2015)

In their research, GIMENEZ et al. (2015), have described image processing pipeline to generate 3D *DT* models from 2D architectural drawings. Although the main focus of their research was generation of 3D *DT* models based on scanned 2D plans, they have also provided a critical assessment of several non-contact and on-site data acquisition 3D model generation techniques, such as laser scanning, videogrammetry, and photogrammetry (GIMENEZ et al., 2015). The extent of granularity of extracted data from such techniques differ from aerial images to 3D Point Cloud Data (*PCD*). Furthermore, the research highlighted that in order to save computational time and memory, the elements of interest on technical drawings could also be sliced (image-slicing) to process each



smaller area separately and then combining the results. Figure 3.1 demonstrates the generation of 3D model from 2D floor plans according to the research by (GIMENEZ et al., 2015). The required steps for extracting features from 2D drawings to create a 3D digital model included: image binarization and text extraction, geometric primitives recognition, building elements recognition, 3D model validation. Since architectural plans contain a large amount of heterogeneous information, image binarization helps with minimizing the quality of information and reducing the noise, while simultaneously preserving the useful information, such as text, which is used to enrich the generated 3D model with semantic information (for example, surface area and room function and other useful dimensions). In addition, the study investigated that sparse-pixel vectorization (SPV) was the most suitable approach to detect geometrical primitives due to its linear time complexity and shape information preservation. The research also proposed that building elements, such as walls and rooms, can be detected by utilizing the edges resulting from earlier vectorization procedure or patch-based, pixel-level segmentation approach. Likewise, graph-based recognition could also be used to recognise building elements where nodes represent elements, such as doors and walls, and the edges represent the relationship between these elements (parallel, orthogonal, equal height). Hence, graph-based methods include all the geographic information needed to generate a building model : geometry (node coordinates), topology (edges), and semantics (node attributes). According to GIMENEZ et al. (2015), while there are several approaches and techniques for each of the steps of creation of 3D model as shown in figure 3.1, very less efforts have been made to address the full-fledged creation of a 3D model in an integrated way, meaning the current techniques are fragmented focusing only on the limited steps of 3D digital model creation.

In addition, YIN et al. (2008) also investigated the creation of 3D building models from CAD drawings and scanned 2D technical drawings with a particular focus on the latter category of input (scanned 2D drawings). They argue that preserving information from a CAD files and documentations (Data Exchange Format (DXF) and AutoCAD Drawing Files (DWG)) makes recognition of elements trivial due to the layered structure of CAD files. On the contrary, for raster images of a floor plan as an input, image-processing and pattern-recognition techniques must be utilized to decipher useful information since there is no obvious distinction between the graphical symbols, line types, dimension styles, and text for a 2D scanned technical drawing when provided as an input. The methodology proposed by YIN et al. (2008) for 3D model creation from 2D technical drawings or 2d floor plans is also inline with the approach suggested by GIMENEZ et al. (2015), which is that all approaches follow the same steps which are noise removal, text-extraction, vectorization, and finally recognition, and then model evaluation. Moreover, YIN et al. (2008) also compares different image vectorization techniques, such as parametric model fitting using Houghs transform, contour tracking, and skeletonization. The authors highlighted that parametric model fitting using Houghs transform is a memory intensive process and lacks generality, while contour tracking is suitable for simple floor plans and does not perform well when floors plans get complicated. While some algorithms that they studied used skeletonization techniques in vectorization and image-recognition, YIN et al. (2008)



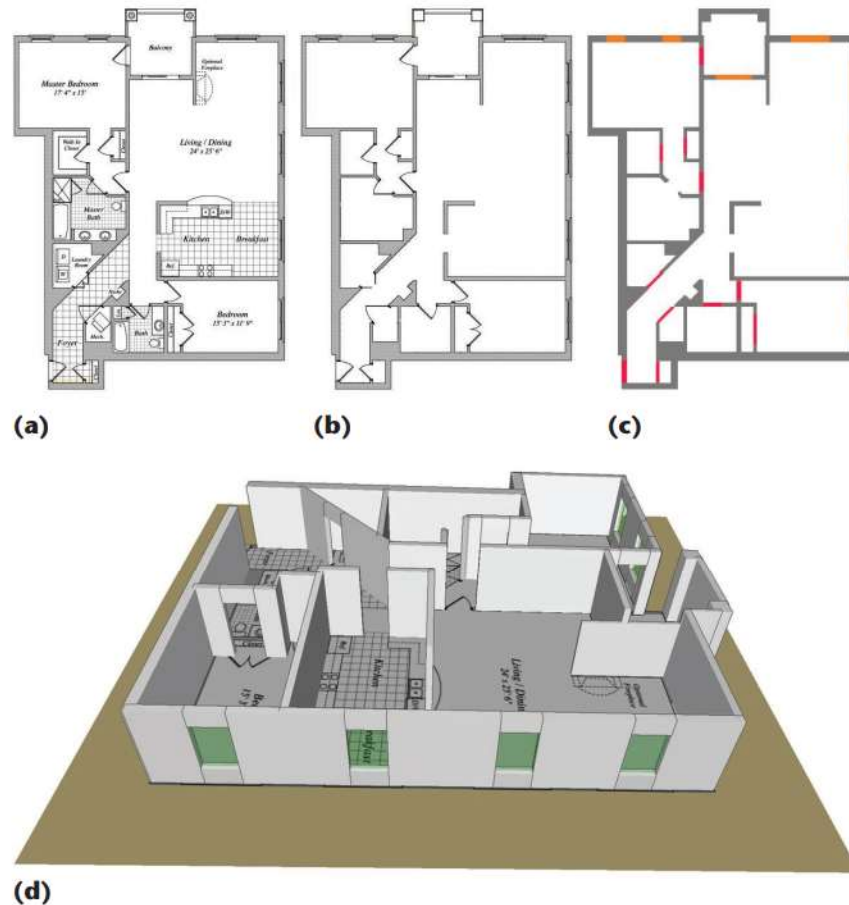


Figure 3.2: Fundamental steps in creation of 3D models from 2D scanned floor plans: (a) an original 2D scanned floor plan, (b) after denoising and text extraction, (c) graphical symbol recognition (such as doors, windows) and 2D geometry creation, and finally, (d) 3D model extrusion, (YIN et al., 2008)

identified shortcomings in those methods, such as poor performance at line intersections and inefficient algorithm as each pixel of the image is visited multiple times.

When dealing with high quality scans of technical drawings, the size of the images can become very large leading to memory-related issues. Dosch and his colleagues suggested an image-tiling algorithm that splits the images in several tiles with each tile overlapping with another to a certain degree, and the width of the overlapping region is chosen such as to allow for good subsequent matching after vectorization process (DOSCH et al., 2000).

### 3.2 Deep Learning

Most of the related work that was mentioned so far was prior to the advancements and popularisation of deep learning techniques in computer vision technology. While the older traditional methods achieved reasonable performances in a specific domain, they had generality issues when utilizing the same methods for other types of 2D scanned drawings;

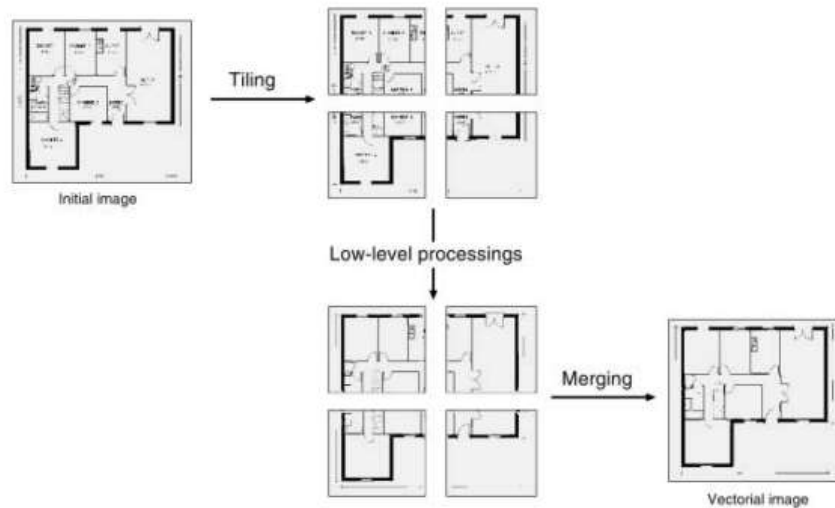


Figure 3.3: Image tiling process as described by Dosch and his colleagues, (DOSCH et al., 2000)

hence, NURMINEN et al. (2019) and MANI et al. (2020) have proposed frameworks that can be extendable to any type of technical drawings, such as architectural floor plans, electrical drawings, or construction drawing.

In their research, NURMINEN et al. (2019) and his colleagues have utilized a deep learning algorithm, such as YOLO which is a real-time object detection algorithm based on CNN to detect objects (such as, pumps and valves) in process and instrumentation (P&I) diagrams scanned from paper and stored in a vector form.

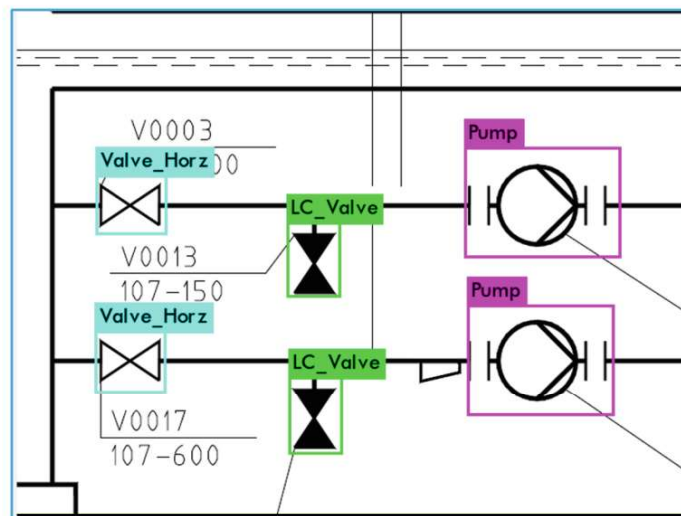


Figure 3.4: An example of high-level element detection in scanned P&I diagrams, (NURMINEN et al., 2019)

In a similar research to NURMINEN et al. (2019), MANI et al. (2020) and his colleagues have also proposed a digitization pipeline that aims to automate the detection of symbols,

text related to symbols, and connections between symbols in the P&I diagrams. The symbols to be detected on the P&I diagram included: (a) locally mounted instrument (LMI) sensors, (b) electrical signal between instruments, (c) sensor (or a tag) in a database, (d) process line, and finally (e) equipment, such as a vessel or pump. MANI et al. (2020) have trained a CNN to detect symbols on P&I diagrams and graph search approach to detect connections between symbols by traversing the diagram along its solid and dashed lines to discover interconnected symbols. In addition, MANI et al. (2020) have claimed that their pipeline will bring value to any company by converting its legacy data such as 2D technical P&I diagrams to a structured and well-defined asset hierarchy. When combined with other operational and enterprise data, the automated pipeline will not only serve as a foundation for a facility-wide digital twinning, but it will also improve and assist companies or production plants in facility management and predictive maintenance. To train their CNN, the research group cropped out tag or LMI sensors from 18 different P&I diagrams and manually labeled them, resulting in 308 tag crops and 687 LMI sensor crops. Moreover, they have extracted 100 crops from each of the 18 different P&I diagrams, resulting in 1800 'symbols-of-not-interest' crops. Inspired by LeNet architecture (LECUN et al., 1989) for digit recognition, MANI et al. (2020) have trained a CNN (figure 3.5) with three convolutional layers (with ReLU activations and max pooling) and two fully-connected dense layers. The results of symbol detection is shown in figure 3.6. For text detection, MANI et al. (2020) have used a neural network called Efficient and Accurate Scene Text Detector (EAST) (ZHOU et al., 2017) to create bounding boxes around any text found on the P&I diagrams shown in figure 3.7. Finally, to interpret the detected text, Tesseract Optical Character Recognition (OCR) was used. Several evaluation metrics, such as precision value, recall value, precision-recall curve, and Intersection over Union (IoU) were used to evaluate the performance of CNN in symbol detection, and it was seen that their CNN architecture displayed reasonable performance.

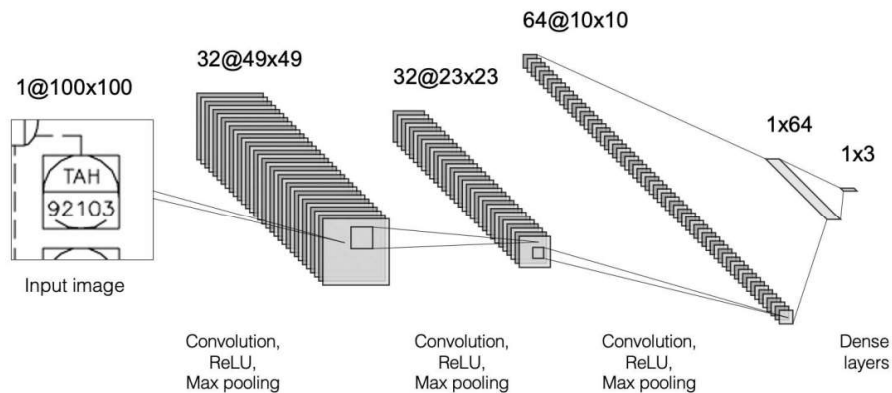


Figure 3.5: A CNN architecture for symbol detection on P&I diagrams, (MANI et al., 2020)

In another study, KANG et al. (2019) have also proposed similar pipelines that perform symbol detection and text recognition on the scanned images of P&I diagrams. They have further stressed upon the conversion of image-based technical drawings into digitize

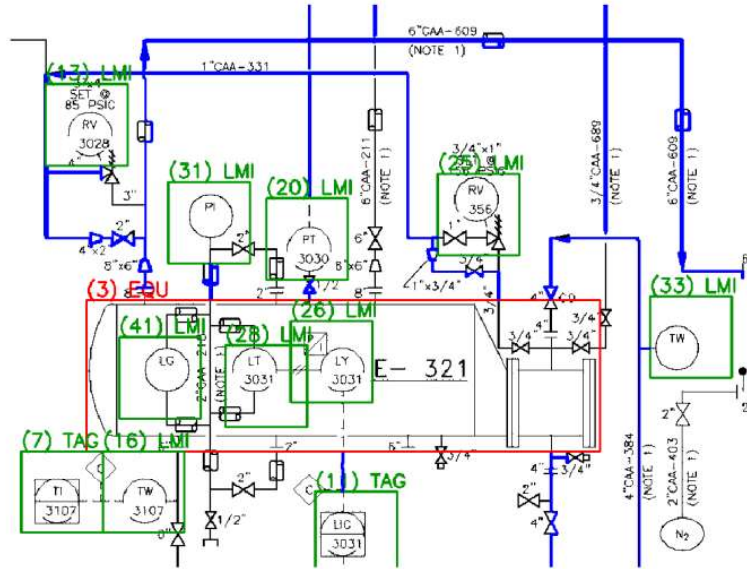


Figure 3.6: An example of source symbol detected (red), connected symbols detected (green), and connection detected between symbols (blue) through lines traversed by depth-first search (graph-based approach), (MANI et al., 2020)

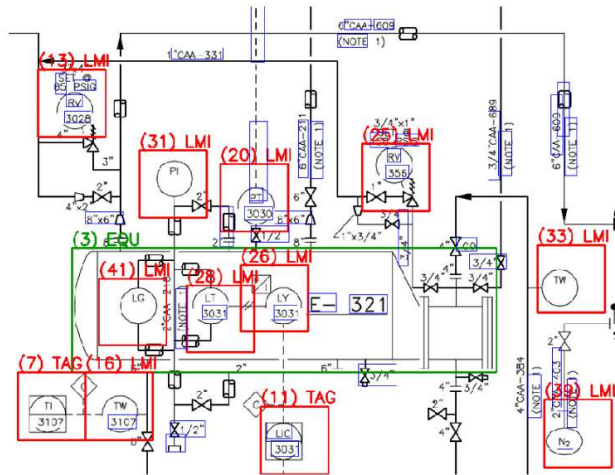


Figure 3.7: An example of symbol and text detection, (MANI et al., 2020)

structured data as AI technology and big data science fields are revolutionizing the digital technology industry. For symbol recognition, they have used template matching, a digital image processing technique, to extract elements from the P&I diagrams to automatically register the extracted data in a database. For line and text detection, sliding window method and aspect ratio calculations have been used respectively. Finally, the extracted symbols are then associated with the attributes of closest text to be then stored in a database for digitization of technical drawings. Template matching is a technique

to extract features of the image, such as shape, texture, color, to 'match' it with the target image using neural networks and deep learning classifiers. While template matching approaches are easier to implement, they usually lack generality and need a vast symbol library (templates) to perform well.

Furthermore, VAN DAELE et al. (2019) utilized a CNN-based architecture to detect machine elements in the 2D technical drawings to extract relevant knowledge from the drawings and assist engineers during the design process. The research group also proposed that such object detectors should also be able to recognize objects in historical analog drawings since such legacy drawings also contain a vast amount of information. To extract the relevant information from the technical drawings, VAN DAELE et al. (2019) have utilized: (a) the tables in the drawings to extract information, such as author, date, parts, material, and (b) the 2d views of the machine parts that contain information on shape, view, and dimensions. Figure 3.8 shows the general framework of extracting elements from 2d technical drawings as proposed by VAN DAELE et al. (2019). Data annotation was performed on 318 technical drawings and the objects in the drawings were classified into three classes: (1) table, (2) two-dimensional CAD drawings, and (3) irrelevant segments. It was noted that a simple CNN, implemented with PyTorch library (PASZKE et al., 2017), performed well with a high accuracy since all three classes were visually distinctive.

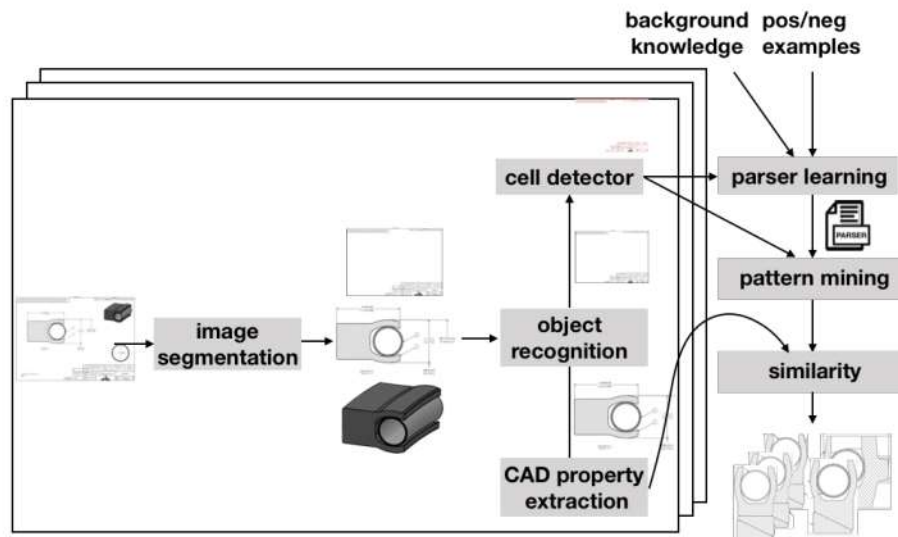


Figure 3.8: Object detection and similarity assessment framework as proposed by (VAN DAELE et al., 2019)

After the class 'two-dimensional CAD drawings' was accurately identified, the problem was turned into a binary classification problem where given a pair of 2d drawing, the classifier should predict whether the pair represents the same design or not. This would enable engineers to quickly find similar designs or relevant designs (in case of partial designs) in a large database of legacy 2d drawings. VAN DAELE et al. (2019) have used ResNet-50 (HE et al., 2016) which is one of the architectures of CNN for this binary classification

task to classify the input into one of the two possible classes: 'same' or 'different'. It was concluded that this classifier performed well with an accuracy of 96.8%.

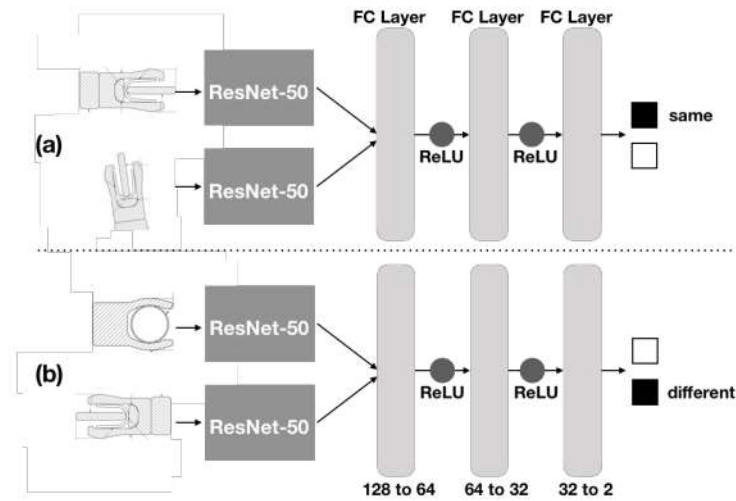


Figure 3.9: CNN architecture ResNet-50 used for binary classification of 2d drawings. If a pair of drawings is similar then assign it to a class 'same', other 'different' (VAN DAELE et al., 2019)

Based on the related research, it can be concluded that most of the techniques outlined above can be categorized into two sections: image processing and deep learning. The next chapter highlights the gaps in the knowledge based on the related research in this section, and then it outlines the objectives of this research.



## Chapter 4

# Research Gaps & Objectives

### 4.1 Research Gaps

Based on the literature review in chapter 3, it can be recognized that not much research has been performed to automate the detection of bridge elements in 2D technical drawings. On the other hand, considerable efforts have been put into detecting elements in 2D architectural floor plans (DOSCH et al., 2000; GIMENEZ et al., 2015; YIN et al., 2008) to generate 3D models of the buildings. It seems that less research has been devoted towards object detection in 2D technical drawings of the bridges. Although many researchers have proposed the methods to create 3D models of existing buildings based on their 2D architectural floor plans, there are significant visible differences between the 2D architectural floor plans and 2D technical drawings of bridges. Moreover, the bridge elements, such as abutment, wing wall, deck, and so on are not similar to the elements of a building. Figure 4.1 shows a side by side comparison between 2D floor plans and 2D technical drawings of bridges.

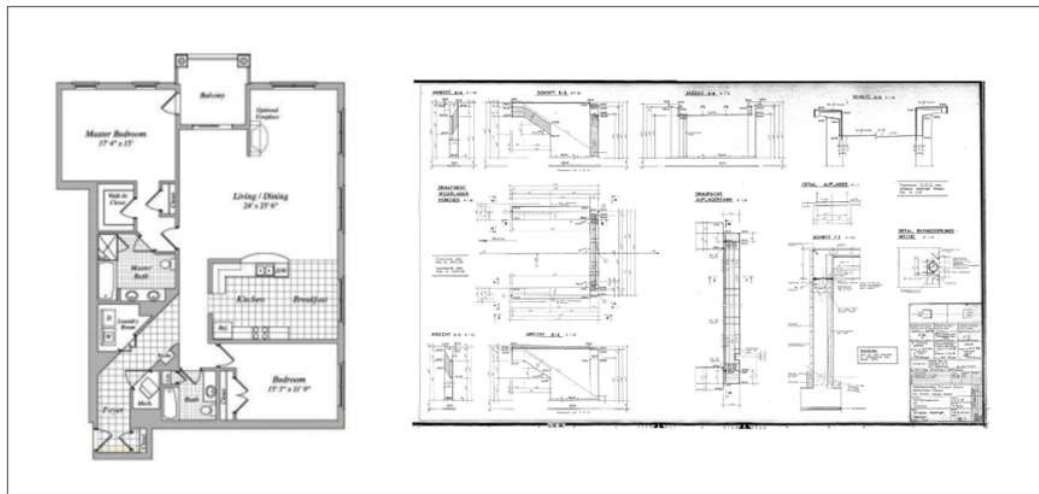


Figure 4.1: An example of a 2D architectural floor plan (left), (YIN et al., 2008) and a 2D technical drawing of a bridge (right)

Several researchers have also utilized deep learning techniques to detect elements in P&I diagrams (KANG et al., 2019; MANI et al., 2020; NURMINEN et al., 2019) and mechanical drawings (VAN DAELE et al., 2019) to extract useful information from company's archived documentations. With the exception of NURMINEN et al. (2019), others have implemented their own CNN based deep learning models. However, NURMINEN et al. (2019) and his colleagues have utilized YOLO for object detection of elements, such as pumps, sensors



and valves in P&I diagrams. But, elements in P&I diagrams are not similar to the bridge elements. Moreover, in the technical drawings of bridges, there are multiple views and sections, such as top, front, and side view.

Therefore, in this thesis, a particular attention is provided to the 2D technical drawings of bridges to extract the useful geometrical and dimensional information. And thus, this would pave a way and serve as a basis in generating DTs of existing bridges.

## 4.2 Research Objectives

As the title of this thesis suggests, the main objective of this research is to automate or at least semi-automate the task of object detection of bridge elements in 2D technical drawings of bridges with deep learning and parametric modeling. Particularly, the author tries to find the answers to the following questions:

- What useful information can be extracted from the 2D technical drawings of the bridges?
- To what degree the process of object detection of bridge elements in technical drawings of bridges can be automated?
- How is the performance of deep learning object detector model which has been selected in this thesis?
- How can the performance of the object detection model be improved?
- Is it possible to augment the existing dataset? If so, what are effects of performing such data augmentation on the overall accuracy?

## Chapter 5

# Object Detection in 2D Technical Drawings

### 5.1 Object detection

The concept of object detection was mentioned earlier in section 2.3 as one of the most frequently performed tasks in the field of computer vision. In the context of this thesis, object detection technique is used to detect the elements of bridges (such as, abutment, retaining wall, wing wall, deck, foundation, frame, and so on) in 2D scanned technical drawings. The results of object detection would then serve as a foundation for generation of DT models of the corresponding bridges. YOLO, a deep learning CNN based object detection algorithm has been utilized to detect bridge elements in technical drawings. The input to YOLO algorithm is pre-processed and annotated technical drawings of bridges and the outputs are trained weights and predicted bounding box coordinates of the detected objects on the test dataset. The post-processing steps include: (1) scaling up of bounding boxes to map them on original sized test images of technical drawings, (2) cropping out the region enclosed by predicted scaled-up bounding boxes on the original sized test images, (3) performing scene text detection to detect text and numbers on the cropped out sections from step (2). Figure 5.1 shows one of the many technical drawings used in this thesis for object detection of bridge elements.

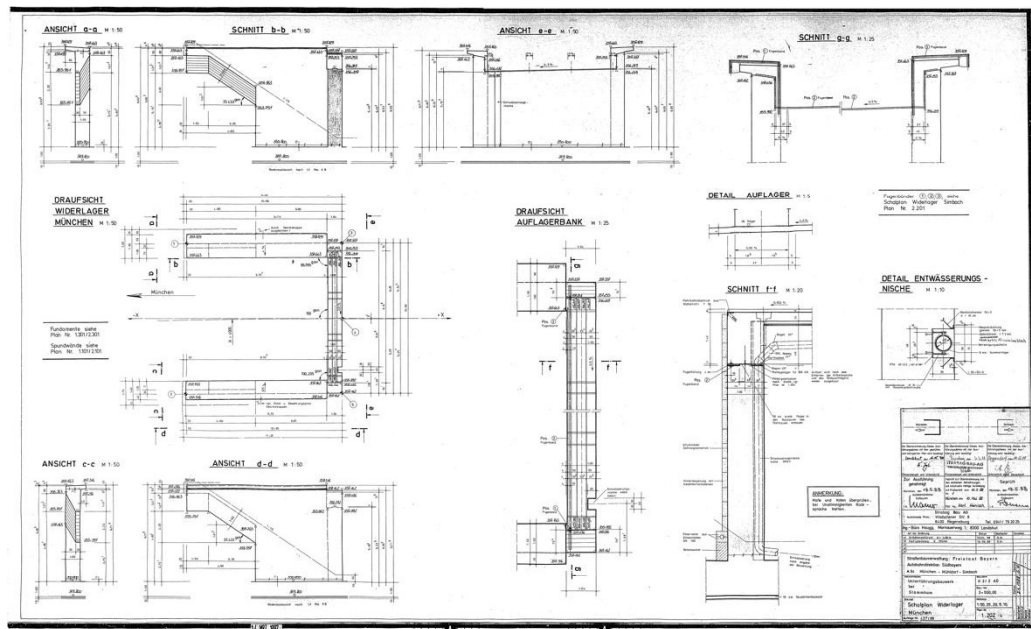


Figure 5.1: An example of a technical drawing showing different views of abutment including wing wall and retaining wall

## 5.2 YOLO

Modern object detectors can be classified into two categories: (a) one-stage object detector, and (b) two-stage object detection model. Two-stage object detection model refers to the ones which breakdown object detection tasks in two stages: (a) detecting possible object regions, and (b) classifying the objects detected in regions in step (a) into object classes. Two-stage object detector, such as Fast-RCNN (GIRSHICK, 2015) and Faster-RCNN (REN et al., 2015), uses Region Proposal Network (RPN), which is fully-convolutional network, that proposes a region which predicts object bounds and objectness scores at each position. This is step (a) as mentioned earlier, and for step (b), the output from RPN is inputted into a classifier to classify the object bounds or regions into classes. As opposed to single-stage detectors, two-stage detectors are relatively slower in detecting objects as detector is run for many iterations in the same image, and thus, preventing the real-time object detection. Although two-stage detectors are slower in object detection tasks, they usually result in higher accuracy as determined by high mAP.

YOLO (REDMON et al., 2016) is a state-of-the-art real-time single-stage object detector based on CNN. YOLO is an abbreviation of 'You Only Look Once', and as the full form suggests is a one stage object detection model. YOLO applies a single neural network to the entire image, and then divides the image into regions to predict bounding boxes and probabilities for each region. YOLO considers some important evaluation parameters, such as Intersection Over Union (IOU) and mAP to assess the performance of its object detector.

### ***Intersection Over Union:***

For each bounding box, an overlap between the predicted bounding box and the labeled bounding box (ground truth) to calculate localization accuracy. Figure 5.2 and equation 5.1 shows how IOU is calculated as the ratio of area of overlap and area of union of predicted bounding boxes and ground truth bounding box. This value also provides us with an estimate of how close the predicted bounding box is to the ground truth bounding box.

$$\text{IOU} = \frac{\text{area of overlap}}{\text{area of union}} \quad (5.1)$$

***Mean Average Precision (mAP) :*** The average precision is calculated as the area under the precision vs recall curve for a set of predictions. Recall measures how well a model finds all the positives. In equation 5.2, True Positive (TP) is the correctly identified positive, whereas, False negative (FN) is when the test results indicates the absence of a condition when it actually exists. Precision (equation 5.3) measures how accurate the predictions are, that is, how much proportion of the predictions are correctly identified. False positive (FP) indicates that a given condition exists when it does not.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (5.2)$$

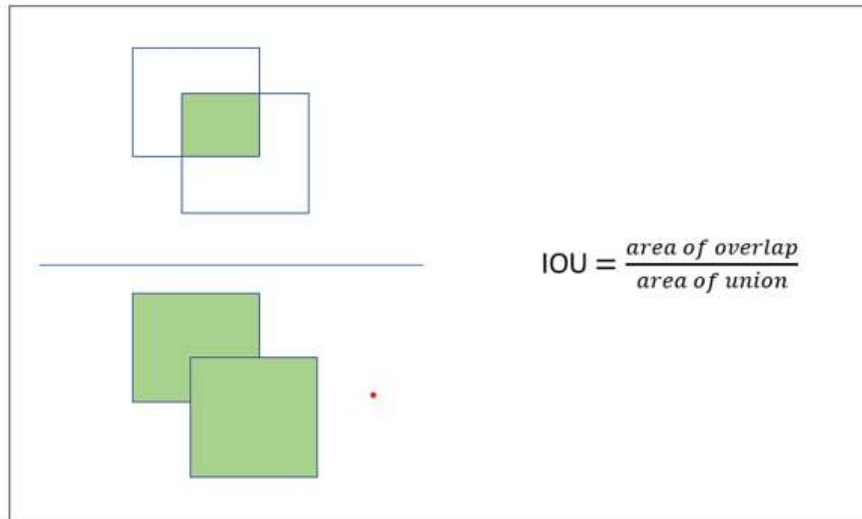


Figure 5.2: Intersection over Union (IOU)

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.3}$$

To process an image for object detection, **YOLO** segments the images into  $N$  grids each having an equal dimensional region of  $S \times S$ . Each of these  $N$  grids predict bounding boxes relative to their cell coordinates; however, this results in the prediction of multiple bounding boxes due to multiple grids predicting the same object (figure 5.3). To circumvent this problem, **YOLO** uses a non-maximal suppression to discard the bounding boxes with lower probabilities and selecting the one that has maximum **IOU** value. Figure

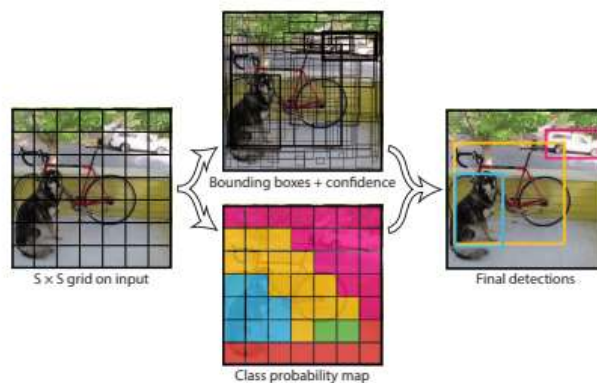


Figure 5.3: Several overlapping bounding boxes, (REDMON et al., 2016)

## 5.3 Dataset

### 5.3.1 Identifying Relevant Classes

The components of a bridge can be grouped into two broad categories: superstructure and substructure. The elements which are included in superstructure category are: deck, and girders. Whereas, the elements included in a substructure category are: abutments, wing walls, foundation, and piers. The superstructure of the bridge not only bears the load passing over it, but it also transmits the load and pressure to the substructure below it. Bearings in a bridge assist in even distribution of loads from superstructure to substructure components. The substructure of a bridge supports the superstructure and transmits the loads to bridge foundations.

The technical drawings of a bridge contain different views (top, side, front) of superstructure and substructure components. As can be observed in figure, 5.1 that it is quite difficult to identify components of the bridge at a first glance. Moreover, in the context of this thesis, the number of the classes of interest for YOLOv4 custom object detection was not obvious in the beginning. Hence, all of the available technical drawings of bridges were thoroughly examined to identify the relevant classes. While exploring the dataset, elements on the technical drawings were classified into relevant classes based on their geometrical similarities. Each of the groups of distinctively shaped elements were identified, marked, and classified into different classes. As a result of this step, 14 classes were identified from technical drawings of 15 existing bridges across Munich region. The identified classes are shown in the table 5.1 below with their class Id and class name.

Table 5.1: Identified classes for object detection in technical drawing of bridges

Class ID	Class Name
0	deck_t_shaped_cross_section
1	deck_beam_shaped_cross_section
2	deck_plain_cross_section
3	deck_top_view
4	abutment_wing_wall
5	abutment_retaining_wall
6	abutment_retaining_wall_cross_section
7	abutment_top_view
8	bridge_top_view
9	bridge_side_view
10	foundation_top_view
11	foundation_side_view
12	frame
13	table

All of the 14 identified classes are shown in the following figures from figure 5.4 to 5.17 along with their class IDs.

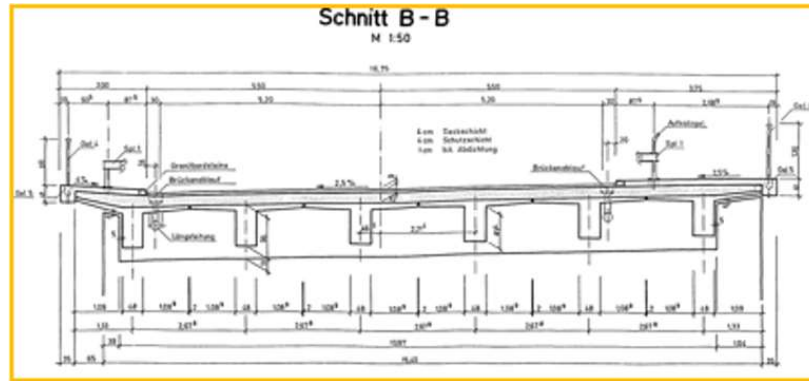


Figure 5.4: An example of class deck\_t\_shaped\_cross\_section (class ID = 0)

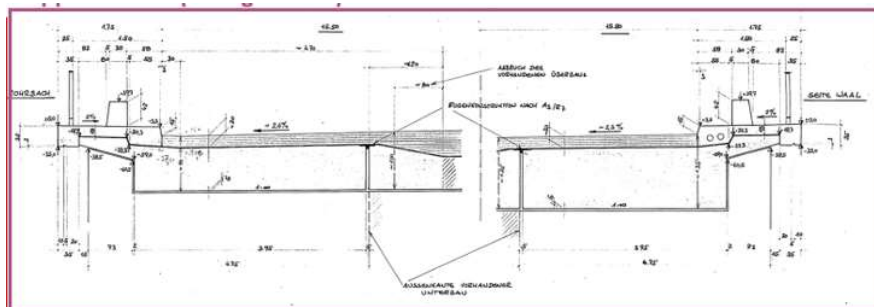


Figure 5.5: An example of class deck\_beam\_shaped\_cross\_section (class ID = 1)



Figure 5.6: An example of class deck\_plain\_cross\_section (class ID = 2)

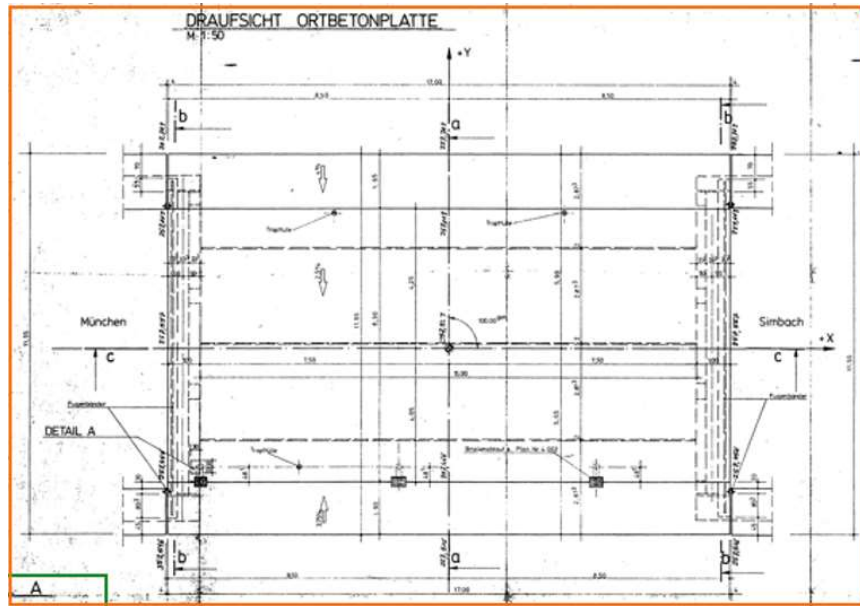


Figure 5.7: An example of class deck\_top\_view (class ID = 3)

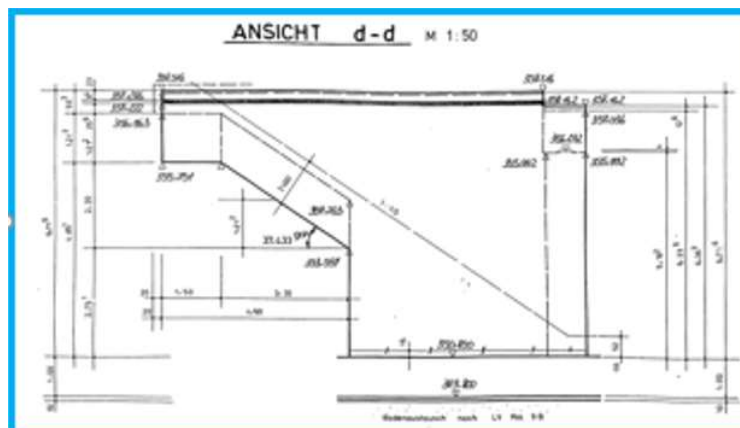


Figure 5.8: An example of class deck\_t\_shaped\_cross\_section (class ID = 4)

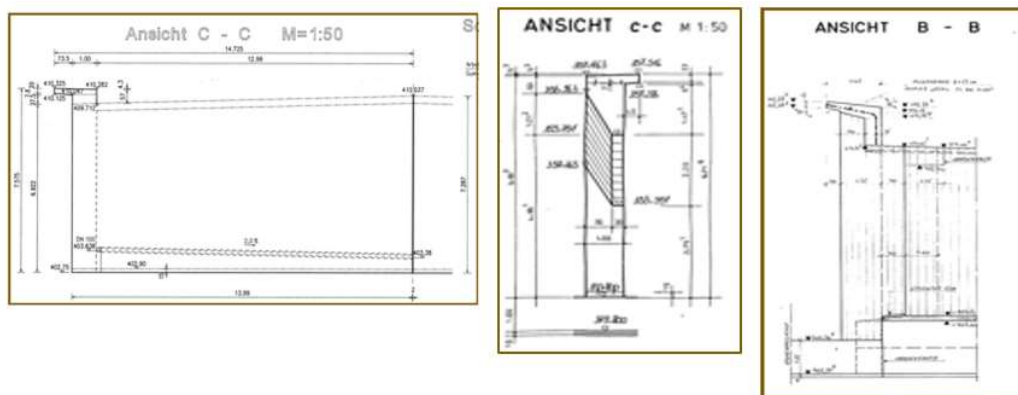


Figure 5.9: Examples of class abutment\_retaining\_wall (class ID = 5). This class also includes the cross-section of the wing-wall since it is actually a part of the retaining wall, and looks very similar



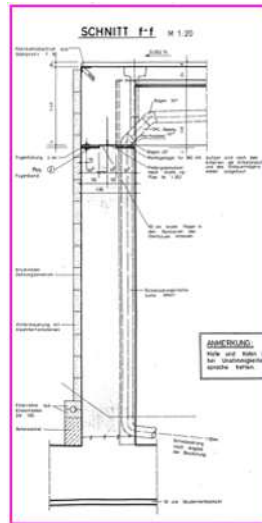


Figure 5.10: An example of class abutment\_retaining\_wall\_cross\_section (class ID = 6)

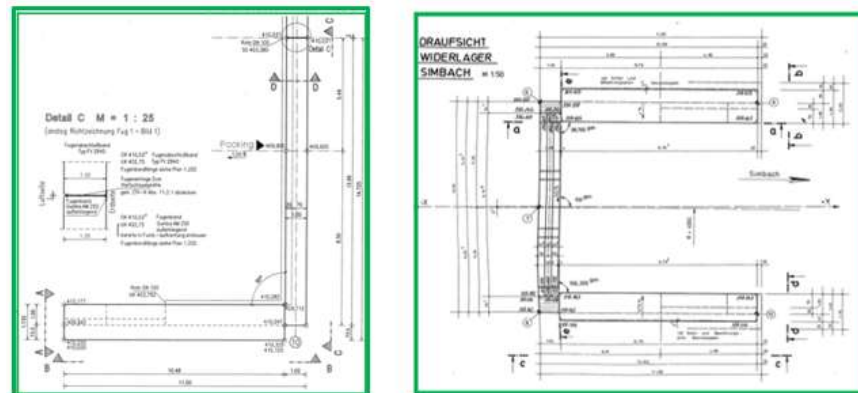


Figure 5.11: An example of class abutment\_top\_view (class ID = 7)

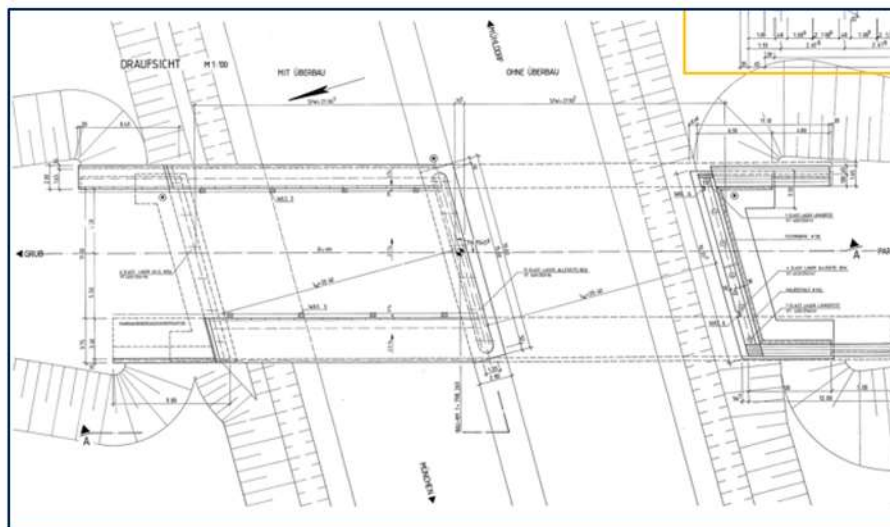


Figure 5.12: An example of class bridge\_top\_view (class ID = 8)

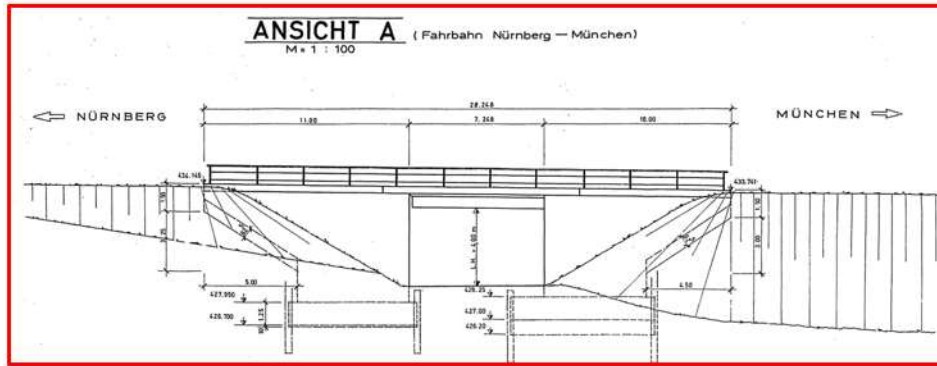


Figure 5.13: An example of class bridge\_side\_view (class ID = 9)

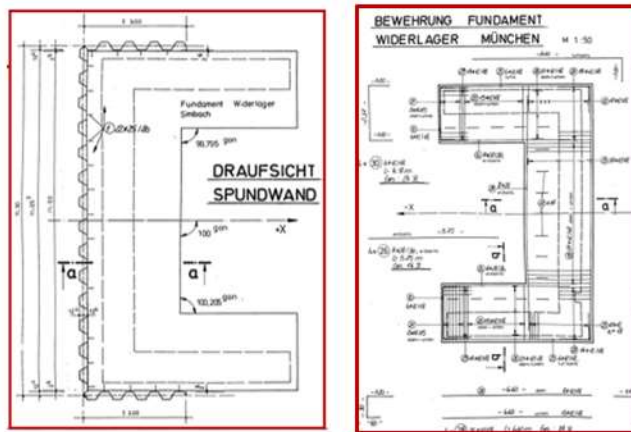


Figure 5.14: An example of class foundation\_top\_view (class ID = 10)

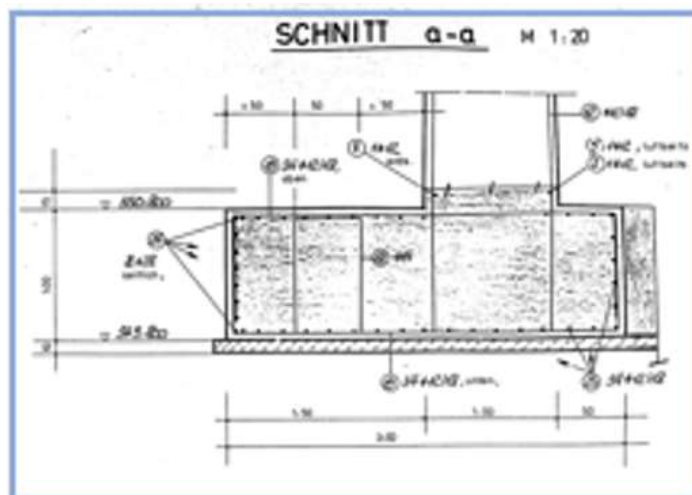


Figure 5.15: An example of class foundation\_side\_view (class ID = 11)

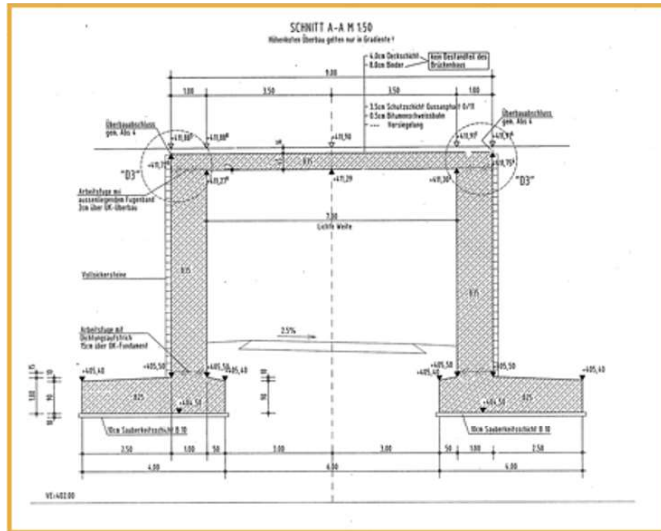


Figure 5.16: An example of class frame (class ID = 12)

STÄDTBAUVEREIN LUDWIG																									
<b>FREI STAAT BAYERN</b> Autobahndirektion Südbayern																									
<b>Standort:</b> BAB A 94 München - Pecking (A 3) Neubau Abschnitt Ingling - Oberberg von km 54 + 100 bis km 55 + 000 mit 200m Pylon / Abstellr.	<b>Blattgröße:</b> 4,64 / 1 <b>Blatt-Nr.:</b> 64 + 548.000 <b>Vermaß:</b> 1 : 50/70 <b>Preis:</b> 2.201,- €																								
<b>Schätzung Widerlager Achse 20 Nord</b>																									
<b>Veränderlicher Bestand</b> gemäß (Datei Profplan) von 20.10.2006:	<b>Neubauwerk gemäß Form 58</b> Set 2 BauPkt.:																								
... München, am 29.10.06	<b>München, am 29.10.06</b> ALFONSWERKZEUG SÜDBAYERN																								
<b>Objekt-Nr.:</b> 0301-0301 <b>Objekt-Nr.:</b> 0301-0301 <b>Objekt-Nr.:</b> 0301-0301	<b>Objekt-Nr.:</b> 0301-0301 <b>Objekt-Nr.:</b> 0301-0301 <b>Objekt-Nr.:</b> 0301-0301																								
<table border="1"> <thead> <tr> <th>Eintrag</th> <th>Prüfung</th> <th>Datum</th> <th>Auftraggeber</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Prüfung</td> <td>20.10.06</td> <td>München</td> </tr> <tr> <td>2</td> <td>Prüfung</td> <td>20.10.06</td> <td>München</td> </tr> <tr> <td>3</td> <td>Prüfung</td> <td>20.10.06</td> <td>München</td> </tr> <tr> <td>4</td> <td>Prüfung</td> <td>20.10.06</td> <td>München</td> </tr> <tr> <td>5</td> <td>Prüfung</td> <td>20.10.06</td> <td>München</td> </tr> </tbody> </table>		Eintrag	Prüfung	Datum	Auftraggeber	1	Prüfung	20.10.06	München	2	Prüfung	20.10.06	München	3	Prüfung	20.10.06	München	4	Prüfung	20.10.06	München	5	Prüfung	20.10.06	München
Eintrag	Prüfung	Datum	Auftraggeber																						
1	Prüfung	20.10.06	München																						
2	Prüfung	20.10.06	München																						
3	Prüfung	20.10.06	München																						
4	Prüfung	20.10.06	München																						
5	Prüfung	20.10.06	München																						
<b>ALFONSWERKZEUG</b> Ingenieurbüro für Projektplanung Alfred 69126 Ludwigshafen Tel: 06201-9491-0 Fax: 06201-9491-111 E-Mail: info@alfons.de Web: www.alfons.de		<b>Berger</b> Ingenieurbüro für Projektplanung Berger 69126 Ludwigshafen Tel: 06201-9491-0 Fax: 06201-9491-111 E-Mail: info@berger.de Web: www.berger.de																							
Die Überzeichnung dieses Plans ist dem Prüfer der Prüfungsstelle sowie die Bescheinigung der Eintragung des Auftraggebers wird beifolgt.																									
München, am 03.11.06																									
Auftraggeber																									

Figure 5.17: An example of class table (class ID = 13)

### 5.3.2 Preprocessing Data

- **Naming:** The technical drawings of 15 bridges were included in the dataset for object detection. For clear identification of technical drawings of each bridges, meaningful names were assigned to the bridges. The bridges were assigned the following names: bw003,bw012, bw016, bw018, bw040, bw043, bw046, bw051, bw052, bw056, bw057, bw058, bw067, bw074. The typical files names were then: bw003\_xx where xx represents the image number sequentially.
- **Resizing:** This is a necessary step since the scanned images of the technical drawings were very large. Typically, the width of the images was in the range of 12000-17000 pixels and height in the range of 7000-10000 pixels. These are a huge number of pixels to be processed, and if the original images were provided as input to YOLOv4 object detector, there were frequent memory related issues and the training would just stop abruptly. Hence, it was important to resize the images to a reasonable size without losing much information on the geometrical shapes of the elements in technical drawings.

To resize the images of technical drawings in bulk, while simultaneously maintaining the aspect ratio, Pillow Library (Pythons Image Library) was utilized. The algorithm 5.1 for resizing images is shown below where resample = 3. This means that `Image.BICUBIC` filter was chosen for resizing which compresses and also optimizes the images. Every filter follows different methodology for resizing or down-scaling the images. GUBUR (2020) has compared each of the filters based on their up-scaling quality, down-scaling quality, and performance. The height of the images was fixed at 720 pixels, so that width could be determined which gives the same aspect ratio as the original image. As it will be discussed later, this was a crucial step since it will enable scaling-up of predicted bounding boxes and map them to original sized images accurately. When predicted bounding boxes were scaled-up correctly, the detected regions bounded by the predicted bounding boxes were then cropped out and were assigned meaningful names to identify the predicted classes easily. The cropped regions were then used to detect text and dimensional information associated with the elements in the technical drawings.

Algorithm 5.1: The script below resizes images in the dataset while maintaining the aspect ratios

```
1 from PIL import Image
2 import warnings
3 import os
4 import PIL
5 import glob
6
7 Image.MAX_IMAGE_PIXELS = None
8 warnings.simplefilter('ignore', Image.DecompressionBombWarning)
9
10
```

```

11 sourcePath = 'C:\\Users\\daniyal\\Desktop\\Thesis\\obj_compressed1 '
12 targetPath = 'C:\\Users\\daniyal\\Desktop\\Thesis\\
    obj_resized_resample_4\\ '
13 print(os.listdir(sourcePath))
14 unopt_images = [file for file in os.listdir(sourcePath) if file .
    endswith(('.JPG'))]
15 print(f'Unoptimized_images_{unopt_images}')
16
17 for image in unopt_images :
18     fixed_height = 720
19     img = Image.open(image)
20     height_percent = (fixed_height / float(img.size[1]))
21     width_size = int((float(img.size[0]) * float(height_percent)))
22     print(width_size)
23     img = img.resize((width_size, fixed_height), resample=4)
24     img.save(targetPath+image)

```

---

### 5.3.3 Data Annotation

Data annotation is the process of labeling data so that a deep learning or a machine learning model can be trained on it (supervised learning). Data annotation is an indispensable task in any computer vision field related to object detection, image classification and localization, and image segmentation. Data annotation allows computers to gain a high-level understanding from the images or videos to interpret or extract useful information. Therefore, without a labeled dataset it would be impossible to train a deep learning model for object detection for custom dataset as in the context of thesis where technical drawings would be used as input images. Labeled images allow an object detector to recognize and learn the important features from a given image, so that after training, it can predict those features or classes on unseen images. Although data annotation is an important step in object detection on custom dataset, it is an extremely time-consuming process. It might take several hours (sometimes days) to label the dataset completely depending on the size of the dataset and the number of relevant objects on any image.

- **Data Annotator:** In this thesis, Labellmg (TZUTALIN, 2015) was used as an image annotator tool. It is a free, open-source software written in Python to label images in its graphical user interface. As it was observed during the data annotation process, Labellmg is a basic and easy-to-use tool that also allows the user to conveniently store the annotated bounding box information in YOLO format in a .txt file. This .txt file is automatically named similar to the image on which annotation has been performed. Figure 5.18 shows the graphical user interface of labellmg with a technical drawing of a bridge opened inside the tool. Before moving on to a next image, a complete annotation process on a single image will generate a .txt file corresponding to that .JPG image. The folder where the dataset is stored would then

look similar to the one in figure 5.19 having all the .txt files along with their .JPG files.

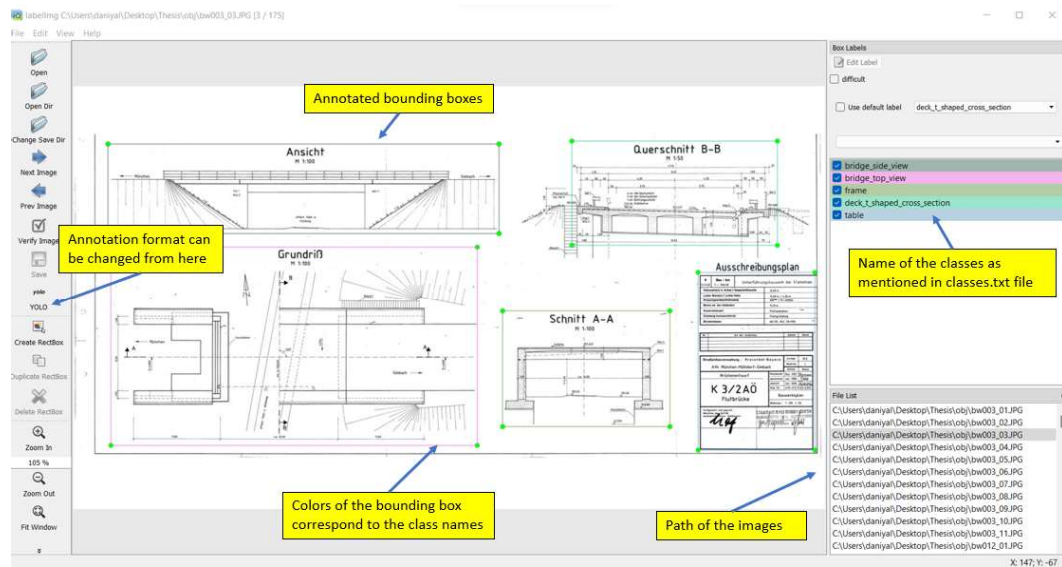


Figure 5.18: Graphical user interface of LabelImg tool with one of the technical drawings of a bridge bw003\_03.JPG opened, and also showing some of the relevant features in the tool

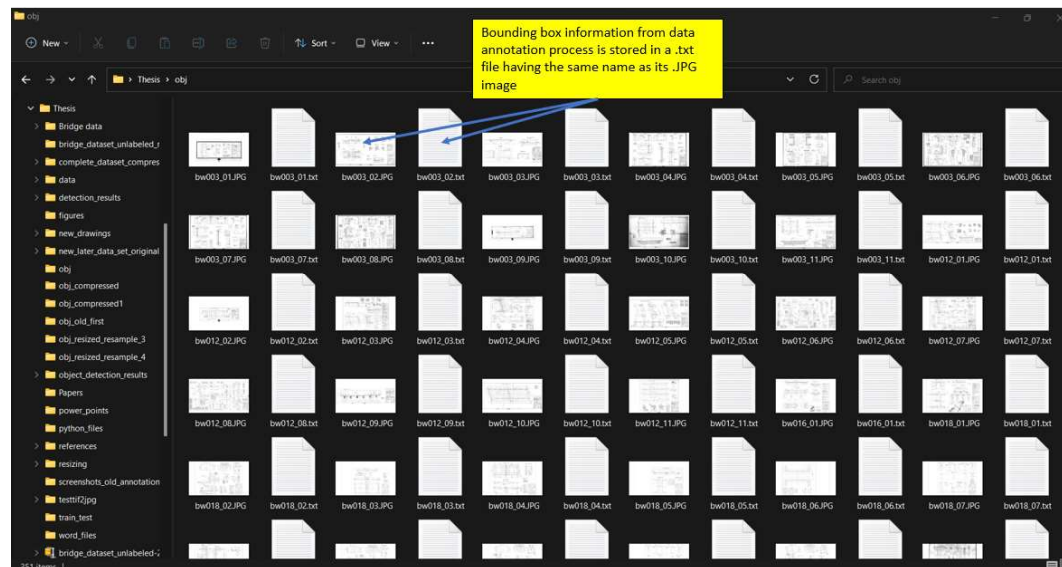


Figure 5.19: A screenshot of a dataset folder showing data labels in .txt files along with their images in .JPG format

- **Bounding Boxes in YOLO format:** Since object detection of the elements in technical drawings of the bridges will be performed with YOLOv4 object detection model, it is necessary that bounding boxes information from the data annotation tool be stored in YOLO format. For one of the technical drawings named as bw003\_03.JPG (also shown in figure 5.18), the content of the bw003\_03.txt file is shown in figure 5.20. Each line of .txt file is a bounding box (BB) within an image with the



following column-wise entries: (1) class ID, (2) normalized center\_x coordinate of BB, (3) normalized center\_y coordinate of BB, (4) normalized width of BB, and finally (5) normalized height of BB.

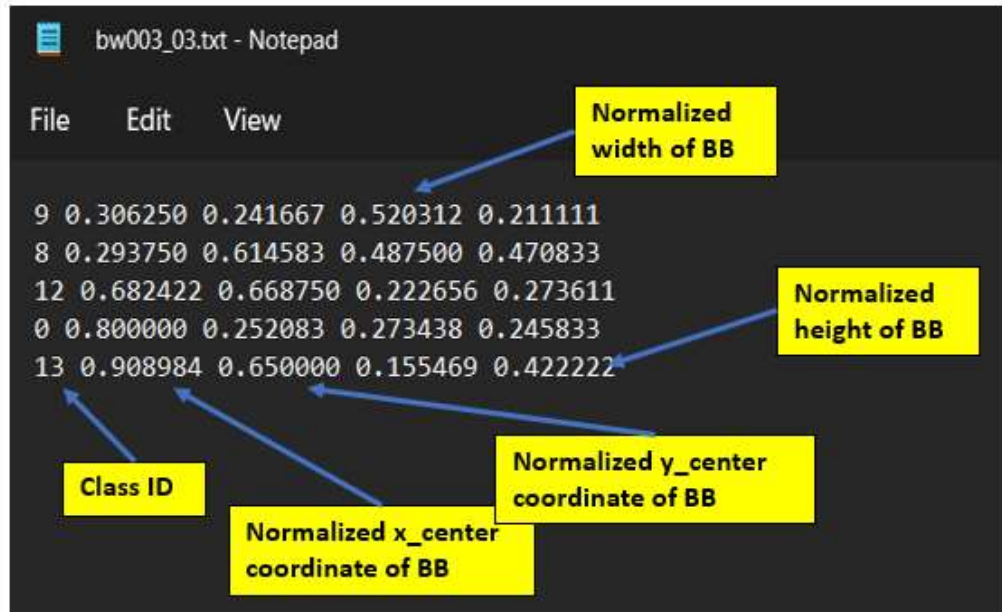


Figure 5.20: The bounding box information stored in a .txt file for a particular technical drawing

A sample calculation to convert the bounding box coordinates into YOLO format is provided below. Please, refer to 5.21 for bounding box coordinates and image size information. Bounding box coordinates in YOLO format are normalized with respect to a given image. A bounding box coordinates are calculated with respect to the top-left corner of the image and is in the following form. The top-left corner of a bounding box is  $(x_{min}, y_{min})$  or  $(64, 273)$ , the top-right corner is  $(x_{max}, y_{min})$  or  $(688, 273)$ , the bottom-left corner is  $(x_{min}, y_{max})$  or  $(64, 612)$ , and finally the bottom-right corner is  $(x_{max}, y_{max})$  or  $(688, 612)$ . The top-left corner of the image has the coordinates  $(x, y)$  or  $(0, 0)$ . Considering the sample bounding box in figure 5.21, its width and height is 624 and 339 respectively. All of the information to convert the bounding box coordinates into YOLO format is now known. Please, refer to the line 2 in the text file in figure 5.20 to understand the sample calculation below. From equation 5.4, it can be seen that normalized  $x_{center}$  coordinate would be equal to 0.29375. This can be confirmed from line 2 in figure 5.20 which is indeed 0.29375. Similarly, the normalization of  $y_{center}$ , width of BB, and height of BB will be performed.

$$\begin{aligned}
 x_c &= \frac{(x_{max} + x_{min})/2}{image\_width} \\
 &= \frac{(688 + 64)/2}{1280} \\
 &= 0.29375
 \end{aligned}
 \tag{5.4}$$

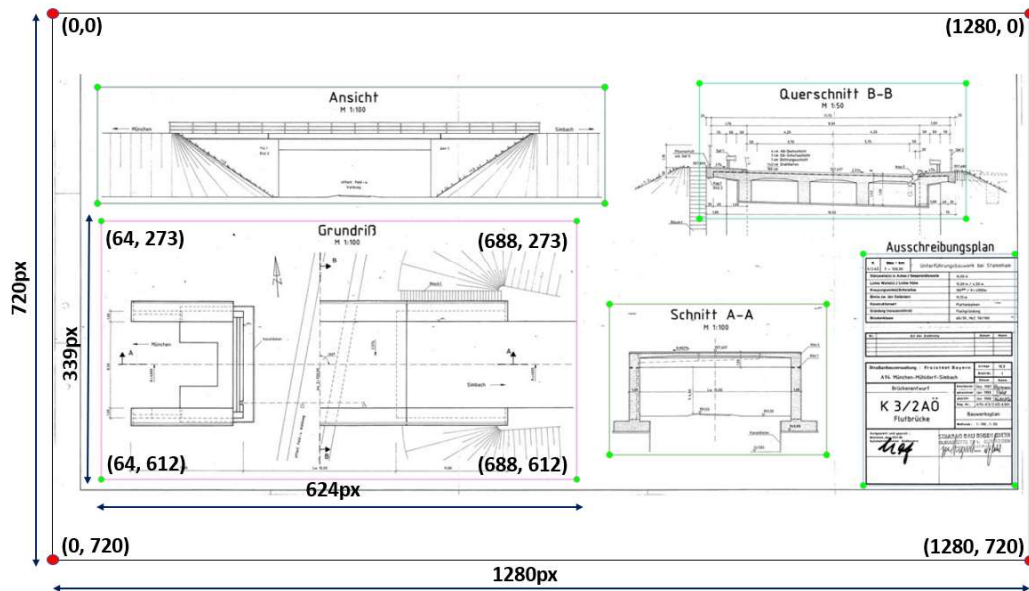


Figure 5.21: A sample image of bridge  $b_w003$  from the technical drawings dataset to show bounding box coordinates and image size in pixels

$$y_c = \frac{(y_{\max} + y_{\min})/2}{\text{image\_height}} \quad (5.5)$$

$$\text{normalized\_width\_of\_BB} = \frac{\text{BB\_width}}{\text{image\_width}} \quad (5.6)$$

$$\text{normalized\_height\_of\_BB} = \frac{\text{BB\_height}}{\text{image\_height}} \quad (5.7)$$

### 5.3.4 Counting Instances

After the data annotation part has been completed, the total number of instances for each of the classes was computed with the help of algorithm 5.2. This also ensures if the dataset is balanced or not such that every class has almost equal number of instances. However, it was observed that some of the classes had a lot of instances while the others had fewer instances only. This resulted in an imbalance dataset problem which necessitated the need to create synthetic images of the classes with fewer instances. To create synthetic images, parametric modeling techniques were utilized to draw base models of the insufficient classes in AutoCAD. The concept of parametric modeling will be discussed later in section 5.5.

Algorithm 5.2: The script below counts the number of instances for each of the classes after data annotation process

```

1 import glob
2 from collections import defaultdict
3 import csv
4 from pathlib import Path

```

```

5 import os
6
7
8 READ_PATH_TO_ANNOTATIONS = "obj"
9 READ_PATH_TO_CLASSES = Path("classes.txt")
10
11 def insensitive_glob(pattern):
12     def either(c):
13         return "[%s%s]" % (c.lower(), c.upper()) if c.isalpha() else c
14     return glob.glob(''.join(map(either, pattern)))
15
16
17 def print_statistics(counter_dict, classes):
18     for class_id in sorted(counter_dict, key=lambda x: int(x)):
19         count = counter_dict[class_id]
20         class_name = classes[int(class_id)] if int(class_id) < len(
21             classes) else None
22         print(f"Class_{class_id}_{class_name}-->Count:{count}")
23
24     print(f"Total-->Count:{sum(counter_dict.values())}")
25
26 if __name__ == '__main__':
27     classes = []
28
29     with open(READ_PATH_TO_CLASSES) as class_file:
30         classes = class_file.read().split('\n')
31
32     txt_files = insensitive_glob(f"{READ_PATH_TO_ANNOTATIONS}/*.txt")
33     overall_counter = defaultdict(int)
34     per_class_counter = defaultdict(lambda: defaultdict(int))
35
36     for txt_file_path in txt_files:
37         # Get the name of the image class
38         img_base_name = os.path.splitext(os.path.split(txt_file_path)
39             [-1])[0].rsplit("_", 1)[0]
40
41         with open(txt_file_path) as csv_file:
42             csv_reader = csv.reader(csv_file, delimiter=',')
43
44             for row in csv_reader:
45                 overall_counter[row[0]] += 1
46                 per_class_counter[img_base_name][row[0]] += 1
47
48     # Print overall statistics
49     print("Overall_statistics:")
50     print_statistics(overall_counter, classes)
51
52     # Print per structure class statistics

```

```

52     for img_base_name, cls_counter in per_class_counter.items():
53         print(f"\n{img_base_name}_statistics")
54         print_statistics(cls_counter, classes)

```

---

The script 5.2 was run in the root directory where `classes.txt` file is present. For the path to annotations (line 8) in the script, the relative path to the dataset folder (named `obj`) was provided. After running the script 5.2, the outputs were: (1) the "overall statistics" of all the bridges (figure 5.22) and the "statistics" of each of the bridges (figure 5.23, 5.24). Similarly, the number of instances of each of the classes were computed for all of the bridges. From figure 5.22, it can be observed that 1142 elements for a total of 14 classes were present after data annotation. The results of counting instances have been summarized in figure 5.25.

```

(base) C:\Users\daniyal\Desktop\Thesis>python count_instances_2.py
Overall statistics:
Class (0, deck_t_shaped_cross_section) --> Count: 20
Class (1, deck_beam_shaped_cross_section) --> Count: 21
Class (2, deck_plain_cross_section) --> Count: 20
Class (3, deck_top_view) --> Count: 19
Class (4, abutment_wing_wall) --> Count: 135
Class (5, abutment_retaining_wall) --> Count: 242
Class (6, abutment_retaining_wall_cross_section) --> Count: 87
Class (7, abutment_top_view) --> Count: 152
Class (8, bridge_top_view) --> Count: 22
Class (9, bridge_side_view) --> Count: 46
Class (10, foundation_top_view) --> Count: 119
Class (11, foundation_side_view) --> Count: 81
Class (12, frame) --> Count: 23
Class (13, table) --> Count: 155
Total --> Count: 1142

```

Figure 5.22: A snippet of the output console where the overall statistics of all of the 15 bridges have been computed

```

bw056 statistics
Class (0, deck_t_shaped_cross_section) --> Count: 7
Class (3, deck_top_view) --> Count: 7
Class (4, abutment_wing_wall) --> Count: 9
Class (5, abutment_retaining_wall) --> Count: 24
Class (6, abutment_retaining_wall_cross_section) --> Count: 8
Class (7, abutment_top_view) --> Count: 19
Class (8, bridge_top_view) --> Count: 1
Class (9, bridge_side_view) --> Count: 6
Class (10, foundation_top_view) --> Count: 20
Class (11, foundation_side_view) --> Count: 8
Class (12, frame) --> Count: 5
Class (13, table) --> Count: 9
Total --> Count: 123

```

Figure 5.23: A snippet of the output console where the statistics of bw056 bridge have been computed

```

bw003 statistics
Class (0, deck_t_shaped_cross_section) --> Count: 5
Class (3, deck_top_view) --> Count: 2
Class (4, abutment_wing_wall) --> Count: 8
Class (5, abutment_retaining_wall) --> Count: 16
Class (6, abutment_retaining_wall_cross_section) --> Count: 4
Class (7, abutment_top_view) --> Count: 8
Class (8, bridge_top_view) --> Count: 2
Class (9, bridge_side_view) --> Count: 2
Class (10, foundation_top_view) --> Count: 3
Class (11, foundation_side_view) --> Count: 3
Class (12, frame) --> Count: 1
Class (13, table) --> Count: 10
Total --> Count: 64

```

Figure 5.24: A snippet of the output console where the statistics of bw003 bridge have been computed

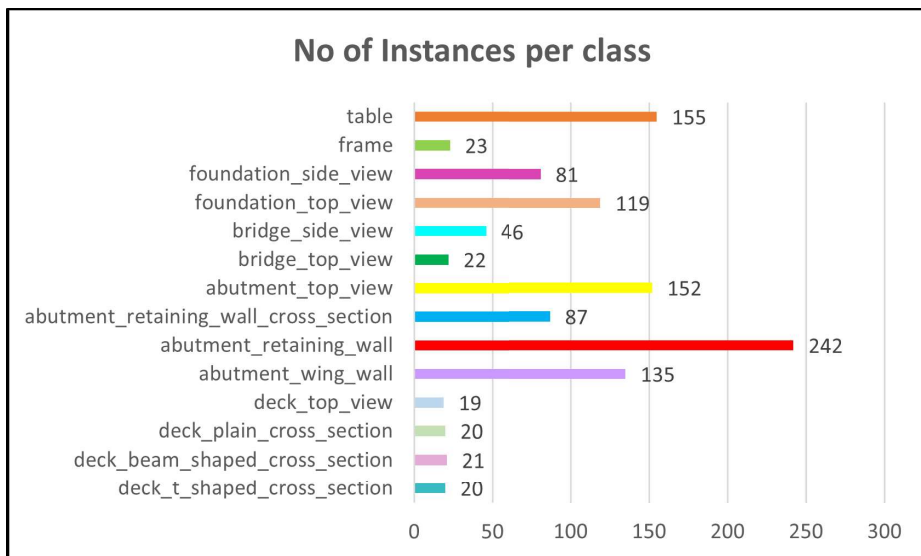


Figure 5.25: A bar chart displaying the total number of instances for each of the classes for a total of 15 bridges

## 5.4 Object Detection with YOLO: A Step-by-Step Guide

### 5.4.1 Training YOLOv4 on a Technical Drawings Dataset

In this section, a complete methodology to build and train a custom YOLOv4 object detector using Google Colab has been outlined. This would be a step-by-step methodology to guide the reader on how to train YOLO on custom dataset. In the scope of this thesis, the custom dataset is images of 2D technical drawings of existing bridges across Munich region. Later on, YOLOv4 was trained for different hyperparameters to find the best combination of hyperparameters (configuration parameters) for better accuracy and time efficiency. Please, refer to section 5.6 for more details on running several unique cases for different

set of hyperparameters. While different sets of hyperparameters resulted in different average loss, mAP values and execution time, YOLOv4 was trained once again for the final time with custom dataset as well as synthetic dataset from parametric modeling. For more details on parametric modeling, please refer to section 5.5. The last training was performed with the best combination of hyperparameters that were selected with regards to their degree of accuracy and time to reach convergence.

### **Step 1: Enabling GPU accelerator in a Google Colab Notebook**

For object detection with YOLOv4, Google Colab Pro's GPU cloud computing environment was utilized. GPU acceleration was enabled for Colab notebook as shown in figure 5.26 to train and process detections much faster than a CPU.

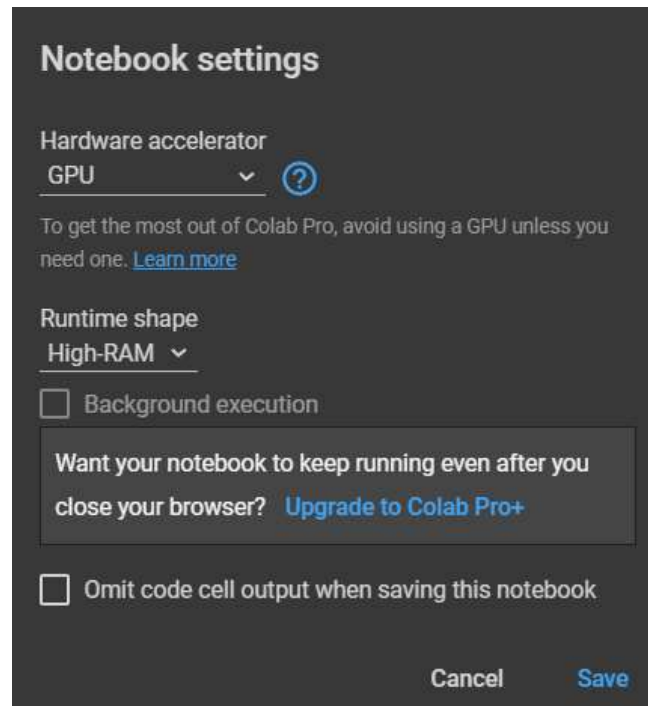


Figure 5.26: Enabling GPU acceleration in Google Colab Pro

### **Step 2: Cloning and Building Darknet**

The following snippet of code will clone AlexeyAB's github repository (ALEXEYAB, 2021) and modify the Makefile to enable OPENCV and GPU for darknet. CUDA version will also be verified, and the darknet would be built so that darknet executable file could be run to train and test the object detector.

Some changes have to be made in the file `detector.c` in `src` folder so that the trained weights are saved in `backup` folder after every 500 iterations. Normally, if this change is not made, the weights will only be saved after every 10000 iterations. The figure 5.27 shows the line 395 after modification.

**Step 4: Downloading pre-trained weights** YOLOv4 has already been trained on Microsoft Common Objects in Context (COCO) dataset (LIN et al., 2014) which includes 80



```

392 //if (i % 1000 == 0 || (i < 1000 && i % 100 == 0)) {
393 //if (i % 100 == 0) {
394 if ((iteration >= (iter_save + 10000) || iteration % 10000 == 0) ||
395     (iteration >= (iter_save + 500) || iteration % 500 == 0) && net.max_batches < 50000)
396 {
397     iter_save = iteration;

```

Figure 5.27: Modifications in detector.c file for iterations to be saved at every 500th iteration

classes of common everyday objects. These pre-trained weights which have been trained up to 137 convolutional layer would be utilized, so that transfer learning technique could be applied such that geometrical primitives such as edges and blobs could be quickly detected to form a feature map based on the combination of these edges and blobs with the scope of object detection in technical drawings.

Algorithm 5.3: clone, build, make darknet from AlexeyAB github repository

```

1
2 # clone darknet repository from AlexeyAB
3 !git clone https://github.com/AlexeyAB/darknet
4
5 # change makefile to have GPU and OPENCV enabled
6 %cd darknet
7 !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
8 !sed -i 's/GPU=0/GPU=1/' Makefile
9 !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
10 !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
11 # verify CUDA
12 !/usr/local/cuda/bin/nvcc --version
13
14 # make darknet
15 !make
16
17 # download pre-trained weights
18 !wget https://github.com/AlexeyAB/darknet/releases/download/
    darknet_yolo_v3_optimal/yolov4.weights

```

**Step 5: Defining helper functions** These functions would help in viewing the images directly in Colab notebook after running the detections. The validation images would then have predicted bounding boxes around the detected objects along with the class probability. The resulting could then be downloaded. Images could also be uploaded from Google Drive once it is mounted to the cloud VM.

Algorithm 5.4: Helper functions to display object detection results on images within Colab notebook

```

1 # define helper functions to view detection results in Colab notebook
2 def imshow(path):
3     import cv2
4     import matplotlib.pyplot as plt
5     %matplotlib inline

```

```

6
7 image = cv2.imread(path)
8 height, width = image.shape[:2]
9 resized_image = cv2.resize(image,(3*width, 3*height), interpolation =
    cv2.INTER_CUBIC)
10
11 fig = plt.gcf()
12 fig.set_size_inches(18, 10)
13 plt.axis("off")
14 plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
15 plt.show()
16
17 # To upload a file
18 def upload():
19     from google.colab import files
20     uploaded = files.upload()
21     for name, data in uploaded.items():
22         with open(name, 'wb') as f:
23             f.write(data)
24             print('saved_file', name)
25
26 # To download a file after detection results
27 def download(path):
28     from google.colab import files
29     files.download(path)

```

---

### Step 6: Mounting Google Drive

To gain access to the Google Drive content to run object detection, the lines of code below mount the Google drive into the cloud VM and creates a symbolic link between `'/content/gdrive/My Drive/'` and `'/mydrive'` to create a shortcut `'/mydrive'` to map to the contents with in `'/content/gdrive/My Drive/'`. Sometimes, this step is also necessary because the space in `'My Drive'` folder can create issues when running certain commands.

---

#### Algorithm 5.5: Mounting Google Drive into Cloud VM

---

```

1 from google.colab import drive
2 drive.mount('/content/gdrive')
3
4 # this creates a symbolic link so that now the path /content/gdrive/My\
    Drive/ is equal to /mydrive
5 !In -s /content/gdrive/My\ Drive/ /mydrive

```

---

### Step 7: Creating a Yolov4 folder

After mounting Google Drive, now is the time to create `Yo1ov4` folder where all the configuration files will be uploaded to be then copied to the `darknet` folder to run the detections. This `Yo1ov4` folder will also contain a backup file where trained weights will be stored

after every 500 iterations. The best weights and the last weights (weights after every 100 iterations) will also be stored in the `backup` folder. The last weight file gets rewritten after every 100 iterations. An example of how the `backup` folder will look like after training is shown in the figure 5.28.

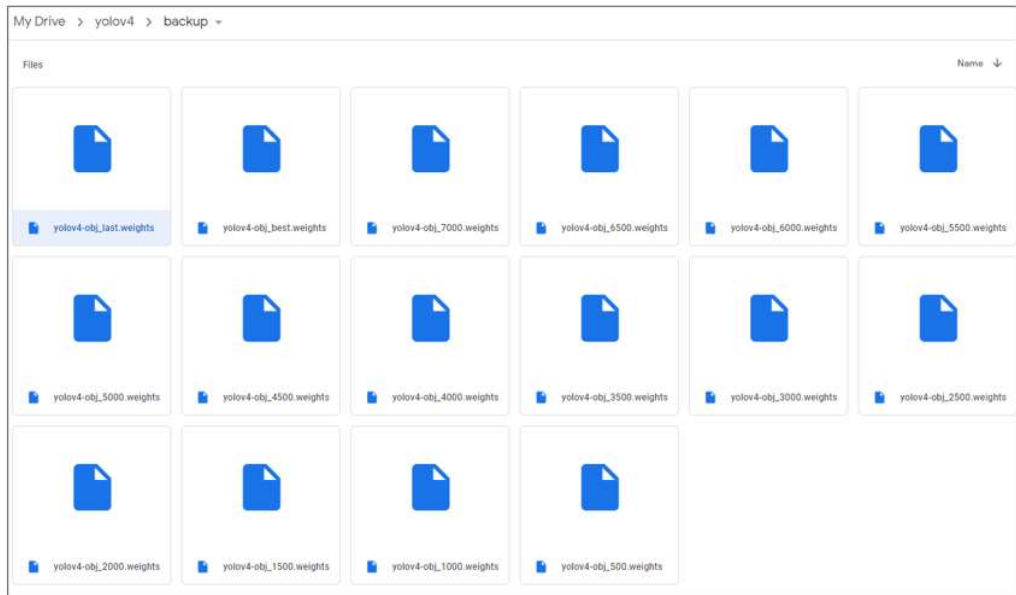


Figure 5.28: The contents of a `backup` folder after training

**Step 8: Creating and modifying configuration files for custom object detection**

Before the training process, the contents of `Yo1ov4` folder are shown in the figure 5.29. Now, the content of each of the files will be discussed in detail.

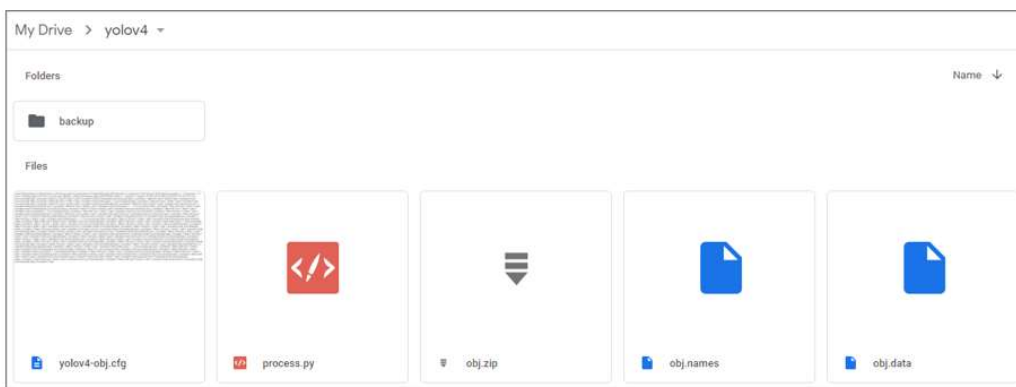


Figure 5.29: The contents of a `Yo1ov4` folder before training

**obj.zip:** This `.zip` file consists of labeled dataset of 2D technical drawings of bridges. This will be copied to `darknet` folder where it will be extracted and divided into `train.txt` and `test.txt` files

**obj.names:** This file contains the name of all the classes. Figure 5.30 shows the content of the `obj.names` file where each line is a class.

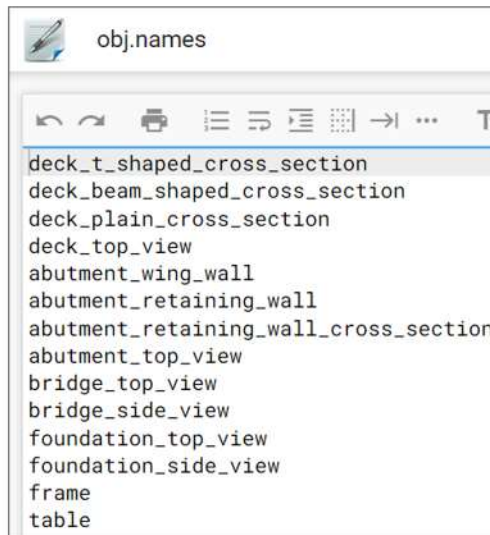


Figure 5.30: The names of classes in `obj.names` file

**obj.data:** This file contains the number of classes, path to the `train.txt` file, `test.txt` file, `obj.names` file, and `backup` folder where the training weights will be stored. Figure 5.30 shows the content of this file.

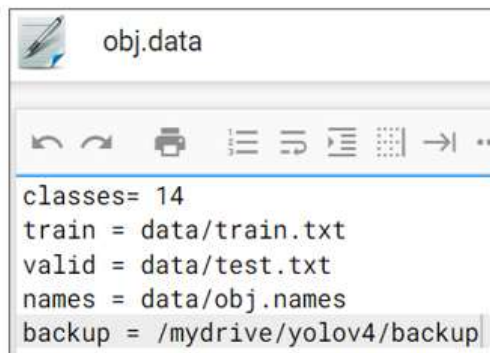


Figure 5.31: Content of a `obj.data` file

**process.py:** This is a python file which is run to split the technical drawings dataset into training data and validation data. The content of this file is shown as script 5.6. The dataset was divided as 90% training data and 10% validation data.

---

Algorithm 5.6: Splitting dataset into training and validation data

---

```
1 import glob
2 import os
3 import numpy as np
4 import sys
5
6 current_dir = "data/obj"
7 split_pct = 10;
8 file_train = open("train.txt", "w")
9 file_val = open("test.txt", "w")
```

```

10 counter = 1
11 index_test = round(100 / split_pct)
12 for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.JPG")):
13     title, ext = os.path.splitext(os.path.basename(
14         pathAndFilename
15     ))
16     if counter == index_test:
17         counter = 1
18         file_val.write(current_dir + "/" + title + '.JPG' + "\n")
19     else:
20         file_train.write(current_dir + "/" + title + '.JPG' + "\n")
21         counter = counter + 1
22 file_train.close()
23 file_val.close()

```

---

**yo1ov4-obj.cfg:** This is the most important file to run the object detection as it contains hyperparameters that can be modified based on trial-and-error to determine the most optimum hyperparameters (shown in the red circle in figure 5.32). Moreover, the parameters (shown in the green circle in figure 5.32) that have to be modified for custom dataset object detection depending on the number of classes are: `max_batches`, `steps`, and `classes` and `filters` in all three `yo1o` layers.

Sample calculations are shown below with regards to technical drawings dataset:

For `max_batches`, it should be noted that the value cannot be lower than 6000 if even you have 1 or 2 classes. The minimum value should be 6000.

$$\begin{aligned}
 \text{max\_batches} &= \text{classes} * 2000 \\
 &= 14 * 2000 \\
 &= 28000
 \end{aligned}
 \tag{5.8}$$

For `steps`, the value is 90% and 80% of the `max_batches`. Hence, the values are `steps = 22400, 25200`.

For `classes` in each of the 3 `yo1o` layers, search for keyword `yo1o` and three instances will be found in `yo1ov4-obj.cfg`. Modify the `classes` from 80 to 14 for technical drawings case. Moreover, `filters` values before each of the 3 `yo1o` layers have to be changed for custom dataset according to the following relation:

$$\begin{aligned}
 \text{filters} &= (\text{classes} + 5) * 3 \\
 &= (14 + 5) * 3 = 57
 \end{aligned}
 \tag{5.9}$$

### **Step 9: Moving custom dataset into Cloud VM for object detection**

All of the files from the `yo1ov4` will be copied into `darknet` directory to run object detection. The lines of code below executes this process. Moreover, after running `process.py` script

```

yolov4-obj.cfg
1- [net]
2 # Testing
3 #batch=1
4 #subdivisions=1
5 # Training
6 batch=32
7 subdivisions=16
8 width=608
9 height=608
10 channels=3
11 momentum=0.949
12 decay=0.0005
13 angle=0
14 saturation = 1.5
15 exposure = 1.5
16 hue=.1
17
18 learning_rate=0.001
19 burn_in=1000
20 max_batches = 28000
21 policy=steps
22 steps=22400,25200
23 scales=.1,.1

```

Figure 5.32: Content of a yolov4-obj.cfg file

in darknet folder, train.txt and test.txt files are generated, and the content of these files are shown in the figure 5.33. The train.txt and test.txt files must then be moved to data sub-folder in darknet folder to maintain path consistencies such as those shown in the figure 5.31.

Algorithm 5.7: Moving custom dataset and files from Google Drive to darknet folder

```

1 # cd back into the darknet directory to run detections
2 %cd darknet
3
4 #to deal with permission denied issues
5 !sudo chmod +x darknet
6 !./darknet
7
8 #Copy the obj.zip file to current directory (darknet)
9 !cp /mydrive/yolov4/obj.zip ../
10
11 # unzip the dataset (obj.zip) into /darknet/data/ subdirectory
12 !unzip ../obj.zip -d data/
13
14 # upload the custom .cfg to darknet cfg folder
15 !cp /mydrive/yolov4/yolov4-obj.cfg ./cfg
16
17 # copy the obj.names and obj.data files to darknet data folder
18 !cp /mydrive/yolov4/obj.names ./data
19 !cp /mydrive/yolov4/obj.data ./data
20
21 #copy and run process.py script to darknet
22 !cp /mydrive/yolov4/process.py ./

```



```

train.txt X Resources
1 data/obj/bw057_07.JPG
2 data/obj/bw043_04.JPG
3 data/obj/bw046_02.JPG
4 data/obj/bw052_15.JPG
5 data/obj/bw056_21.JPG
6 data/obj/bw003_06.JPG
7 data/obj/bw074_09.JPG
8 data/obj/bw058_03.JPG
9 data/obj/bw018_09.JPG
10 data/obj/bw054_02.JPG
11 data/obj/bw051_09.JPG
12 data/obj/bw067_09.JPG
13 data/obj/bw046_09.JPG
14 data/obj/bw074_03.JPG
15 data/obj/bw057_02.JPG
16 data/obj/bw052_11.JPG
17 data/obj/bw040_04.JPG
18 data/obj/bw052_02.JPG
19 data/obj/bw018_11.JPG
20 data/obj/bw067_01.JPG
21 data/obj/bw058_05.JPG
22 data/obj/bw003_10.JPG
23 data/obj/bw056_07.JPG
24 data/obj/bw052_08.JPG
25 data/obj/bw057_05.JPG
26 data/obj/bw052_01.JPG
27 data/obj/bw003_04.JPG

test.txt X Resources
1 data/obj/bw040_09.JPG
2 data/obj/bw054_10.JPG
3 data/obj/bw067_08.JPG
4 data/obj/bw056_02.JPG
5 data/obj/bw012_06.JPG
6 data/obj/bw051_02.JPG
7 data/obj/bw074_12.JPG
8 data/obj/bw052_07.JPG
9 data/obj/bw003_11.JPG
10 data/obj/bw012_02.JPG
11 data/obj/bw012_05.JPG
12 data/obj/bw043_03.JPG
13 data/obj/bw012_01.JPG
14 data/obj/bw018_06.JPG
15 data/obj/bw058_09.JPG
16 data/obj/bw052_09.JPG
17 data/obj/bw051_05.JPG

```

Figure 5.33: Content of a `train.txt` (left) and `test.txt` (right) files. Please, note only few lines are shown here

### Step 10: Running YOLOv4 on technical drawings

Finally, YOLOv4 object detector can now be run on the custom dataset of technical drawings of bridges. Following line of code has to be executed for training to begin. It has a `train` command, path to `obj.data` and `yolov4-obj.cfg`, pretrained weights, `-dont_show` to suppress the output, and `-map` flag to also calculate `mAP` values. This step completes the procedures to train custom object detector using Google Colab in cloud VM. The training process should continue as long as average loss value decreases to a small value in the range of 0.5-2, and `mAP` value increases to a higher value. If no changes are seen in the values of average loss and `mAP`, then the training process could be stopped even before the training is completed by itself. However, it is recommended to train the custom object detector for 20000 iterations after determining the best set of hyperparameters. (ALEXEYAB, 2021).

Algorithm 5.8: This command starts the training for custom object detection

```

1 !./darknet detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv
  .137 -dont_show -map
2
3 #if for some reasons the training stops, use the following command to
  start the training with last saved weights
4 !./darknet detector train data/obj.data cfg/yolov4-obj.cfg /mydrive/
  yolov4/backup/yolov4-obj_last.weights -dont_show -map

```

## 5.4.2 Testing the trained YOLOv4 on Technical Drawings Dataset

### Step 1: Testing trained YOLOv4 on a single technical drawing

After the training has been completed, the trained model will be tested on validation dataset as well as some test images which were not included in validation dataset. To test the detector, `yolov4-obj.cfg` file has to be modified as follows (code 5.9). The line 8 in the algorithm 5.9 includes `test` command, path to `obj.data`, `yolov4-obj.cfg`, `best.weights` and the validation or testing image (in this case image `bw057_15`).

#### Algorithm 5.9: Modifications in `yolov4-obj.cfg` file for testing the detector

```
1 # need to set yolov4-obj.cfg to test mode
2 %cd cfg
3 !sed -i 's/batch=64/batch=1/' yolov4-obj.cfg
4 !sed -i 's/subdivisions=16/subdivisions=1/' yolov4-obj.cfg
5 %cd ..
6
7 #to test the detector run the following command
8 !./darknet detector test data/obj.data cfg/yolov4-obj.cfg /mydrive/
   yolov4/backup/yolov4-obj_best.weights /mydrive/images/bw057_15.jpg
9 imshow('predictions.jpg')
10 !cp predictions.jpg /mydrive/images/bw057_15_1_1.jpg
```

### Step 2: Testing trained YOLOv4 on multiple technical drawings

YOLOv4 detections can also be run on multiple images at once provided that a `.txt` file, in this case `images.txt`, contains the absolute paths to the test or validation images. the figure 5.34 shows the paths to the multiple images to run detections at once. After running the code in 5.10, the `result.txt` contains coordinates of predicted bounding boxes as well as predicted class names.

Algorithm 5.10: Execute the following code to run detections on multiple images and save the results of predicted bounding boxes in a `.txt` file

```
1 !./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -
   dont_show -ext_output < /mydrive/images.txt > result.txt
2 download('result.txt')
```

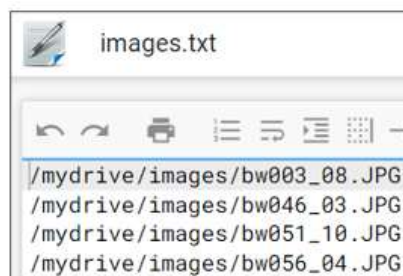


Figure 5.34: Content of a `image.txt` file for running detections on multiple test images at once.

More details related to the testing phase of object detector will be provided in chapter 6, where the detector will be tested on multiple test images to show the predicted bounding boxes on the resized images to avoid memory related issues. The coordinates of predicted bounding boxes will be present in `.result.txt` file. The coordinates in this file are not in YOLO format; hence, the predicted bounding boxes coordinates will first be converted to YOLO format. The bounding boxes in YOLO format would then be scaled-up to map them to the original sized images, so that the regions bounded by the bounding boxes can be extracted and cropped out to be saved as separate images for scene-text detection. All of the mentioned tasks will now be provided in the chapter 6.

## 5.5 Parametric Modeling and Data Augmentation

Parametric Modeling is a technique in CAD that assists designer in saving time by eliminating the need to redraw a design model every time if one of the design parameters changes. Parametric modeling involves specifying design constraints and parameter constraints to control the size and shape of the model. In this thesis, the parametric models for the classes with insufficient instances were created in AutoCAD. The tab which allows parametric modeling in AutoCAD is shown in figure 5.35.

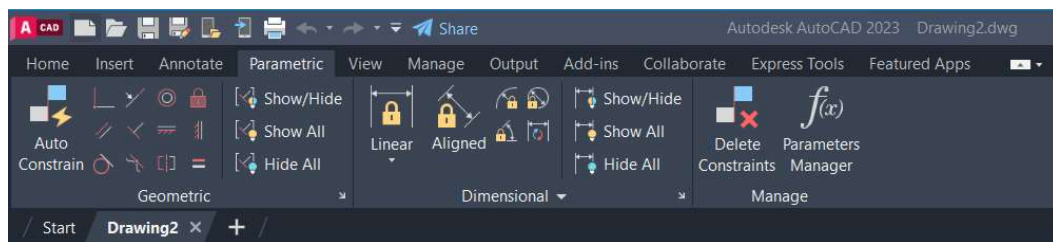


Figure 5.35: Parametric Modeling tab in AutoCAD

As can be observed from figure 5.49 that several classes, such as `frame`, `bridge_side_view`, `bridge_top_view`, `deck_top_view`, `deck_plain_cross_section`, `deck_beam_shaped cross section`, and `deck_t_shaped_cross_section` have insufficient number of instances. Thus, the original dataset is imbalanced. This is why parametric modeling techniques have been implemented in AutoCAD to at least try to create model elements of classes with fewer instances to augment the dataset. This is also one of the techniques of data augmentation where new data is generated or added to a current imbalanced dataset so as to balance the dataset and improve the overall accuracy.

Some of the parametric models for insufficient classes will now be described. It will be difficult to cover all the insufficient classes in detail within the scope of this section. Only parametric models of classes `frame`, `deck_t_shaped_cross_section`, and `bridge_side_view` will be described in detail. Rest of the parametric models are summarized in the figure 5.44 with all geometrical and dimensional constraints hidden so that the reader can have a clearer view of the drawings.

For class "frame", the parametric model has been shown in figure 5.36 with geometrical and dimensional constraints. In addition, the figure 5.37 also shows how easily the parametric models could be modified to create a completely new and unique instance of a class. The modifications are shown in the circles. Figure 5.38 displays that dynamic dimensions can also be converted to annotated dimensions so that upon exporting the .pdf file of a synthetic technical drawing, the dimensions are also visible on the drawing.

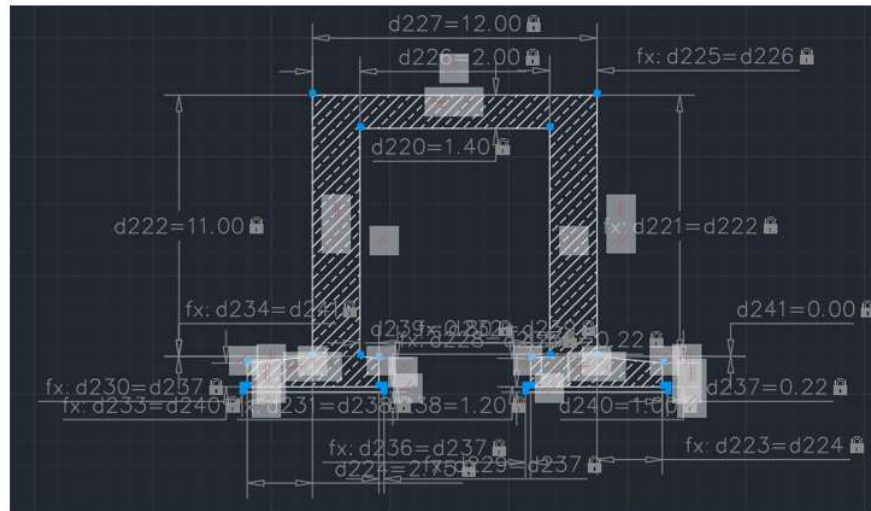


Figure 5.36: An example of parametric model of class "frame" with geometric and dimensional constraints

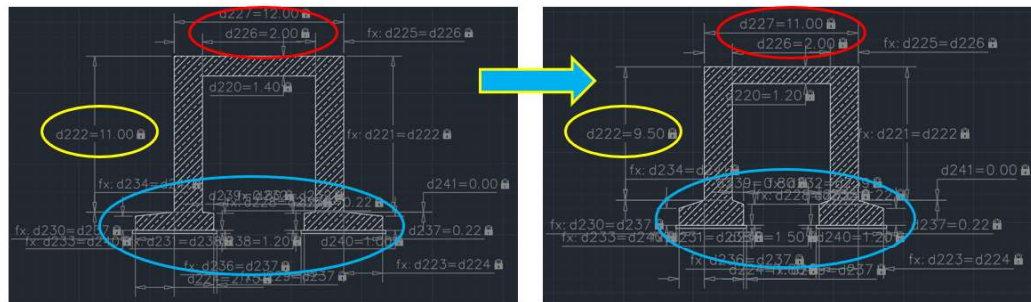


Figure 5.37: An example of parametric model of class "frame". It can be noticed that the dimensions have been changed easily to create another unique instance of the same class. Some of the changes are shown in the circles

For class "deck\_t\_shaped\_cross\_section", the parametric model has been shown in figure 5.39 with geometrical and dimensional constraints. In addition, the figure 5.40 also shows how easily the parametric models could be modified to create a completely new and unique instance of that class. The modifications are shown in the circles.

For class "bridge\_side\_view", the parametric model has been shown in figure 5.41 and 5.42 with geometrical and dimensional constraints. Moreover, 5.43 shows two instances of class "bridge\_side\_view" with geometrical and dimensional constraints hidden for reader to have a clear view of the diagram.

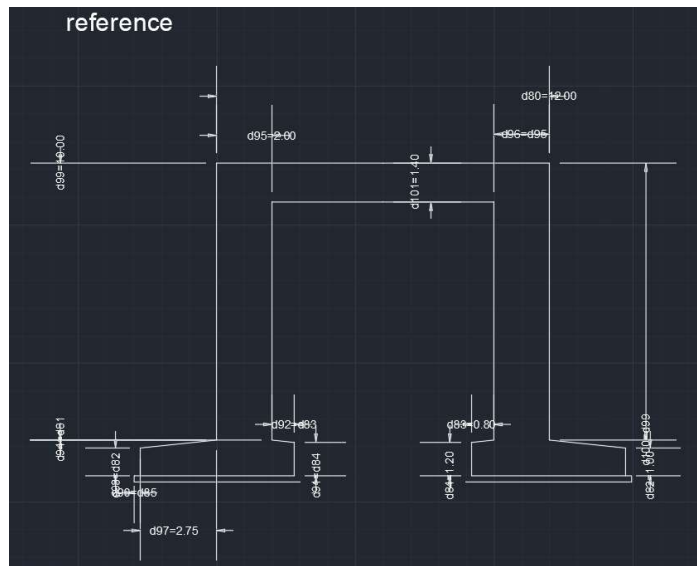


Figure 5.38: To convert dynamic dimensions into annotated dimensions for showing dimensions on the synthetic technical drawings

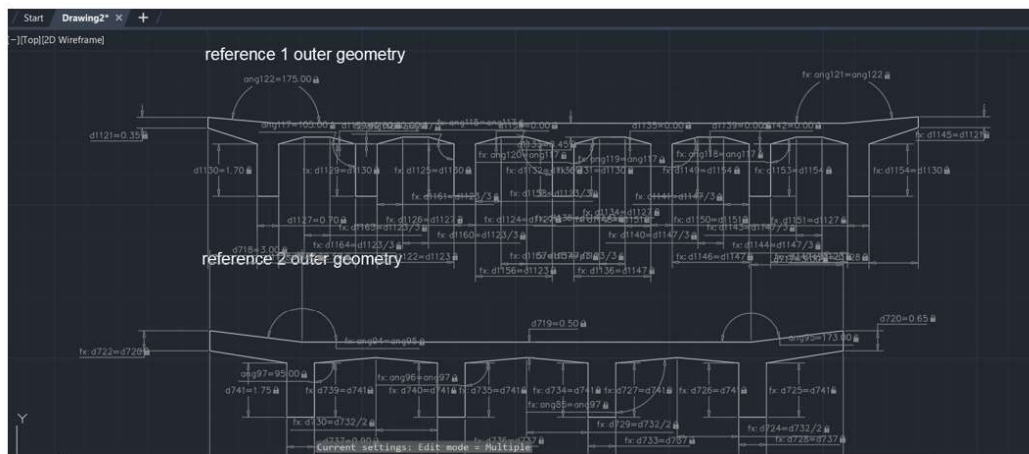


Figure 5.39: An example of parametric model of class "deck\_t\_shaped\_cross\_section" with geometric and dimensional constraints

The parametric models were first exported as .pdf files from AutoCAD, and then they were converted to .JPG files. The converted .JPG files are referred to as `syndata_original` in this thesis. After resizing the `syndata_original` dataset, it is referred to as `syndata_resized` which is then provided to Labellmg for data annotation process. Figure 5.45 shows an example of a synthetic technical drawing with annotated bounding boxes in Labellmg tool. Figure 5.46 shows another example of synthetic technical drawing displaying the augmented classes. The synthetic data is resized according to the procedure and algorithm 5.1 mentioned earlier in section 5.3.2. Similar to the bridge dataset, synthetic dataset was also assigned meaningful names, such as `syn_xx.JPG` where `syn` denotes that a technical drawing belongs to synthetic dataset and `_xx` denotes the image number sequentially. Assigning such meaningful names also makes the next step easier which is counting the number of instances. To count the number of instances of each of the classes



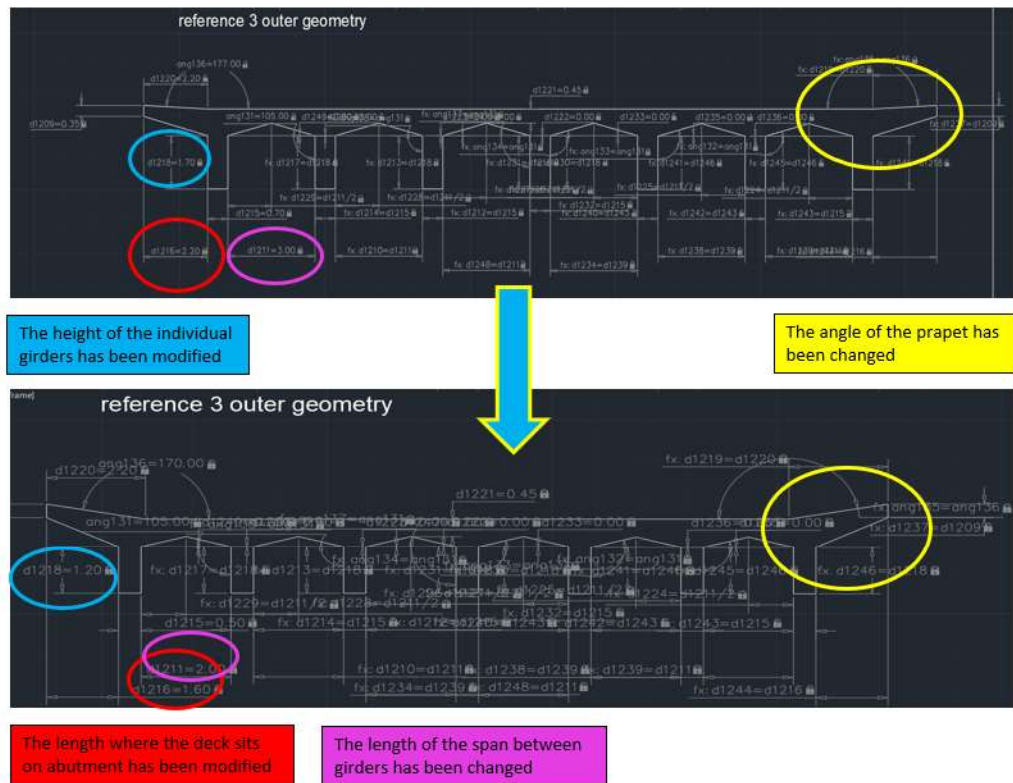


Figure 5.40: An example of parametric model of class "deck\_t\_shaped\_cross\_section" showing modifications performed to create another instance easily of the same class

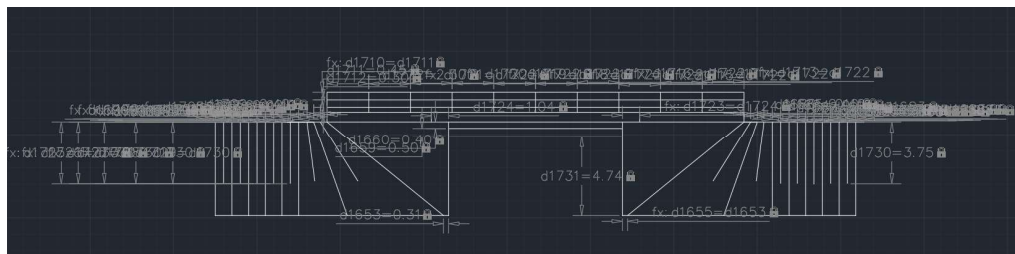


Figure 5.41: An example of parametric model of class "bridge\_side\_view" with geometric and dimensional constraints

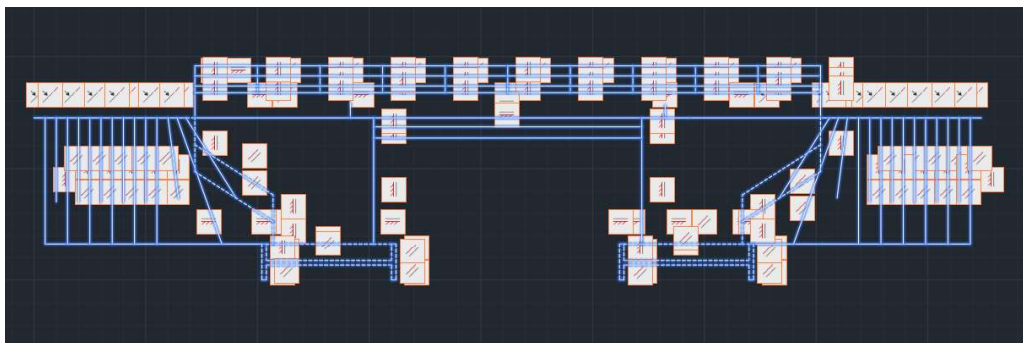


Figure 5.42: An example of parametric model of class "bridge\_side\_view" with geometric constraints



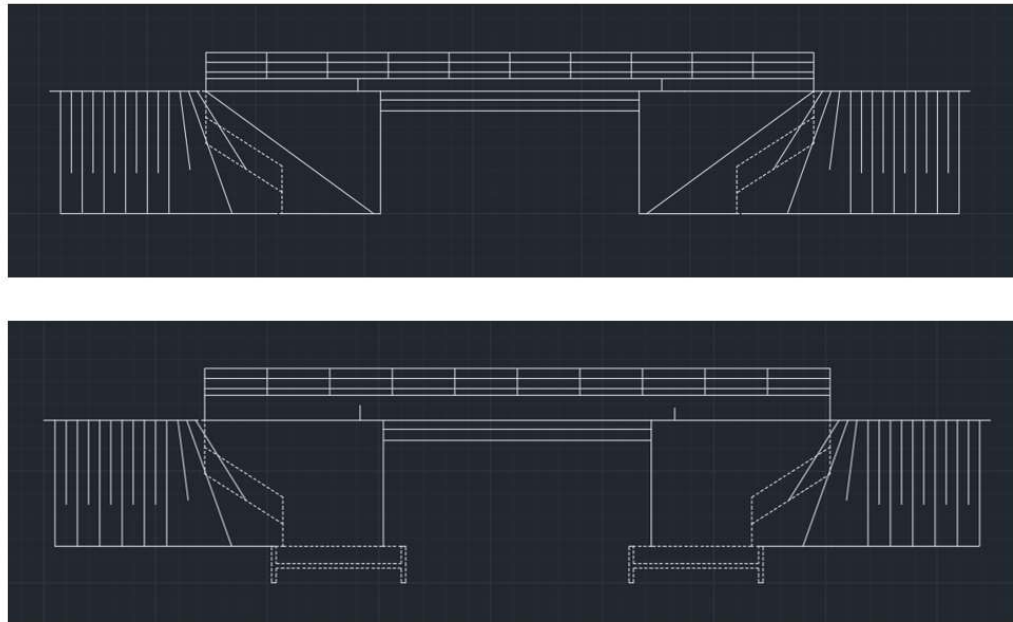


Figure 5.43: An example of parametric model of class "bridge\_side\_view" with geometric constraints

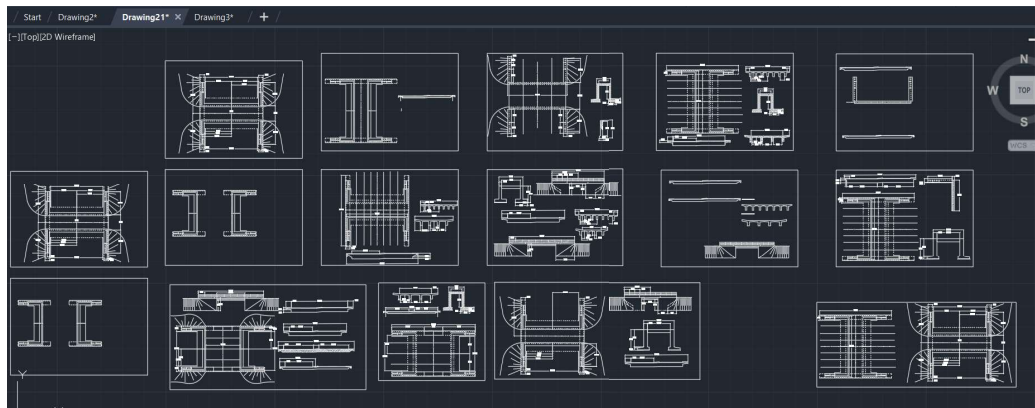


Figure 5.44: Parametric models of all of the insufficient classes are shown in this diagram with geometrical and dimensional constraints hidden

in synthetic dataset, algorithm 5.2 is executed again in the base directory and the location of annotated synthetic resized dataset (`syndata_resized`) is provided.

The total number of newly created instances of each of the insufficient classes is displayed in figure 5.47. After adding the synthetic data to the original dataset, the total number of instances for all of the 14 classes equals to 1213 instances as can be seen in figure 5.48. This is also summarized in form of a bar chart in figure 5.50. It should be noticed that the number of instances of each of the insufficient classes was increased by 30% to 40% only (with exception of frame class).

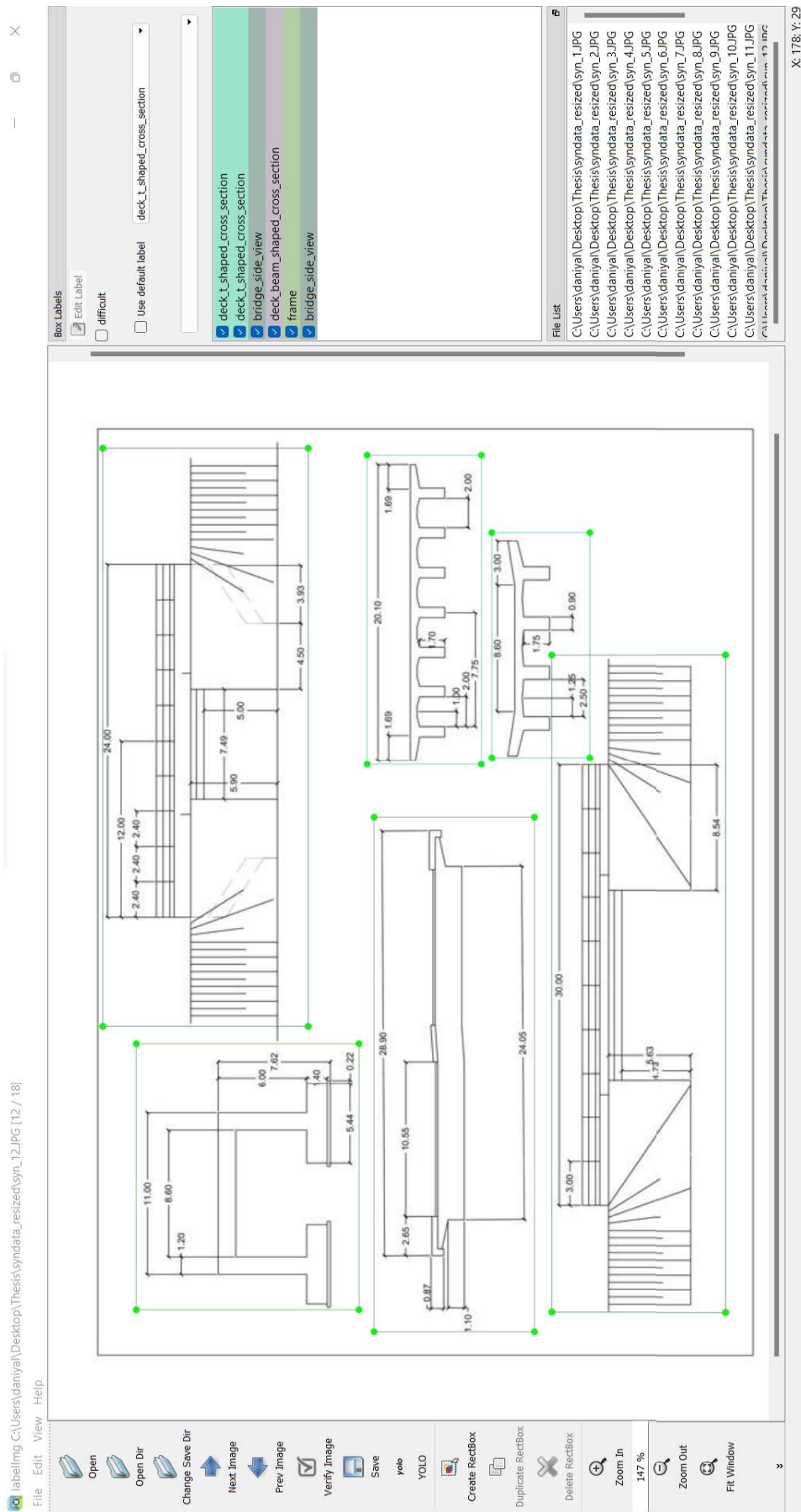


Figure 5.45: An example of annotated synthetic 2D technical drawing created using parametric modeling in the Labelling environment

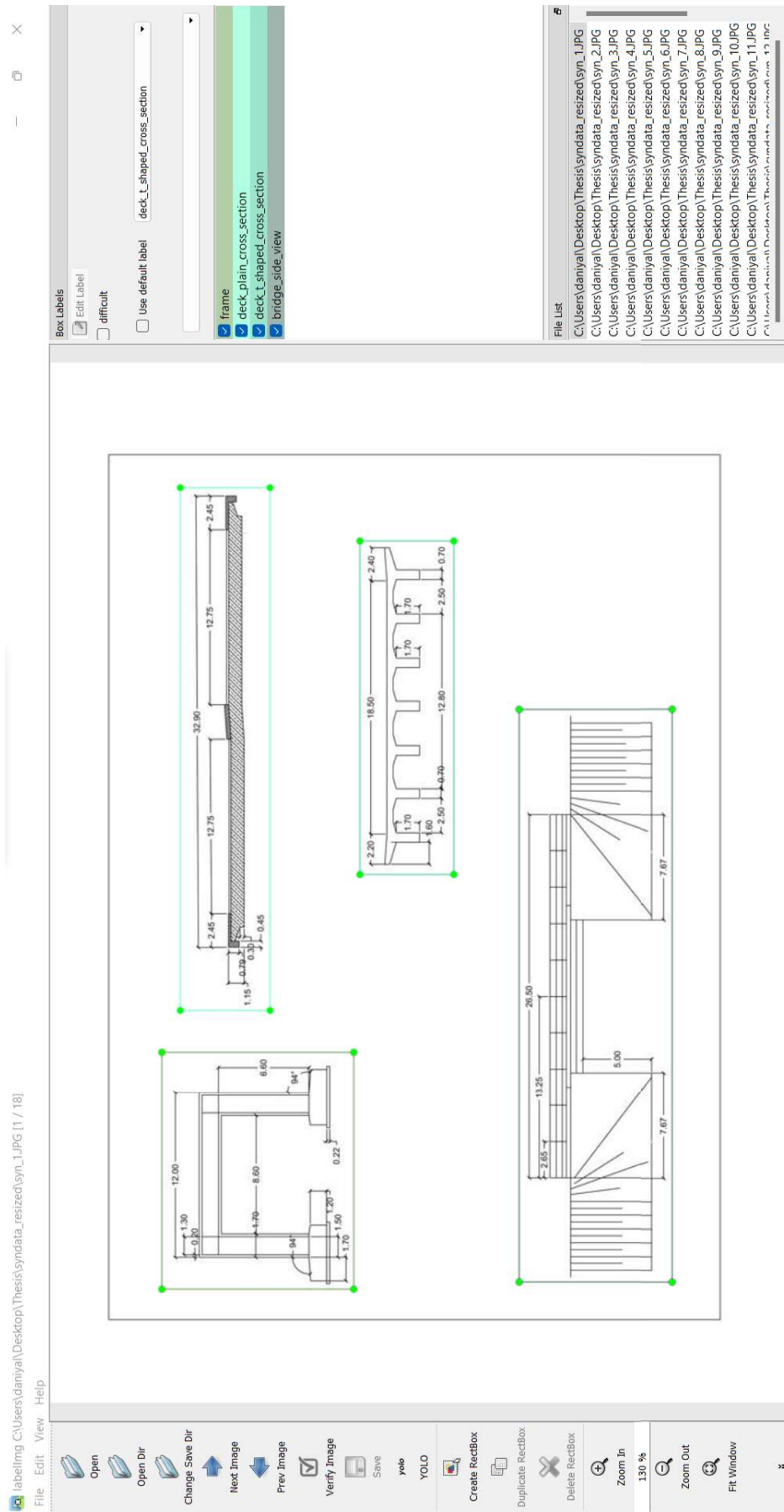


Figure 5.46: An example of synthetic 2D technical drawing with labeled bounding boxes in LabelImg data annotating tool

```

syn statistics
Class (0, deck_t_shaped_cross_section) --> Count: 15
Class (1, deck_beam_shaped_cross_section) --> Count: 8
Class (2, deck_plain_cross_section) --> Count: 8
Class (3, deck_top_view) --> Count: 7
Class (5, abutment_retaining_wall) --> Count: 3
Class (6, abutment_retaining_wall_cross_section) --> Count: 1
Class (7, abutment_top_view) --> Count: 1
Class (8, bridge_top_view) --> Count: 7
Class (9, bridge_side_view) --> Count: 9
Class (12, frame) --> Count: 12
Total --> Count: 71

```

Figure 5.47: Total number of synthetic instances for each of the classes as well as overall total

```

(base) C:\Users\daniyal\Desktop\Thesis> python count_instances_2.py
Overall statistics:
Class (0, deck_t_shaped_cross_section) --> Count: 35
Class (1, deck_beam_shaped_cross_section) --> Count: 29
Class (2, deck_plain_cross_section) --> Count: 28
Class (3, deck_top_view) --> Count: 26
Class (4, abutment_wing_wall) --> Count: 135
Class (5, abutment_retaining_wall) --> Count: 245
Class (6, abutment_retaining_wall_cross_section) --> Count: 88
Class (7, abutment_top_view) --> Count: 153
Class (8, bridge_top_view) --> Count: 29
Class (9, bridge_side_view) --> Count: 55
Class (10, foundation_top_view) --> Count: 119
Class (11, foundation_side_view) --> Count: 81
Class (12, frame) --> Count: 35
Class (13, table) --> Count: 155
Total --> Count: 1213

```

Figure 5.48: After adding the synthetic data to the original dataset, the total number of instances for all of the 14 classes equals to 1213 instances

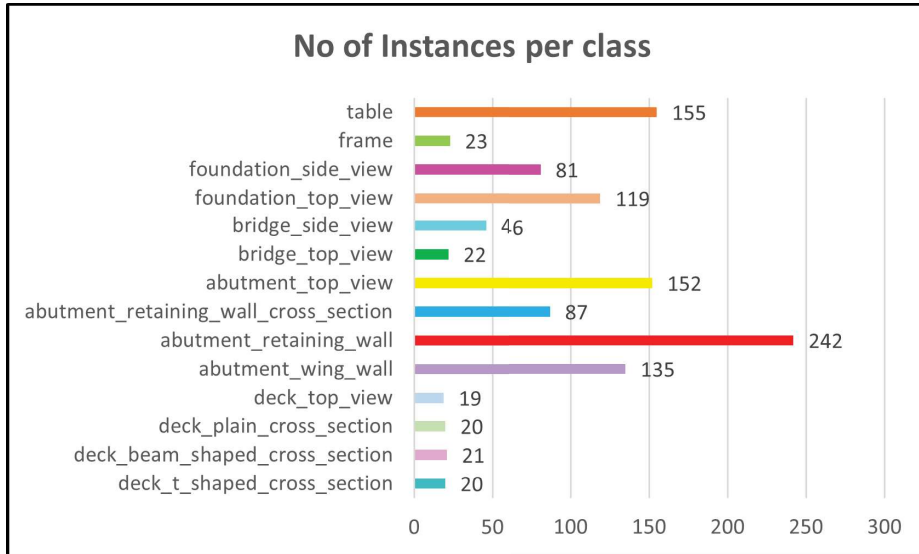


Figure 5.49: Revisiting the bar chart (also included earlier as figure 5.25 in section 5.3.4) displaying the total number of instances for each of the classes for a total of 15 bridges

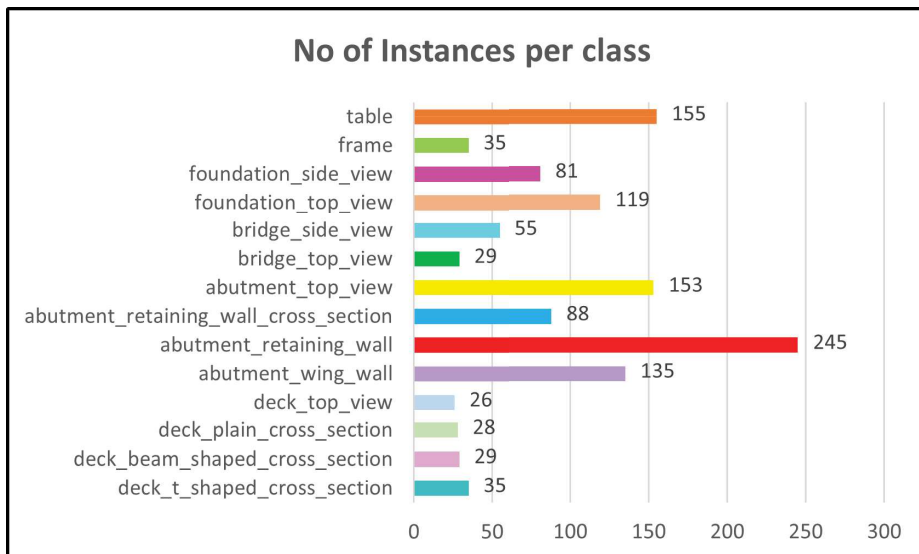


Figure 5.50: A bar chart showing the number of instances of each of the classes after labeling synthetic data from parametric modeling

## 5.6 Hyperparameters

The accuracy and performance of object detectors such as [YOLOv4](#) which is based on deep learning model are dependent on hyperparameters. The hyperparameters, in this research, were obtained through a trial and error process. The table 5.2 includes the hyperparameters that were modified for each training. Hyperparameters are usually modified to arrive at an optimum set of hyperparameters in terms of accuracy and performance.

Table 5.2: Different combinations of hyperparameters tested in this thesis

Case Description	input layer width	input layer height	batch size	subdivisions
First run	416	416	32	16
Second run	608	608	32	16
Third run	416	416	32	8
Synthetic run	608	608	32	16

The results of the cases mentioned above are summarized in chapter 6. Upon increasing the network resolution (`input_layer_width` and `input_layer_height`), changes in the values of evaluation parameters such as average loss, average [IOU](#), and [mAP](#) will be monitored, and an optimum combination of hyperparameters will then be selected to train the model over a larger number of iterations. All the cases (except the first run) were trained for 8000-9500 iterations, and the execution was stopped when no further improvements in the evaluation parameters were observed. The `first_run` case was run for only 3500 iterations.

Batch size was kept constant at 32 and not any larger value such as 64, which is often a norm, because the original bridge dataset consisted of only 175 technical drawings which had a total of 1142 instances. Batch size is the number of images that will be processed in one batch during one iteration, and trained weights get updated after each iteration. Whereas, subdivisions is the number of mini-batches a batch is split into. For instance, if the batch size is 32, and subdivision is 16, it means  $32/16 = 2$  images are sent for preprocessing. This process will then be performed 16 times until the batch is completed, and a new iteration will start with 32 new images. If memory-related issues are encountered during the training process, then a higher value of subdivisions must be set (16,32,64). If a higher subdivisions value is chosen, it takes a longer time to train since the number of images being loaded is reduced.



## Chapter 6

# Results

*This section includes the results of a trained model, evaluation parameters, and evaluation diagrams. The results of predicted bounding boxes on resized images will be shown in this section. The predicted bounding boxes coordinates are then exported and stored in a `.txt` file. This text file is then be used to convert the bounding box coordinates into [YOLO](#) format. This conversion makes it easier to scale-up the bounding boxes to map them on the original sized images since the bounding box coordinates are normalized with respect to the image. The scaled-up bounding boxes would then be shown, and the script to scale-up the bounding boxes is also included in this section. After the bounding boxes have been mapped on to the original sized images, the regions enclosed by the predicted bounding boxes or regions where objects are detected are cropped out of the original sized images. These cropped out regions are then stored as separate images with meaningful names to perform scene-text detection. Detection results on technical drawings would be shown on images from validation dataset and test dataset.*

### 6.1 Case 1: 416\_416\_32\_16

For case 1, where network resolution (input layer) width and height = 416, batchsize = 32, and subdivisions = 16, the object detection results based on the `yolov4-obj_best.weights` is shown in the figure. This was the initial case with the above mentioned hyperparameters, and it was run for just 3500 iterations.

Table 6.1: Hyperparameters for the first case have been highlighted in the table below

Case Description	input layer width	input layer height	batch size	subdivisions
First run	416	416	32	16
Second run	416	416	32	8
Third run	608	608	32	16
Synthetic run	608	608	32	16

### 6.2 Case 3: 608\_608\_32\_16

For case 3, the network resolution (input layer) width and height = 416, batchsize = 32, and subdivisions = 16. The diagram showing average loss and [mAP](#) curve with respect to number of iterations is shown in figure 6.2. This case was initially run for above 8000 iterations, and the convergence in average loss can be observed in figure 6.2 . At the same time, the values of [mAP](#) is seen to be converging. Later, after running the first three cases

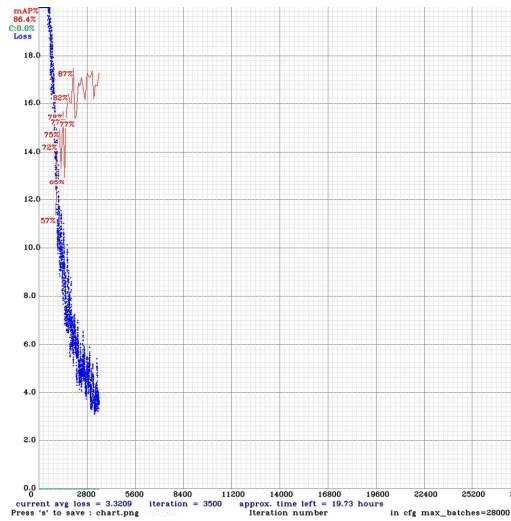


Figure 6.1: A loss and convergence diagram with respect to number of iterations

Table 6.2: Hyperparameters for the second case have been highlighted in the table below

Case Description	input layer width	input layer height	batch size	subdivisions
First run	416	416	32	16
Second run	416	416	32	8
Third run	608	608	32	16
Synthetic run	608	608	32	16

as shown in table 5.2, it was observed that the third case (this case) outperformed all other cases, and thus, this case was run for more iterations until 20000th iteration. This has resulted in an increase in the quality of trained weights and overall object detection results. The details on the other two cases (second run and synthetic run) will not be provided here, but they will only be discussed briefly in chapter 7. Technical drawings showing predicted bounding boxes are also shown in this section, and from the results, it seems that YOLOv4 has performed well in detection of elements in bridge technical drawings. To run and show multiple technical drawings within the console output with predicted bounding boxes, the following lines of code (algorithm 6.1) have to be executed. This display the technical drawings with predicted bounding boxes as well as download and save the detections in a specified folder on Google Drive. Figure 6.5 shows the "resized\_valid\_results" folder where all the detection results on validation technical drawings are stored.

Algorithm 6.1: Execute the following code to run detections on multiple images and display the predicted bounding boxes on technical drawings in the console output

```

1 from PIL import Image
2 import warnings
3 import os
4 import PIL
5 import glob
6
7 Image.MAX_IMAGE_PIXELS = None
8 warnings.simplefilter('ignore', Image.DecompressionBombWarning)

```

```

9
10
11 sourcePath = '/mydrive/validation'
12 targetPath = '/mydrive/results'
13 print(os.listdir(sourcePath))
14 test_images = [file for file in os.listdir(sourcePath) if file.endswith
    (('JPG', 'jpg', 'jpeg'))]
15 print(f'test_images_{test_images}')
16
17 for image in test_images :
18     os.system(f"./darknet_detector_test_data/obj.data_cfg/yolov4-obj.
        cfg_/mydrive/yolov4/backup/yolov4-obj_best.weights_{sourcePath
            }/{image}_ext_output")
19     imshow('predictions.jpg')
20     os.system(f"cp_predictions.jpg_{targetPath}/detect_results_{image
        }")

```

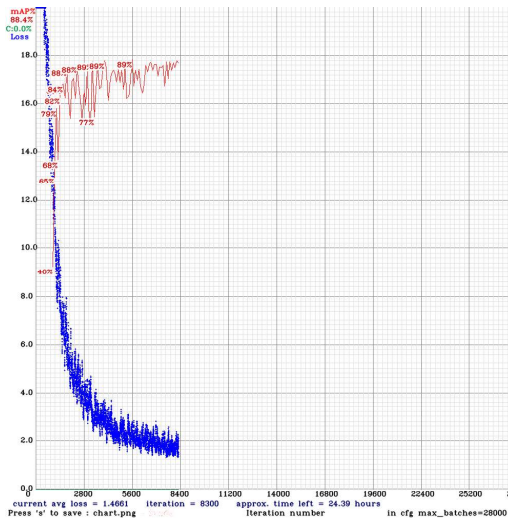


Figure 6.2: A loss and convergence diagram with respect to number of iterations

After receiving the technical drawings with predicted bounding boxes, the coordinates of those boxes were generated and stored in a .txt file. To receive the coordinates of predicted bounding boxes for multiple validation technical drawings, following lines of code have to be executed (algorithm 6.2).

Algorithm 6.2: Execute the following code to run detections on multiple images and display the coordinates of predicted bounding boxes in the console as well as export them to a .txt file

```

1 os.system(f"./darknet_detector_test_data/obj.data_cfg/yolov4-obj.cfg_/
    /mydrive/yolov4/backup/yolov4-obj_best.weights_-dont_show_-
    ext_output_{sourcePath}/mydrive/validation.txt_{image}_bbcoordinates.txt")
2 download('bbcoordinates.txt')

```

```

calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
20
detections_count = 154, unique_truth_count = 95
class_id = 0, name = deck_t_shaped_cross_section, ap = 100.00%      (TP = 2, FP = 0)
class_id = 1, name = deck_beam_shaped_cross_section, ap = 0.00%    (TP = 0, FP = 1)
class_id = 2, name = deck_plain_cross_section, ap = 100.00%      (TP = 1, FP = 0)
class_id = 3, name = deck_top_view, ap = 75.00%                  (TP = 3, FP = 0)
class_id = 4, name = abutment_wing_wall, ap = 100.00%            (TP = 8, FP = 0)
class_id = 5, name = abutment_retaining_wall, ap = 90.91%        (TP = 19, FP = 0)
class_id = 6, name = abutment_retaining_wall_cross_section, ap = 100.00% (TP = 5, FP = 0)
class_id = 7, name = abutment_top_view, ap = 100.00%            (TP = 7, FP = 1)
class_id = 8, name = bridge_top_view, ap = 100.00%              (TP = 3, FP = 0)
class_id = 9, name = bridge_side_view, ap = 100.00%            (TP = 5, FP = 2)
class_id = 10, name = foundation_top_view, ap = 97.98%          (TP = 8, FP = 0)
class_id = 11, name = foundation_side_view, ap = 90.91%         (TP = 9, FP = 0)
class_id = 12, name = frame, ap = 100.00%                       (TP = 2, FP = 0)
class_id = 13, name = table, ap = 93.33%                        (TP = 14, FP = 1)

for conf_thresh = 0.25, precision = 0.95, recall = 0.91, F1-score = 0.92
for conf_thresh = 0.25, TP = 86, FP = 5, FN = 9, average IoU = 79.13 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.891522, or 89.15 %
Total Detection Time: 1 Seconds

```

Figure 6.3: A console output showing the average precision values for each of the classes as well as the mean average precision (mAP) for the best weights received for the highlighted hyperparameters in table 6.2

## 6.2.1 Converting Predicted Bounding Boxes Coordinates into YOLO format

The content of the text file named "bbcoordinates.txt" is shown in figure 6.4. Only a few lines have been shown. The file shows coordinates of predicted bounding boxes (in yellow) in figure 6.4. However, the coordinates are not in the YOLO format. The coordinates are in the form of left\_x, top\_y width, and height. Moreover, only one .txt (bbcoordinates.txt) file is received which contains coordinates of all of the predicted bounding boxes in all of the validation images. In other words, for all the validation images, only one file is received from YOLO object detector. This file (figure 6.4) contains predicted bounding box coordinates for all of the validation images together. However, separate .txt files are required for each of the validation image. To parse the bbcoordinates.txt file and convert the coordinates of predicted bounding boxes in YOLO format, the equations 5.4 to 5.7 and the python script (algorithm 6.3) is used.

Algorithm 6.3: Execute the following code to convert predicted bounding boxes coordinates into YOLO format

```

1 from pathlib import Path
2 import re
3 from collections import defaultdict
4 import cv2
5
6 READ_PATH_TO_CLASSES = Path("classes/classes.txt")
7 READ_PATH_TO_RESULT = Path("results/bbcoordinates.txt")
8 WRITE_PATH_FOR_YOLO_PRED = Path("output/")
9 READ_DOWNSCALED_IMGS = Path("resized_valid/")

```

```

/mydrive/resized_valid/BW074-6_9.JPG: Predicted in 53.338000 milli-seconds.
abutment_top_view: 100% (left_x: 24 top_y: 251 width: 269 height: 284)
abutment_retaining_wall: 100% (left_x: 42 top_y: 448 width: 99 height: 214)
abutment_retaining_wall_cross_section: 100% (left_x: 496 top_y: 26 width: 62 height: 200)
abutment_retaining_wall: 100% (left_x: 568 top_y: 36 width: 299 height: 220)
abutment_wing_wall: 100% (left_x: 579 top_y: 290 width: 260 height: 246)
table: 100% (left_x: 847 top_y: 465 width: 157 height: 248)
Enter Image Path: Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
/mydrive/resized_valid/BW067-1_17.JPG: Predicted in 53.372000 milli-seconds.
abutment_wing_wall: 100% (left_x: 41 top_y: 72 width: 223 height: 144)
abutment_top_view: 100% (left_x: 43 top_y: 382 width: 266 height: 325)
abutment_retaining_wall_cross_section: 100% (left_x: 213 top_y: 232 width: 66 height: 148)
abutment_top_view: 100% (left_x: 253 top_y: 378 width: 300 height: 329)
abutment_retaining_wall: 100% (left_x: 307 top_y: 73 width: 277 height: 130)
abutment_retaining_wall_cross_section: 100% (left_x: 315 top_y: 241 width: 83 height: 142)
deck_top_view: 100% (left_x: 575 top_y: 376 width: 230 height: 328)
abutment_retaining_wall: 100% (left_x: 599 top_y: 86 width: 268 height: 114)
abutment_retaining_wall: 100% (left_x: 708 top_y: 233 width: 50 height: 144)
abutment_wing_wall: 100% (left_x: 913 top_y: 73 width: 200 height: 135)
table: 100% (left_x: 972 top_y: 459 width: 158 height: 254)

```

Coordinates of predicted bounding boxes in yellow

Figure 6.4: The content of bbcoordinates.txt file with predicted bounding box coordinates (yellow rectangles) and name of validation image shown in blue circle

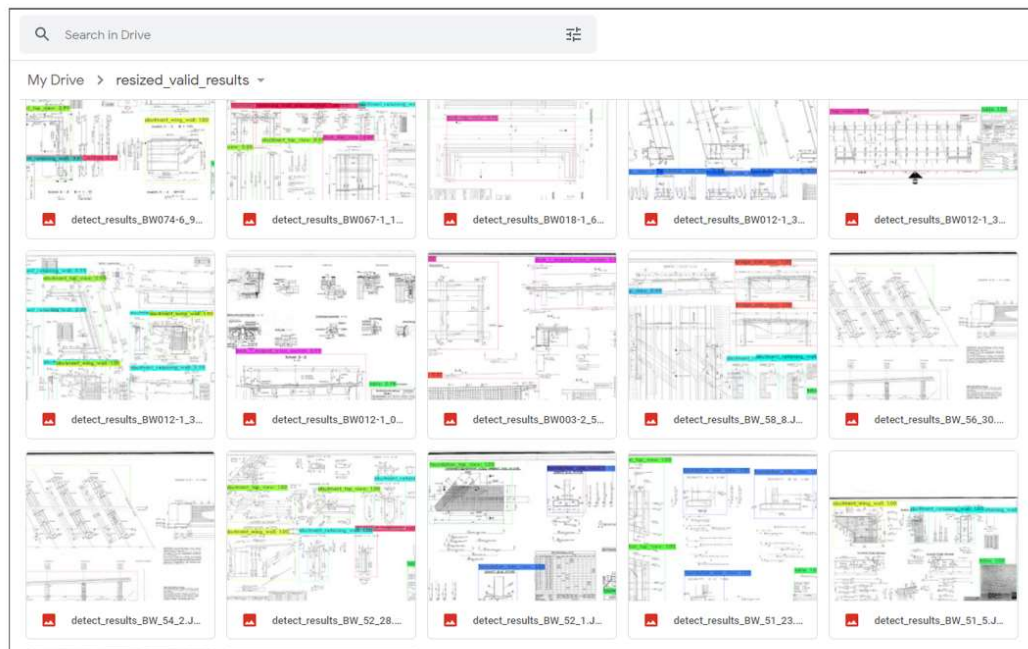


Figure 6.5: The content of "resized\_valid\_results" folder after running the test detector on validation images

```

10
11 READ_EXTENSION = ".JPG"
12
13
14 def class_to_ids(classes):
15     ''' Assigns class_name_to_ids. Negative_1 indicates invalid '''
16     idx = 0
17     class_ids = defaultdict(lambda: -1)
18     for class_name in classes:
19         if class_name:

```



```

20         class_ids[class_name] = idx
21         idx += 1
22
23     return class_ids
24
25
26 if __name__ == "__main__":
27     # Read the classes
28     class_ids = None
29     with open(READ_PATH_TO_CLASSES) as class_file:
30         class_data = class_file.read().split('\n')
31         class_ids = class_to_ids(class_data)
32
33     # Read the result file and dump them to separate files
34     with open(READ_PATH_TO_RESULT) as result_file:
35         result_data = result_file.read().split("Enter_Image_Path")
36
37     # Discard the header as it is not required
38     result_data = result_data[1:]
39
40     for detect_block in result_data:
41         # Discard the first 3 lines as they are not necessary:
42         detect_block = detect_block.split('\n')[3:]
43
44     if detect_block:
45         # Get the name of the file from the first line of the
46         block
47         file_name = re.split(READ_EXTENSION, detect_block[0],
48                               flags=re.IGNORECASE)[0].split("/)[-1]
49
50         # Get the prediction image dimensions
51         img = cv2.imread(str(READ_DOWNSCALED_IMGS.joinpath(
52                               file_name + READ_EXTENSION)))
53         height, width, _ = img.shape
54
55         file_data = []
56         # For each file prediction, create a new yolo
57         compatible file
58         for row in detect_block[1:]:
59             # Strip round brackets from the string and split it
60             at colon
61             row_clean = re.sub("[()]", "", row).split(":")
62
63             if row_clean and row_clean[0]:
64                 yolo_pred = []
65                 # Prepare yolo format string
66                 yolo_pred.append(int(class_ids[row_clean[0]]))
67                 # Class id

```



```

62         # yolo_pred.append(class_ids[pred_clean[1].
           split()[0]]) # Class probability
63     yolo_pred.append(int(row_clean[2].split()[0]))
           # x
64     yolo_pred.append(int(row_clean[3].split()[0]))
           # y
65     yolo_pred.append(int(row_clean[4].split()[0]))
           # width
66     yolo_pred.append(int(row_clean[5].split()[0]))
           # height

67
68     yolo_str = []
69     yolo_str.append(str(yolo_pred[0]))
70     yolo_str.append(str((yolo_pred[1] + yolo_pred
71         [3]/2)/(width)))
72     yolo_str.append(str((yolo_pred[2] + yolo_pred
73         [4]/2)/(height)))
74     yolo_str.append(str(yolo_pred[3] / width))
75     yolo_str.append(str(yolo_pred[4] / height))
76
77     file_data.append(" ".join(yolo_str))
78
79     # Finally, save the prediction for each image into
80     # separate file
81     with open(WRITE_PATH_FOR_YOLO_PRED.joinpath(file_name +
82         ".txt"), mode="w") as pred_file:
83         for pred in file_data:
84             pred_file.write(f"{pred}\n")

```

---

## 6.2.2 Scaling-up Predicted Bounding Boxes

In this section, the predicted bounding boxes would now be scaled-up to map the predicted bounding boxes to the original sized images. Initially, the original sized images were resized to smaller images since running **YOLO** on original sized images caused memory-related issues. Moreover, to use the original sized images even during the testing phase also caused a long time to process and predict detections. That is why it was necessary to resize the images. The testing and validation was performed on the resized images, and the predicted bounding boxes were displayed on the resized validation images initially. Some of the results of predicted bounding boxes on resized validation images from folder "resized\_valid\_results" (shown in figure 6.5) will now be shown. The figures from figure 6.6 to 6.11 shows the results of object detection with predicted bounding boxes around the classes of interest as well as their class probability. In addition, the scaling-up of predicted bounding boxes, displaying them on original sized images, and cropping out the bounded region were necessary steps because the dimensions and other useful text in resized images were difficult to read as they were got blurred during the resize process. With

predicted bounding boxes being displayed on original sized images, the text could be easily read during scene-text detection process.

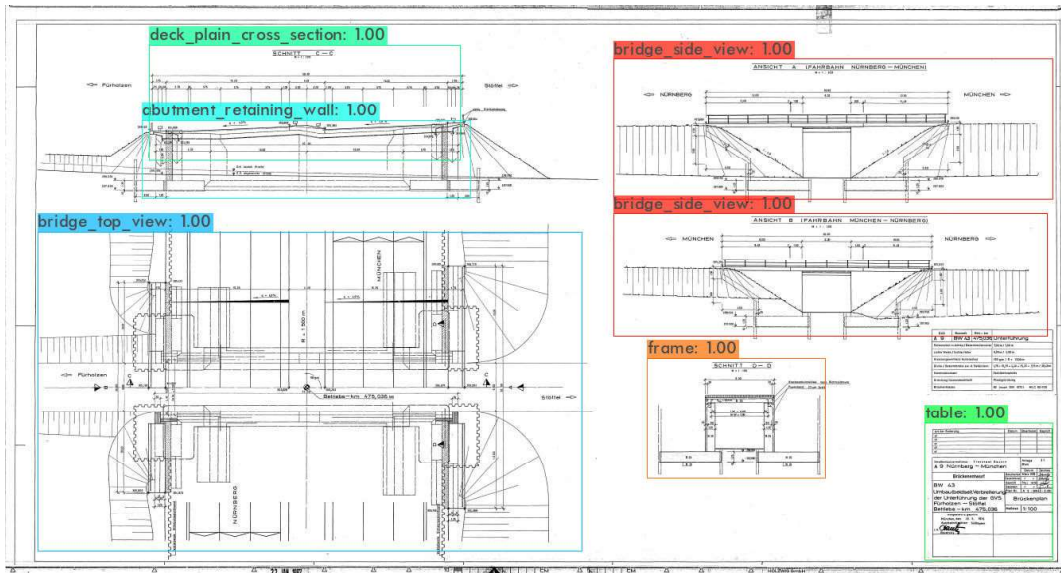


Figure 6.6: Object Detection results on a bridge technical drawing named as BW\_43\_21

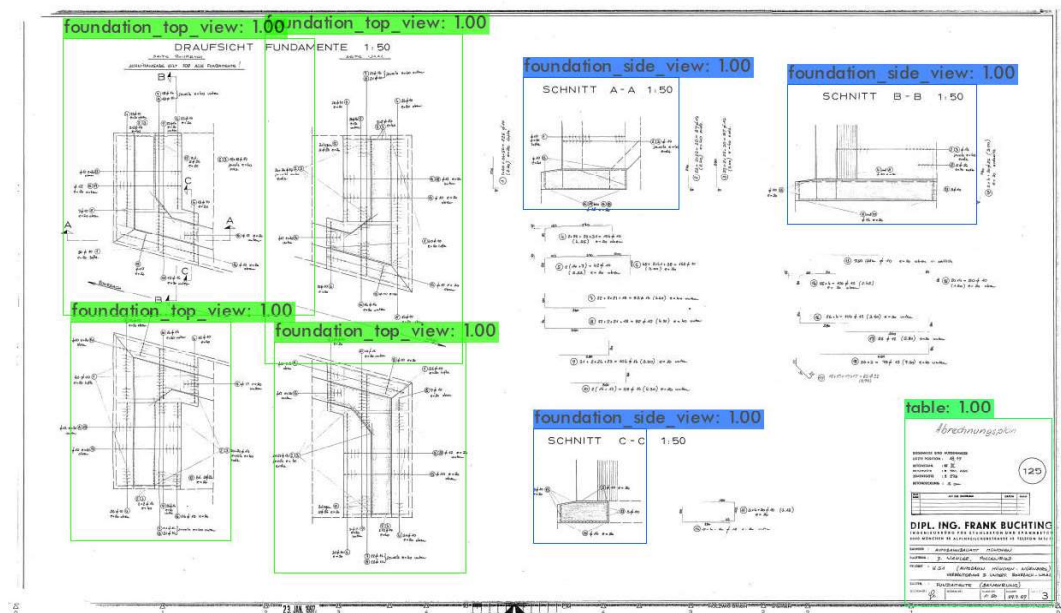


Figure 6.7: Object Detection results on a bridge technical drawing named as BW\_51\_23, showing the classes "foundation\_top\_view" and "foundation\_side\_view" identified

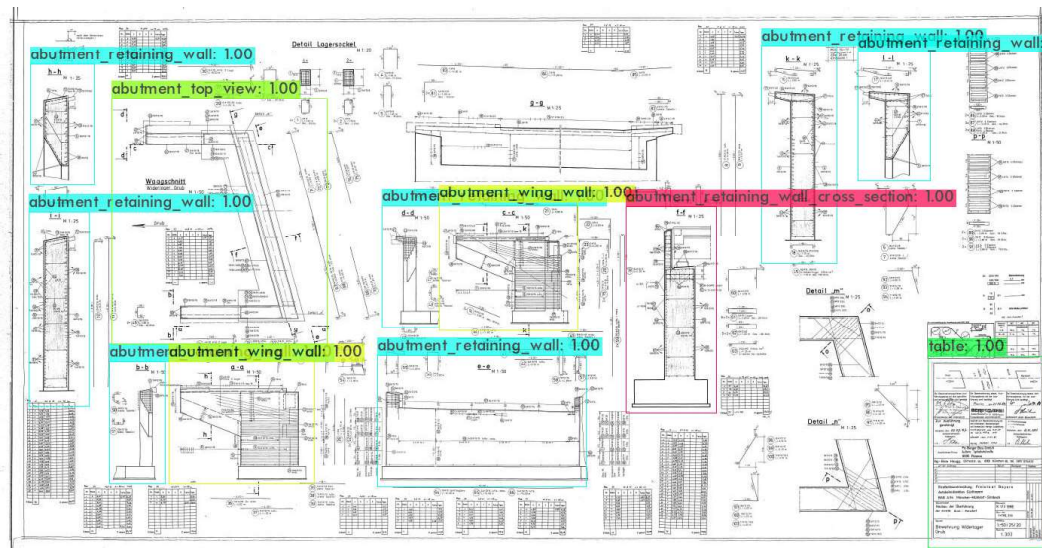


Figure 6.8: Object Detection results on a bridge technical drawing named as BW012-1\_3, showing the classes "abutment\_top\_view", "abutment\_retaining\_wall\_cross\_section", "abutment\_wing\_wall", "abutment\_retaining\_wall", and "table" identified

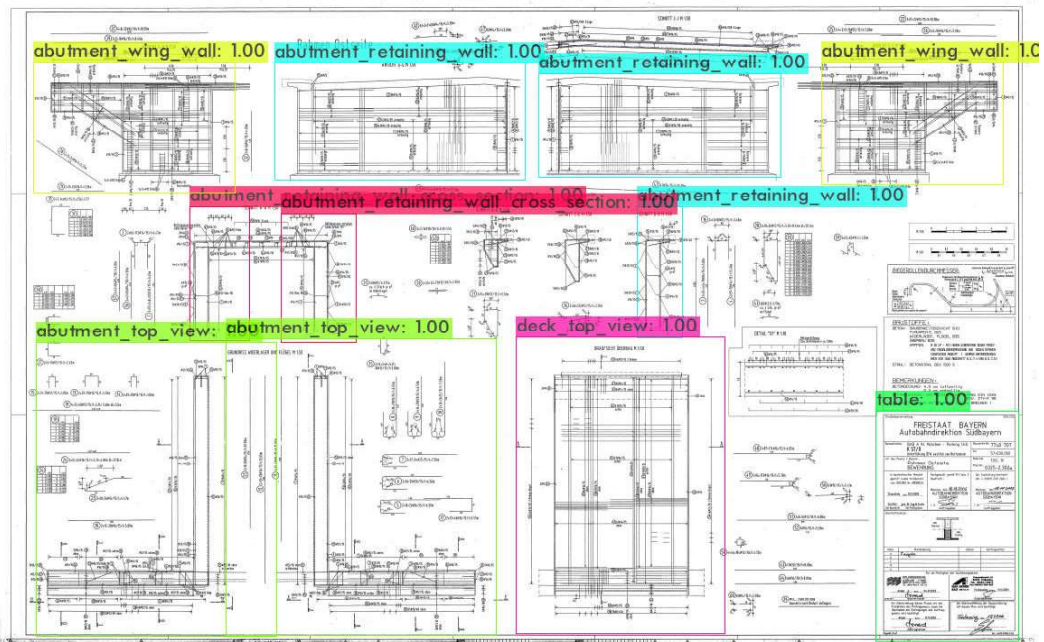


Figure 6.9: Object Detection results on a bridge technical drawing named as BW067-1\_17, showing the classes "abutment\_top\_view", "abutment\_retaining\_wall\_cross\_section", "abutment\_wing\_wall", "abutment\_retaining\_wall", "deck\_top\_view", and "table" identified

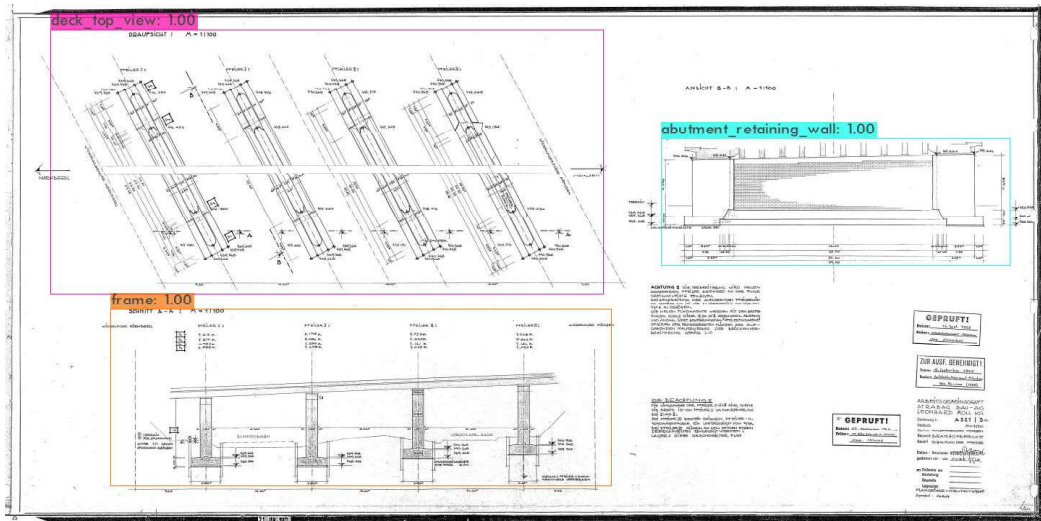


Figure 6.10: Object Detection results on a bridge technical drawing named as BW\_56\_30, showing the classes "deck\_top\_view", "abutment\_retaining\_wall", and "frame" identified

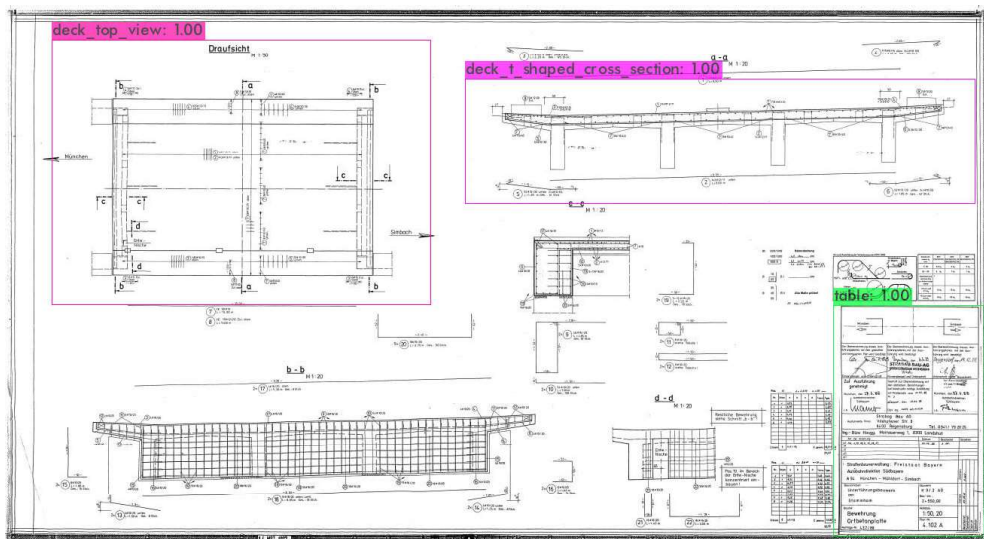
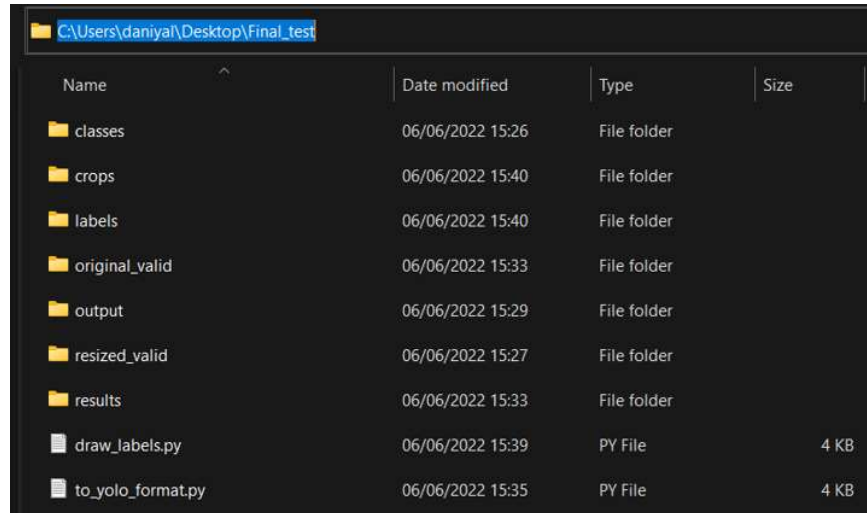


Figure 6.11: Object Detection results on a bridge technical drawing named as BW\_56\_30, showing the classes "deck\_top\_view", "deck\_tshaped\_cross\_section", and "table" identified



After receiving predictions on the resized validation technical drawings, the process of scaling-up of bounding boxes will now be performed. Please note that correct paths of the folders have to be provided, and for the subsequent postprocessing steps, the folder structure is shown in figure 6.12. Now all the folders and their contents will be explained here:



Name	Date modified	Type	Size
classes	06/06/2022 15:26	File folder	
crops	06/06/2022 15:40	File folder	
labels	06/06/2022 15:40	File folder	
original_valid	06/06/2022 15:33	File folder	
output	06/06/2022 15:29	File folder	
resized_valid	06/06/2022 15:27	File folder	
results	06/06/2022 15:33	File folder	
draw_labels.py	06/06/2022 15:39	PY File	4 KB
to_yolo_format.py	06/06/2022 15:35	PY File	4 KB

Figure 6.12: The folders structure for subsequent postprocessing steps

***classes folder:***

This folder contains `classes.txt` file to identify classes and write labels for scaled-up bounding boxes.

***results folder:***

This folder contains the `bbcoordinates.txt` file which is the text file in which predicted bounding box coordinates are present. The path of this file was then provided as an input parameter in algorithm 6.3 to convert the predicted bounding boxes coordinates into YOLO format.

***output folder:***

This folder contains separate `.txt` files for each of the validation technical drawings after running the algorithm 6.3 which converts the detected bounding boxes coordinates in `bbcoordinates.txt` file to YOLO format. The algorithm 6.3 also parses the `bbcoordinates.txt` file so that each of the validation technical drawings has a separate `.txt` file containing detected bounding box coordinates in YOLO format. The content of the output folder is shown in figure 6.13. The predicted bounding boxes `.txt` files in this folder are then used along with the original sized images to scale-up the bounding boxes using the algorithm 6.4 and display the bounding boxes on the original sized validation images to crop out the regions bounded by the detected bounding boxes. The cropped out regions are then provided to CRAFT algorithm (BAEK et al., 2019), which is a scene text detection model based on neural networks.

Name	Date modified	Type	Size
BW_40_13.txt	06/06/2022 15:35	Text Document	1 KB
BW_43_21.txt	06/06/2022 15:35	Text Document	1 KB
BW_51_5.txt	06/06/2022 15:35	Text Document	1 KB
BW_51_23.txt	06/06/2022 15:35	Text Document	1 KB
BW_52_1.txt	06/06/2022 15:35	Text Document	1 KB
BW_52_28.txt	06/06/2022 15:35	Text Document	1 KB
BW_54_2.txt	06/06/2022 15:35	Text Document	1 KB
BW_56_30.txt	06/06/2022 15:35	Text Document	1 KB
BW_58_8.txt	06/06/2022 15:35	Text Document	1 KB
BW003-2_5.txt	06/06/2022 15:35	Text Document	1 KB
BW012-1_0.txt	06/06/2022 15:35	Text Document	1 KB
BW012-1_3.txt	06/06/2022 15:35	Text Document	1 KB
BW012-1_31.txt	06/06/2022 15:35	Text Document	1 KB
BW012-1_33.txt	06/06/2022 15:35	Text Document	1 KB
BW018-1_6.txt	06/06/2022 15:35	Text Document	1 KB
BW067-1_17.txt	06/06/2022 15:35	Text Document	1 KB
BW074-6_9.txt	06/06/2022 15:35	Text Document	1 KB

Figure 6.13: The content of output folder

***original valid:***

This folder contains the original sized images to be used in scaling-up the bounding boxes. Since the images were resized to smaller images while keeping the aspect ratio constant, the scaled-up detected bounding boxes map at the exact desired location.

***resized valid:***

This folder contains the resized images to be used in scaling-up the bounding boxes. Since the images were resized to smaller images while keeping the aspect ratio constant, the scaled-up detected bounding boxes map at the exact desired location.

***labels:***

The files in this folder are generated after running the algorithm 6.4. It contains the original sized images with predicted bounding boxes generated on them with their class labels. Figure 6.14 shows predicted scaled-up bounding boxes as well as their labels on the original image. Moreover, the content of the labels folder is shown in figure 6.16. Please, notice the large size of the original images on which the predicted scaled-up bounding boxes are displayed.

***crops:***

This folder contains all the cropped regions extracted out from the original sized images after displaying the scale-up bounding boxes on them to further process for scene-text detection. The contents of the crop folder is shown in figure 6.17. Here also meaningful names are provided to each of the cropped image. The name is defined as `bridgename_classname_instancenumber`.



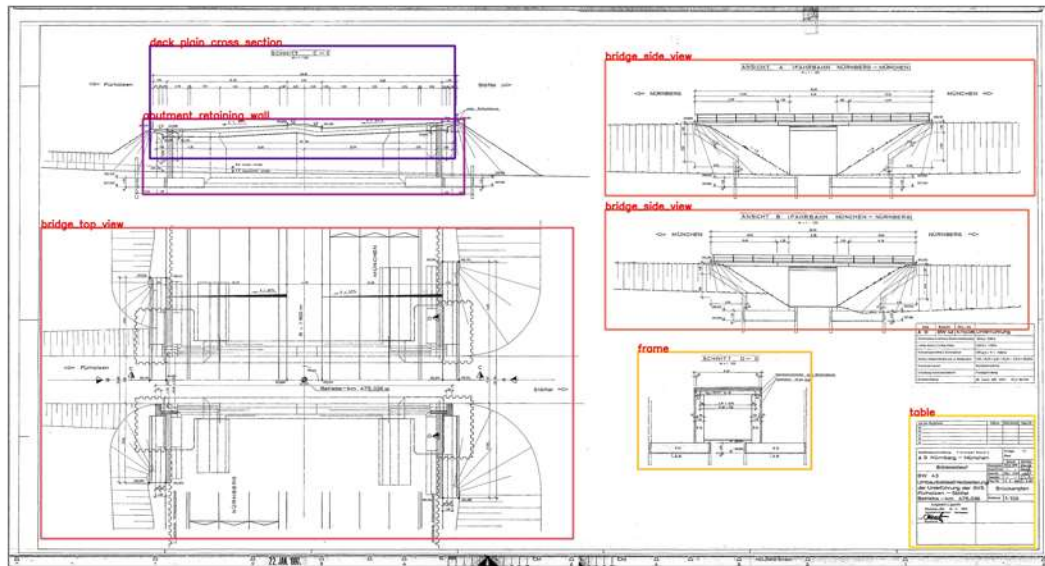


Figure 6.14: The original sized images corresponding to the resized image in figure 6.6

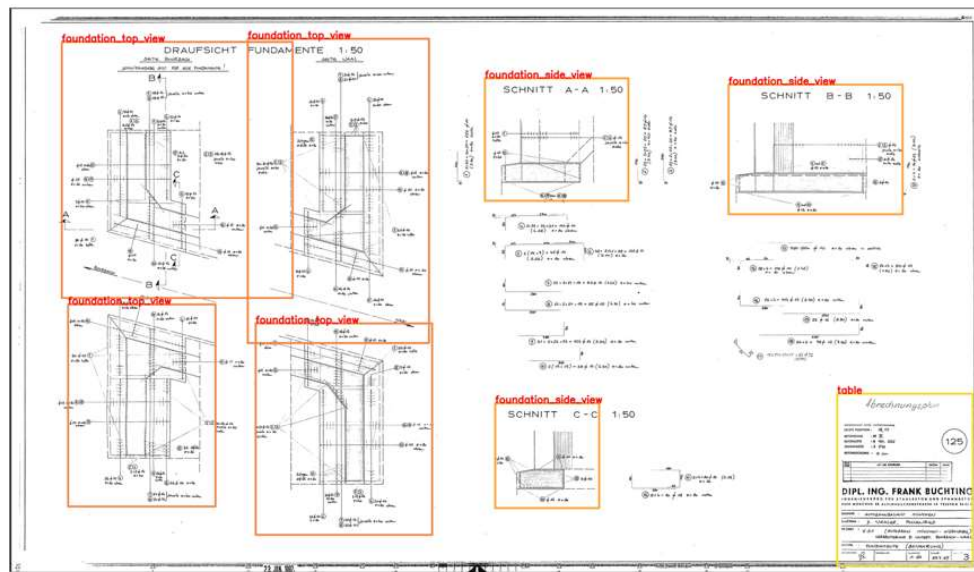


Figure 6.15: The original sized images corresponding to the resized image in figure 6.7

Algorithm 6.4: Execute the following code to scale-up the bounding boxes to be displayed on the original sized images and then crop out the regions enclosed by the predicted bounding boxes

```

1  from pathlib import Path
2  import glob
3  import os
4  import cv2
5  import csv
6  from matplotlib import cm
7

```

Name	Date	Type	Size	Tags
BW_40_13.JPG	06/06/2022 16:47	JPG File	7.950 KB	
BW_43_21.JPG	06/06/2022 16:47	JPG File	9.090 KB	
BW_51_5.JPG	06/06/2022 16:47	JPG File	13.324 KB	
BW_51_23.JPG	06/06/2022 16:47	JPG File	7.677 KB	
BW_52_1.JPG	06/06/2022 16:47	JPG File	6.493 KB	
BW_52_28.JPG	06/06/2022 16:47	JPG File	10.297 KB	
BW_54_2.JPG	06/06/2022 16:47	JPG File	6.008 KB	
BW_56_30.JPG	06/06/2022 16:47	JPG File	10.138 KB	
BW_58_8.JPG	06/06/2022 16:47	JPG File	9.461 KB	
BW003-2_5.JPG	06/06/2022 16:46	JPG File	9.261 KB	
BW012-1_0.JPG	06/06/2022 16:46	JPG File	7.667 KB	
BW012-1_3.JPG	06/06/2022 16:46	JPG File	17.539 KB	
BW012-1_31.JPG	06/06/2022 16:47	JPG File	5.681 KB	
BW012-1_33.JPG	06/06/2022 16:47	JPG File	11.245 KB	
BW018-1_6.JPG	06/06/2022 16:47	JPG File	9.059 KB	
BW067-1_17.JPG	06/06/2022 16:47	JPG File	15.116 KB	
BW074-6_9.JPG	06/06/2022 16:47	JPG File	14.605 KB	

Figure 6.16: The content of output folder

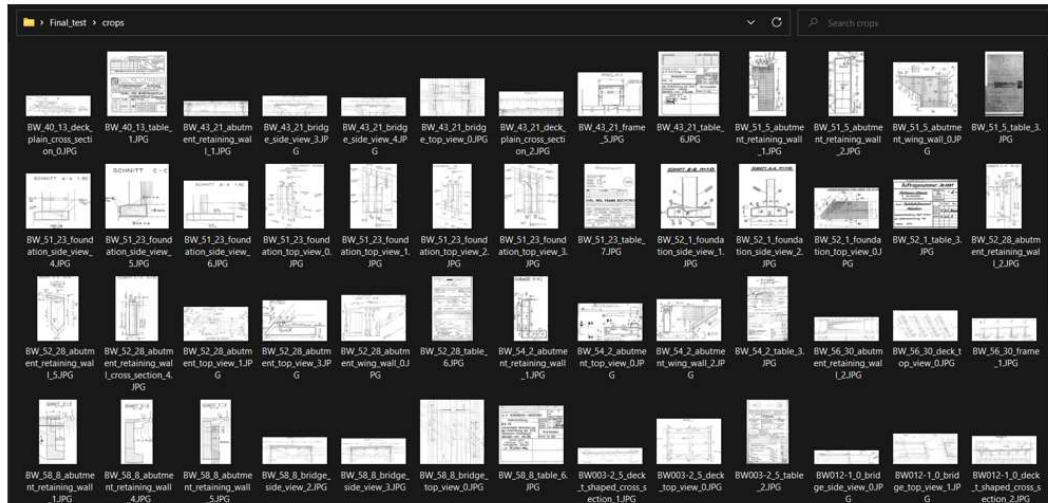


Figure 6.17: The content of crops folder

```

8
9 READ_PATH_TO_CLASSES = Path("classes/classes.txt")
10 READ_PATH_FOR_YOLO_PRED = Path("output/")
11 READ_IMGS_DIR = Path("original_valid/")
12 WRITE_LABELLED_IMGS_DIR = Path("labels/")
13 WRITE_CROPPED_IMGS = Path("crops/")
14
15 EXTENSION = ".JPG"

```

```

16
17
18 def insensitive_glob(pattern):
19     def either(c):
20         return "[%s%s]" % (c.lower(), c.upper()) if c.isalpha() else c
21     return glob.glob(''.join(map(either, pattern)))
22
23
24 def draw_bb(img, x0, y0, x1, y1, label, color, line_thickness = 30):
25     img = cv2.rectangle(img, (x0, y0), (x1, y1), color, line_thickness)
26
27     (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.6,
28     1)
29     img = cv2.rectangle(img, (x0, y0 - 20), (x0 + w, y0), color, -1)
30     img = cv2.putText(img, label, (x0, y0 - 5), cv2.
31     FONT_HERSHEY_SIMPLEX, 5, (0,0,255), 20)
32
33
34 def clamp(n, smallest, largest):
35     return max(smallest, min(n, largest))
36
37
38 def get_bgr(rgba_normalized):
39     return (
40         int(rgba_normalized[2] * 255),
41         int(rgba_normalized[1] * 255),
42         int(rgba_normalized[0] * 255),
43     )
44
45 if __name__ == "__main__":
46     # Get the classes names
47     classes = []
48
49     with open(READ_PATH_TO_CLASSES) as class_file:
50         classes = class_file.read().split('\n')
51
52     # Create the colormap, see more options here. Required for
53     # distinct colors
54     # https://matplotlib.org/3.5.0/tutorials/colors/colormaps.html
55
56     cmap = cm.get_cmap("plasma")
57
58     # Get all the yolo prediction text files names
59     pred_files = insensitive_glob(f"{READ_PATH_FOR_YOLO_PRED}/*.txt")
60
61     # Get just the base file names without extensions
62     pred_base_names = list(map(lambda x: os.path.basename(x).split('.')[0], pred_files))

```

```

60
61 # For each image, we draw the bounding box
62 for idx, file_path in enumerate(pred_files):
63     # Open the corresponding image
64     img_name = pred_base_names[idx] + EXTENSION
65     img = cv2.imread(str(READ_IMGS_DIR.joinpath(img_name)))
66     org_img = img.copy()
67     height, width, _ = img.shape
68
69     # Read the predictions text file
70     with open(file_path) as csv_file:
71         csv_reader = csv.reader(csv_file, delimiter=',')
72         counter = 0
73
74         for row in csv_reader:
75             if row:
76                 cls_id = int(row[0])
77                 label = classes[cls_id]
78                 cx = int(float(row[1]) * width)
79                 cy = int(float(row[2]) * height)
80                 bb_width_half = int(float(row[3]) * width / 2)
81                 bb_height_half = int(float(row[4]) * height / 2)
82
83                 x0 = clamp(cx - bb_width_half, 0, width - 1)
84                 y0 = clamp(cy - bb_height_half, 0, height - 1)
85                 x1 = clamp(cx + bb_width_half, 0, width - 1)
86                 y1 = clamp(cy + bb_height_half, 0, height - 1)
87
88                 draw_bb(img, x0, y0, x1, y1, label, get_bgr(cmap(
89                     cls_id / (len(classes) - 1))))
90
91                 # Save the ROI to file
92                 cv2.imwrite(str(WRITE_CROPPED_IMGS.joinpath(f"{
93                     pred_base_names[idx]}_{label}_{counter}{
94                     EXTENSION}")), org_img[y0:y1, x0:x1])
95                 counter += 1
96
97     # Save the image after drawing bounding boxes
98     cv2.imwrite(str(WRITE_LABELLED_IMGS_DIR.joinpath(img_name)),
99                 img)

```

---

### 6.2.3 Scene Text Detection

In this section, scene text detection is performed on the cropped regions extracted from the original sized images. This task is performed by utilizing a deep learning based [CRAFT](#) scene text detection algorithm to extract useful information related to the bridge elements

in technical drawings (see algorithm 6.5. The information could be textual or dimensional. The algorithm is a PyTorch implementation of [CRAFT](#).

The result of [CRAFT](#) are stored in a folder called `scenedetect`, and the contents of the folder is shown in figure 6.18. Furthermore, the bounding boxes around the text are cropped out to extract the useful textual and dimensional information as shown in figure 6.19.

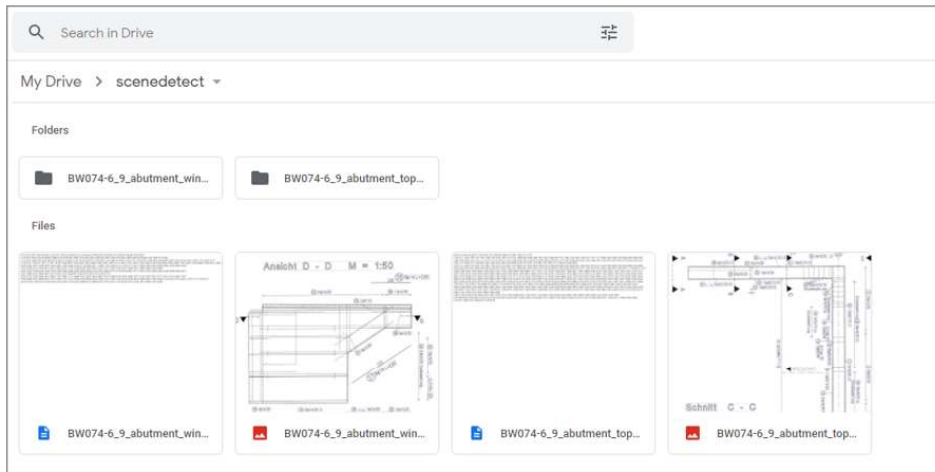


Figure 6.18: The content of `scenetext` folder after running [CRAFT](#)

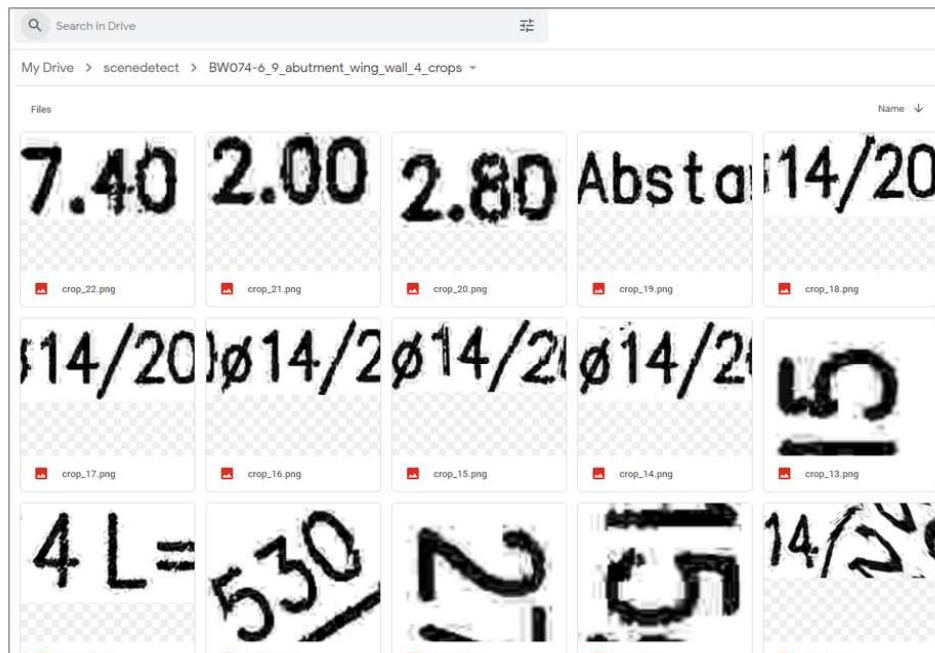


Figure 6.19: The content of the subfolder of a cropped region after running [CRAFT](#) displaying further the cropped regions extracted from bounding boxes around the textual information

Algorithm 6.5: Execute the following code to performed scene text detection using [CRAFT](#)

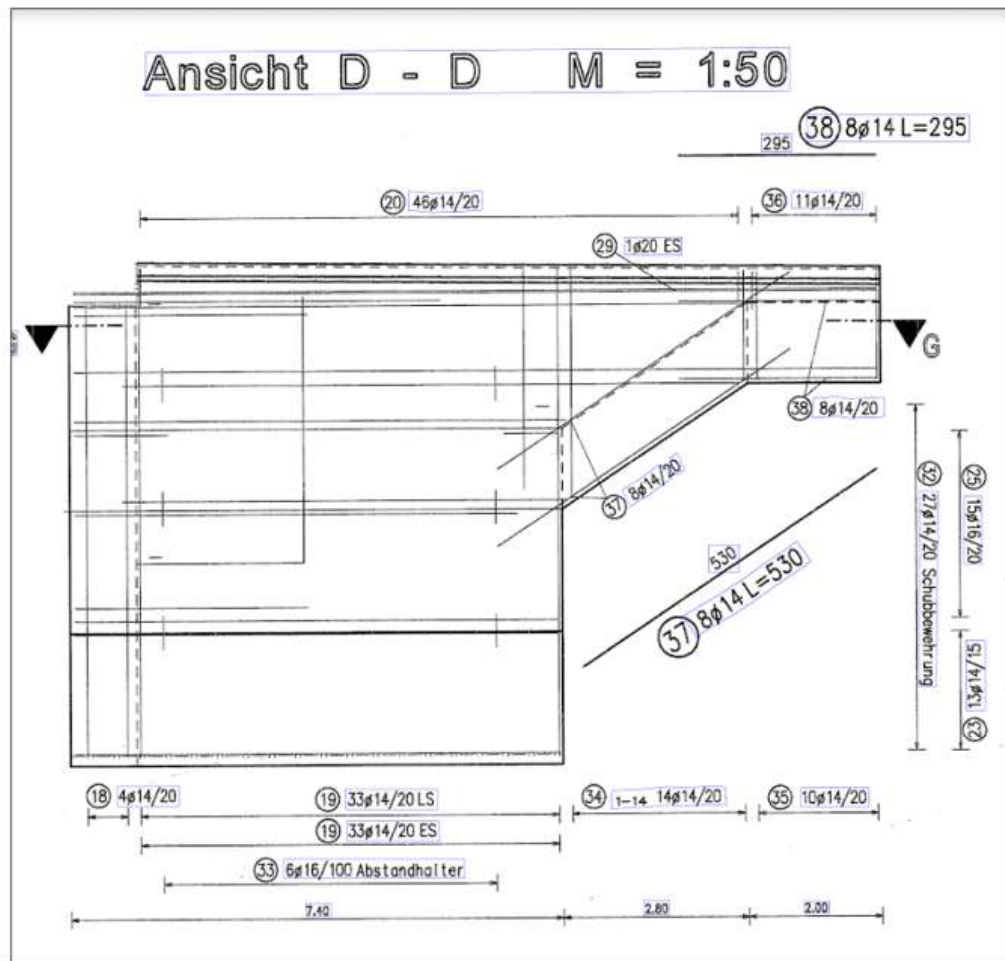


Figure 6.20: Scene text detection performed on cropped image showing the class "abutment\_wing\_wall"

```

1 pip install craft-text-detector
2
3 from google.colab import drive
4 drive.mount('/content/gdrive')
5
6 !In -s /content/gdrive/My\ Drive/ /mydrive
7 !ls /mydrive
8
9 !pwd
10 %cd /mydrive
11
12 # import Craft class
13 from craft_text_detector import Craft
14
15 # set image path and export folder directory
16 image = 'crops/BW074-6_9_abutment_top_view_0.JPG' # can be filepath ,
           PIL image or numpy array

```



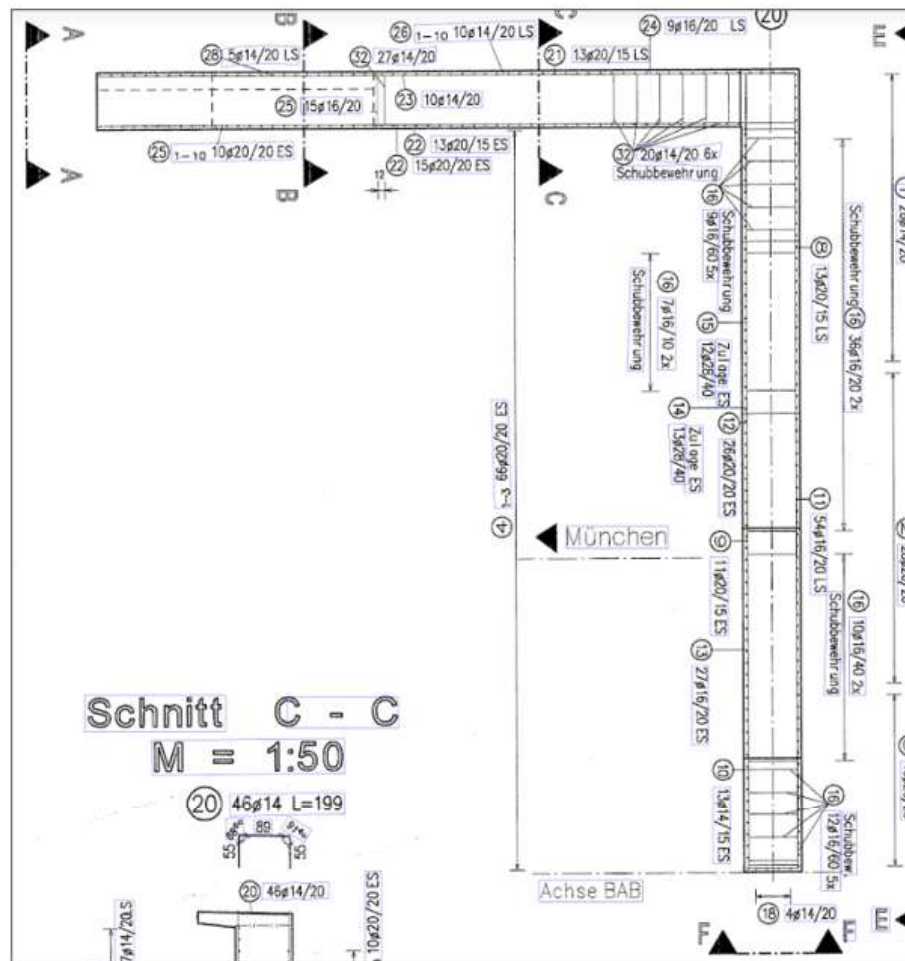


Figure 6.21: Scene text detection performed on cropped image showing the class "abutment\_top\_view"

```

17 output_dir = 'scenedetect/'
18
19 # create a craft instance
20 craft = Craft(output_dir=output_dir, crop_type="poly", cuda=False)
21
22 # apply craft text detection and export detected regions to output
    directory
23 prediction_result = craft.detect_text(image)
24
25 # unload models from ram/gpu
26 craft.unload_craftnet_model()
27 craft.unload_refinenet_model()

```

## Chapter 7

# Conclusion & Future Works

labeled dataset as contribution of technical drawings

*This section concludes the thesis with discussions on findings and provisions for future works. The questions asked in the research objectives section 4.2 will now be discussed under the section 7.1*

### 7.1 Findings and Conclusions

This thesis particularly focused on object detection of bridge elements in technical drawings of bridges using deep learning and parametric modeling techniques. YOLO was trained on the training dataset of 2D technical drawings to detect the classes of interest. Moreover, parametric modeling was performed to augment the dataset. Lastly, scene text detection with the help of CRAFT algorithm was implemented to extract dimensional and textual information from the cropped regions of the technical drawings.

- *What useful information can be extracted from the 2D technical drawings of the bridges?*

In this thesis, object detection in technical drawings of bridges was performed to detect the bridge elements and assign them to their classes of interest. The YOLO object detector was trained on resized technical drawings of bridges and the detected bounding boxes were then scaled-up to map them to the original sized images. These were necessary steps since firstly, if the model is trained on original sized images then several memory-related issues were encountered. Secondly, the scaling-up of bounding boxes was performed because the textual and dimensional information on the resized images was not easily readable as it got blurred during the resize process. Scene text detection using CRAFT was performed to mark bounding boxes on the text and then crop out those bounding box regions.

- *To what degree the process of object detection of bridge elements in technical drawings of bridges can be automated? In other words, what are the contributions of this research?*

In this thesis, object detection and scene text detection was performed to automate or at least automate the object and text detection process in technical drawings of bridges. Since generating DT of the existing bridges is a time-consuming, error prone, and a manual process, in this thesis, efforts were put into automating some of the aspects of detecting objects and text in technical drawings. Several scripts

are mentioned in the earlier chapters to automatically generate train and test text files, to run the detection results on multiple images, to save the output of bounding box coordinates to a text file, to convert the predicted bounding box coordinates to YOLO format, to count the total number of instances of each of the classes, to resize the original sized images, to scale-up the predicted bounding boxes and display them on the original sized images, to crop out the predicted bounded regions on original sized images, and finally to perform scene text detection to extract useful information out of the cropped regions. Also, the trained weights that were achieved in this model training can now be used to detect any other bridge technical drawing. This has contributed in a way that the annotated bridge technical drawing dataset is also available for further reuse in future works.

- *How is the performance of deep learning object detector model which has been selected in this thesis?*

It was observed that YOLO has performed seemingly well in object detection of bridge elements in 2D technical drawings, as can be interpreted from mAP which was 89.15% for the best set of combination of hyperparameters within the scope of this thesis. Also, it can be observed from figures 6.6 to 6.11 that the class probabilities of the classes of interest are also reasonable. However, it was observed that YOLO struggles with detecting smaller objects. Objects that were smaller than 15% of the image size were poorly detected. However, increasing the network resolution to higher numbers such as 832x832 can resolve this issue. On the other hand, the trained YOLO model was able to detect larger objects with a reasonable accuracy. Moreover, it was also able to detect the overlapping classes as can be seen in figure 6.6.

- *How can the performance of the object detection model be improved?*

In the scope of this thesis, the best combination of hyperparameters were arrived by trial and error process. It was observed that as the network resolution increases from 416 to 608, the mAP values improved. However, it took longer to train the model. Moreover, evaluation parameters for the third\_case is summarized in table 7.1 for the only iterations until 8000.

- *Is it possible to augment the existing dataset? If so, what are effects of performing such data augmentation on the overall accuracy?*

The technical drawing dataset used for training was imbalanced as the number of instances for some of the classes were way more than the other classes. The classes with fewer number of instances could not be augmented with a lot of synthetic data as this can also result in lower mAP values. As the classes were already insufficient, a greater amount of synthetic data could not be generated or added to the existing dataset. Figure 5.47 shows the total number of newly added instances for each of the classes. Although adding synthetic data to an existing dataset increases the amount of training data available, it is not always true that adding a huge amount of synthetic data to existing dataset would result in an increase in accuracy values.

Table 7.1: Evaluation parameters for third\_run case

iteration	precision	recall	f1score	avg iou	map
500	0,35	0,14	0,2	22,6	18,99
1000	0,34	0,66	0,45	22,47	46,21
1500	0,8	0,86	0,83	63,85	83,87
2000	0,9	0,85	0,88	72,87	76,92
2500	0,94	0,89	0,92	75,98	84,1
3000	0,94	0,89	0,92	76,61	86,96
3500	0,92	0,91	0,91	75,12	83,05
4000	0,93	0,93	0,93	76,03	88,89
4500	0,9	0,88	0,89	74,27	86,93
5000	0,94	0,88	0,91	79,65	87
5500	0,94	0,89	0,92	79,91	84,71
6000	0,94	0,87	0,91	79,81	87,21
6500	0,92	0,87	0,9	77,13	86,67
7000	0,93	0,89	0,91	79,13	88,16
7500	0,97	0,87	0,92	79,67	83,76
8000	0,9	0,87	0,89	75,56	88,42
best	0,95	0,91	0,92	79,13	89,15
last	0,92	0,87	0,9	77,32	88,45

Also, it can be noticed that the dataset still remains unbalanced. This is due to the fact that several classes, such as `abutment_retaining_wall`, `abutment_wing_wall`, `abutment_top_view`, `label`, `foundation_side_view`, `foundation_top_view`, and `abutment_retaining_wall_cross_section` are more frequently present in any of the technical drawings of a bridge since there are 2 abutments, 4 wing walls, 2 retaining walls, 2 foundations, and so on. Moreover, these classes also have a higher number of instances in the dataset because many of the technical drawings for each of the bridges also included reinforcement details drawn within each of these classes. This has automatically increased the number of instances for each of these classes. Figure 7.1 shows reinforcement details included within the shapes of frequently occurring classes which has increased the number of instances. It was observed that training with synthetic augmented data did not improve the accuracy. This could be possible because of the still imbalanced dataset. The accuracy might not have also improved since the dataset even after data augmentation is still smaller; hence, a larger dataset is required to increase the accuracy further. From figure 7.2, it can be observed that the best `mAP` value was 88.14% for "syn\_run" case as compared to the best case (third\_run 89.15%).

## 7.2 Future works

In this research, object detection was performed to detect bridge elements in 2D technical drawings. This has also resulted in trained weights for as many as 20000 iterations for the

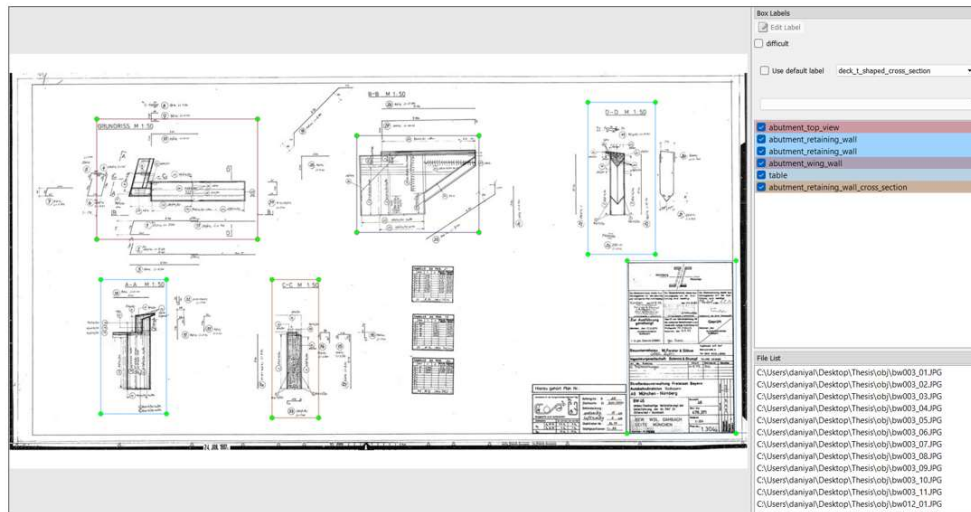


Figure 7.1: A technical drawing in Labellmg tool showing reinforcement details

```

Personal Cloud Storage & | X backup - Google Drive X process_syn.py - Text Editor X obj.data - Text Editor X obj.names - Text Editor X
colab.research.google.com/drive/18YRm2DXngD80ALU61tun14Xq6g3yblk#scrollTo=imc0NP19HtUq
Facebook Gmail YouTube WhatsApp TUM Online Polymorphism in C... TUM Mail Computational Mec... Barre de traction fix...
yolov4_bridge_object_detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
darknet
gdrive
sample_data
obj.zip
syndata_resized.zip
(most mAP calculation at 7800 iterations)
Last accuracy: mAP@0.50 = 84.92 % (best = 88.14 %)
7800: 1.89/245, 1.885559 avg loss, 100000 data = 2716871 seconds, 249600 images, 24.333198 hours left
Resizing: initial size: 608 x 608 try to allocate additional workspace size = 53.23 MB
to allocate done!
calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
detections_count = 134, unique_truth_count = 95
class_id = 0, name = deck_t_shaped_cross_section, ap = 100.00% (TP = 2, FP = 0)
class_id = 1, name = deck_base_shaped_cross_section, ap = 0.000% (TP = 0, FP = 1)
class_id = 2, name = deck_plain_cross_section, ap = 83.33% (TP = 2, FP = 1)
class_id = 3, name = deck_top_view, ap = 75.00% (TP = 3, FP = 0)
class_id = 4, name = abutment_wing_wall, ap = 100.00% (TP = 8, FP = 0)
class_id = 5, name = abutment_retaining_wall, ap = 94.63% (TP = 21, FP = 2)
class_id = 6, name = abutment_retaining_wall_cross_section, ap = 100.00% (TP = 5, FP = 0)
class_id = 7, name = abutment_top_view, ap = 100.00% (TP = 7, FP = 1)
class_id = 8, name = bridge_top_view, ap = 100.00% (TP = 3, FP = 0)
class_id = 9, name = bridge_side_view, ap = 100.00% (TP = 5, FP = 1)
class_id = 10, name = foundation_top_view, ap = 68.21% (TP = 5, FP = 0)
class_id = 11, name = foundation_side_view, ap = 81.82% (TP = 8, FP = 0)
class_id = 12, name = frame, ap = 100.00% (TP = 1, FP = 0)
class_id = 13, name = table, ap = 93.33% (TP = 14, FP = 1)

```

Figure 7.2: A console output showing the best mAP value and iteration number

best set of combinations of hyperparameters. These weight could now be re utilized in future works to perform detections on any other set of technical drawings. Furthermore, a complete set of annotated technical drawing dataset is now available for further reuse in any future works.

In this thesis, scene text detection was also performed on the cropped out regions of bounded by the predicted bounding boxes on the original sized images. The results of this process are bounding boxes surrounding the textual information on the technical drawings as well as cropped out textual information. These results can be re-utilize in future works in a way that now OCR can be carried out on to make the information machine-readable. This would then assist in connecting the dimensional information (numbers) to the edges of the bridge elements. This creates another step forward in generation of DTs of existing bridges.

However, it should also be noticed that even more data is needed to increase the performance of the model. It can be concluded that the results of thesis, such as shape information from object detection and textual information from scene text detection would seemingly assist or lay foundations for future researchers in generating DTs of the existing bridges.



# Bibliography

- AASHTO. (2008). *Bridging the gap: Restoring and rebuilding the nation's bridges*. American Association of State Highways and Transportation Officials.
- AASHTO. (2018). The manual for bridge evaluation, american association of state highway and transportation officials.
- DIN. (1999). *Inspection and testing of engineering structures in connection with roads, din 1076*.
- FHWA. (2018). Bridge preservation guide: Maintaining a resilient infrastructure to preserve mobility.
- ALEXEYAB. (2021). Yolo v4, v3 and v2 for windows and linux.
- ASCE. (2021). Report card for america's infrastructure.
- BAEK, Y., LEE, B., HAN, D., YUN, S., & LEE, H. (2019). Character region awareness for text detection. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9365–9374.
- BORRMANN, A., FORSTER, C., LIEBICH, T., KÖNIG, M., & TULKE, J. (2020). Germany's governmental bim initiative—the bim4infra2020 project implementing the bim roadmap. *International Conference on Computing in Civil and Building Engineering*, 452–465.
- BOTH, P. v. (2012). Potentials and barriers for implementing bim in the german aec market: Results of a current market analysis.
- BRYDE, D., BROQUETAS, M., & VOLM, J. M. (2013). The project benefits of building information modelling (bim). *International journal of project management*, 31(7), 971–980.
- CEBR. (2014). The future economic and environmental costs of gridlock in 2030 - centre for economic and business research.
- CHAN, B., GUAN, H., JO, J., & BLUMENSTEIN, M. (2015). Towards uav-based bridge inspection systems: A review and an application perspective. *Structural Monitoring and Maintenance*, 2(3), 283–300.
- CIMINO, C., NEGRI, E., & FUMAGALLI, L. (2019). Review of digital twin applications in manufacturing. *Computers in Industry*, 113, 103130.
- DOSCH, P., TOMBRE, K., AH-SOON, C., & MASINI, G. (2000). A complete system for the analysis of architectural drawings. *International Journal on Document Analysis and Recognition*, 3(2), 102–116.
- DWIVEDI, R. (2020). My experiment with unet – building an image segmentation model.
- FHWA. (1995). *Recording and coding guide for the structure inventory and appraisal of the nation's bridges. fhwa-pd-96-001*. US Department of Transportation, Federal Highway Administration, Office of Engineering Bridge Division.
- FHWA. (2022a). National bridge inventory (NBI) - based on the coding guide.
- FHWA. (2022b). *Specifications for the national bridge inventory (NBI)*. Federal Highway Administration.

- FUGAS, K. (2019). Everything you should know about basics of bim technology.
- GHOSH, S., DAS, N., DAS, I., & MAULIK, U. (2019). Understanding deep learning techniques for image segmentation. *ACM Computing Surveys (CSUR)*, 52(4), 1–35.
- GIL, N., & BECKMAN, S. (2009). Introduction: Infrastructure meets business: Building new bridges, mending old ones. *California Management Review*, 51(2), 6–29.
- GIMENEZ, L., HIPPOLYTE, J.-L., ROBERT, S., SUARD, F., & ZREIK, K. (2015). Reconstruction of 3d building information models from 2d scanned plans. *Journal of Building Engineering*, 2, 24–35.
- GIRSHICK, R. (2015). Fast r-cnn. *Proceedings of the IEEE international conference on computer vision*, 1440–1448.
- GUBUR, K. (2020). How to optimize images with python for seo and ux.
- HAARDT, P. (2008). The german approach to bridge management: From reactive to predictive management procedures.
- HE, K., ZHANG, X., REN, S., & SUN, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.
- HEARN, G. (2007). *Bridge inspection practices* (Vol. 375). Transportation Research Board.
- HEARN, G., PURVIS, R. L., THOMPSON, P., BUSHMAN, W. H., MCGHEE, K. K., & MCKEEL, W. (2000). Bridge maintenance and management: A look to the future. *TRB 81st Annual Meeting: A3C06: Structures Maintenance and Management*, 1–7.
- HOU, S., WU, G., & LI, H. (2018). An integrated model-based bridge management system. *Maintenance, safety, risk, management and life-cycle performance of bridges*, 198–204.
- HUANG, T. (1996). Computer vision: Evolution and promise.
- HURT, M. A., & SCHROCK, S. D. (2016). *Highway bridge maintenance planning and scheduling*. Butterworth-Heinemann.
- ISAILOVIĆ, D., STOJANOVIC, V., TRAPP, M., RICHTER, R., HAJDIN, R., & DÖLLNER, J. (2020). Bridge damage: Detection, ifc-based semantic enrichment and visualization. *Automation in Construction*, 112, 103088.
- JANIESCH, C., ZSCHECH, P., & HEINRICH, K. (2021). Machine learning and deep learning. *Electronic Markets*, 31(3), 685–695.
- JONES, S. A., & LAQUIDARA-CARR, D. (2017). The business value of bim for infrastructure 2017.
- KANG, S.-O., LEE, E.-B., & BAEK, H.-K. (2019). A digitization and conversion tool for imaged drawings to intelligent piping and instrumentation diagrams (pid). *Energies*, 12, 2593. <https://doi.org/10.3390/en12132593>
- KARSOLIYA, S. (2012). Approximating number of hidden layer neurons in multiple hidden layer bpnn architecture. *International Journal of Engineering Trends and Technology*, 3(6), 714–717.
- KHATAMI, D., & SHAFEI, B. (2021). Impact of climate conditions on deteriorating reinforced concrete bridges in the us midwest region. *Journal of Performance of Constructed Facilities*, 35(1), 04020129.

- KIM, Y. J., & YOON, D. K. (2010). Identifying critical sources of bridge deterioration in cold regions through the constructed bridges in north dakota. *Journal of Bridge Engineering*, 15(5), 542–552.
- KOCH, C., PAAL, S. G., RASHIDI, A., ZHU, Z., KÖNIG, M., & BRILAKIS, I. (2014). Achievements and challenges in machine vision-based inspection of large concrete structures. *Advances in Structural Engineering*, 17(3), 303–318.
- KRITZINGER, W., KARNER, M., TRAAR, G., HENJES, J., & SIHN, W. (2018). Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11), 1016–1022.
- LECUN, Y., BENGIO, Y., & HINTON, G. (2015). Deep learning. *nature*, 521(7553), 436–444.
- LECUN, Y., BOSER, B., DENKER, J., HENDERSON, D., HOWARD, R., HUBBARD, W., & JACKEL, L. (1989). Handwritten digit recognition with a back-propagation network. In D. TOURETZKY (Ed.), *Advances in neural information processing systems*. Morgan-Kaufmann. <https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf>
- LECUN, Y., BOTTOU, L., BENGIO, Y., & HAFFNER, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.
- LEHMANN, F. (2020). Non-destructive testing and monitoring as elements of building inspection. *Otto Graf J*, 19, 119–130.
- LI, F. F., JOHNSON, J., & YEUNG, S. (2017). Lecture 11: Detection and segmentation.
- LI, F. F., JOHNSON, J., & YEUNG, S. (2021). Convolutional neural networks for visual recognition.
- LIN, T.-Y., MAIRE, M., BELONGIE, S., HAYS, J., PERONA, P., RAMANAN, D., DOLLÁR, P., & ZITNICK, C. L. (2014). Microsoft coco: Common objects in context. *European conference on computer vision*, 740–755.
- MADANI, A., ARNAOUT, R., MOFRAD, M., & ARNAOUT, R. (2018). Fast and accurate view classification of echocardiograms using deep learning. *NPJ digital medicine*, 1(1), 1–8.
- MANI, S., HADDAD, M. A., CONSTANTINI, D., DOUHARD, W., LI, Q., & POIRIER, L. (2020). Automatic digitization of engineering diagrams using deep learning and graph search. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 176–177.
- MATHWORKS. (n.d.). Convolutional neural network (cnn).
- MCCLURE, S., & DANIELL, K. (2010). Development of user-friendly software application for extracting information from national bridge inventory source files. *Transportation research record*, 2202(1), 137–147.
- MIGILINSKAS, D., POPOV, V., JUOCEVICIUS, V., & USTINOVICHUS, L. (2013). The benefits, obstacles and problems of practical bim implementation. *Procedia Engineering*, 57, 767–774.
- MIHAJLOVIC, I. (2019). Everything you ever wanted to know about computer vision.
- MIRZAEI, Z. (2014). *Overview of existing bridge management systems - report by the iabmas bridge management committee*.
- MITCHELL, T. (2006). *The discipline of machine learning*.

- NEGRI, E., FUMAGALLI, L., & MACCHI, M. (2017). A review of the roles of digital twin in cps-based production systems. *Procedia Manufacturing*, 11, 939–948.
- NURMINEN, J. K., RAINIO, K., NUMMINEN, J.-P., SYRJÄNEN, T., PAGANUS, N., & HONKOILA, K. (2019). Object detection in design diagrams with machine learning. *International Conference on Computer Recognition Systems*, 27–36.
- NWANKPA, C., IJOMAH, W., GACHAGAN, A., & MARSHALL, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.
- OMAR, T., & NEHDI, M. L. (2018). Condition assessment of reinforced concrete bridges: Current practice and research challenges. *Infrastructures*, 3(3), 36.
- PARROTT, A., & WARSHAW, L. (2017). Industry 4.0 and the digital twin: Manufacturing meets its march.
- PASZKE, A., GROSS, S., CHINTALA, S., CHANAN, G., YANG, E., DEVITO, Z., LIN, Z., DESMAISON, A., ANTIGA, L., & LERER, A. (2017). Automatic differentiation in pytorch.
- PREGNOLATO, M. (2019). Bridge safety is not for granted—a novel approach to bridge management. *Engineering Structures*, 196, 109193.
- PULKIT, S. (2019). Computer vision tutorial: A step-by-step introduction to image segmentation techniques (part 1).
- RASHIDI, M., SAMALI, B., & SHARAFI, P. (2016). A new model for bridge management: Part a: Condition assessment and priority ranking of bridges. *Australian Journal of Civil Engineering*, 14(1), 35–45.
- REDMON, J., DIVVALA, S., GIRSHICK, R., & FARHADI, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.
- REITSEMA, A. D., LUKOVIĆ, M., GRÜNEWALD, S., & HORDIJK, D. A. (2020). Future infrastructural replacement through the smart bridge concept. *Materials*, 13(2). <https://doi.org/10.3390/ma13020405>
- REN, S., HE, K., GIRSHICK, R., & SUN, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.
- REYNOLDS, A. (2019). Convolutional neural networks (cnn).
- RI-EBW-PRÜF. (2003). *Recording and assessment of damages, preservation and maintenance, guideline ri-ebw-prüf*.
- ROBERTS, L. G. (1963). *Machine perception of three-dimensional solids* (Doctoral dissertation). Massachusetts Institute of Technology.
- ROCHA, G., MATEUS, L., FERNÁNDEZ, J., & FERREIRA, V. (2020). A scan-to-bim methodology applied to heritage buildings. *Heritage*, 3(1), 47–67.
- RUSSELL, S. J. (2010). *Artificial intelligence a modern approach*. Pearson Education, Inc.
- SABACK DE FREITAS BELLO, V., POPESCU, C., BLANKSVÄRD, T., & TÄLJSTEN, B. (2021a). Bridge management systems: Overview and framework for smart management. *IABSE Congress Ghent 2021, Structural Engineering for Future Societal Needs, Ghent, Belgium, 22-24 September, 2021*, 629–637.

- SABACK DE FREITAS BELLO, V., POPESCU, C., BLANKSVÄRD, T., & TÄLJSTEN, B. (2021b). Framework for facility management of bridge structures using digital twins. *IABSE Congress Ghent 2021, Structural Engineering for Future Societal Needs, Ghent, Belgium, 22-24 September, 2021*, 629–637.
- SACKS, R., EASTMAN, C., LEE, G., & TEICHOLZ, P. (2018). *Bim handbook: A guide to building information modeling for owners, designers, engineers, contractors, and facility managers*. <https://doi.org/10.1002/9781119287568>
- SHIM, C. S., KANG, H., DANG, N. S., & LEE, D. (2017). Development of bim-based bridge maintenance system for cable-stayed bridges. *Smart Struct. Syst*, 20(6), 697–708.
- SHUKLA, L. (2019). Designing your neural networks.
- SIB-BAUWERKE. (2013). Asb-ing (anweisung straßeninformationsbank für ingenieurbauten, teilsystem bauwerksdaten) und schlüsseltabellen, stand: 2013/10.
- SPENCER, B. F., HOSKERE, V., & NARAZAKI, Y. (2019). Advances in computer vision-based civil infrastructure inspection and monitoring. *Engineering*, 5(2), 199–222. <https://doi.org/https://doi.org/10.1016/j.eng.2018.11.030>
- SRIKANTH, I., & AROCKIASAMY, M. (2020). Deterioration models for prediction of remaining useful life of timber and concrete bridges: A review. *Journal of traffic and transportation engineering (English edition)*, 7(2), 152–173.
- SUCCAR, B. (2009). Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in construction*, 18(3), 357–375.
- TZUTALIN. (2015). Labelimg.
- VAN DAELE, D., DECLEYRE, N., DUBOIS, H., & MEERT, W. (2019). An automated engineering assistant: Learning parsers for technical drawings. *arXiv preprint arXiv:1909.08552*.
- YIN, X., WONKA, P., & RAZDAN, A. (2008). Generating 3d building models from architectural drawings: A survey. *IEEE computer graphics and applications*, 29(1), 20–30.
- ZHOU, X., YAO, C., WEN, H., WANG, Y., ZHOU, S., HE, W., & LIANG, J. (2017). East: An efficient and accurate scene text detector. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2642–2651. <https://doi.org/10.1109/CVPR.2017.283>
- ZHU, Z., GERMAN, S., & BRILAKIS, I. (2010). Detection of large-scale concrete columns for automated bridge inspection. *Automation in construction*, 19(8), 1047–1055.

# Declaration

I hereby affirm that I have independently written the thesis submitted by me and have not used any sources or aids other than those indicated.

Munich, 06.June.2022

---

Location, Date, Signature