Department of Computer Science

MSc in Artificial Intelligence

Academic Year 2023-2024

Transfer learning Integrated YOLOv10 for Metal Surface Defect Detection

Meng Heng Chong-2374392
Supervised by Dr. Weibo Liu

A report submitted in partial fulfilment of the requirement for the degree of Master of Science

Brunel University
Department of Computer Science
Uxbridge, Middlesex UB8 3PH
United Kingdom
Tel: +44 (0) 1895 203397
Fax: +44 (0) 1895 251686

ABSTRACT

In manufacturing industries, the production of metal strips is prone to defects like scratches, crazing, inclusion etc., which affects the quality and mechanical performance of industrial products. To address the quality issue, metal surface defect detection aims to capture and localise the defects accurately. However, current defect detection methods struggle in either detecting tiny defects or real time defect detection. Recently, the latest YOLO has set new benchmarks in object detection tasks, with high average precision and fast latency speed. However, the application of YOLOv10 in metal surface defect detection remains unexplored. In this study, a YOLOv10 based model is proposed for metal surface defect detection. This is achieved by employing transfer learning to adapt YOLOv10, pre-trained on the COCO dataset, to detect defects on metal surfaces. The pre-trained YOLOv10 is retrained on benchmark metal surface defects dataset, 'NEU-DET' and 'GC10-DET'. The experiment on NEU-DET dataset shows that YOLOv10 based model achieved 77.7% mAP at 32.89FPS, outperforming SSD, faster R-CNN, and improved faster R-CNN. Another experiment on GC10-DET dataset shows that YOLOv10 based model achieved 66% mAP at 64.52FPS, outperforming SSD and faster R-CNN. However, the YOLOv10 based model have low recall compared to other methods. Future research should focus on modification of YOLOv10 framework to achieve better performance. Additionally, this study provides a comprehensive review of object detection methods and defect detection methodologies.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my dissertation supervisor, Dr. Weibo Liu for guiding me to complete my project. His expertise, patience and suggestions are vital in leading the direction and outcome of this dissertation. Besides that, I would also like to express my appreciation to my friends who did not hesitate to have a discussion when I faced problems. Thank you all for your generous support and encouragement.

I certify that the work presented in the dissertation is my own unless referenced

Signature.....

Date: 10/9/2024

TOTAL NUMBER OF WORDS: 13140

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION 1
1.1 Research aim and objectives 2
1.2 Research approach 2
1.3 Dissertation outline 2
CHAPTER 2: LITERATURE REVIEW 3
2.1 Overview of defect detection research 3
2.2 Typical object detection framework 3
2.3 One-stage object detection method 5
2.3.1 you-only-look-once (YOLO) series 5
2.3.2 Single shot multi box detector 20
2.3.3 Deconvolutional single shot detector 22
2.4 Two-stage object detection method 23
2.4.1 R-CNN 23
2.4.2 Fast R-CNN 24
2.4.3 Faster R-CNN 25
2.5 Related works 26
2.6 Identification of research gap 28
CHAPTER 3: METHODOLOGY 29
3.1 Overall Approach 29
3.2 Methods 29
3.2.1 Mapping objective to methods 30
3.3 Instruments 33
3.4 Dataset 34
3.4.1 NEU-DET 34
3.4.2 GC10-DET 35
3.5 Ethics 36
3.6 Limitation 36
CHAPTER 4: RESULTS 37
4.1 Evaluation metrics 37
4.2 Results on NEU-DET dataset 38
4.2.1 Results 38
4.2.2 Visualisation of results 39
4.2.3 Comparison between actual labels and model predictions 43
4.3 Results on GC10-DET dataset 46
4.3.1 Results 46

4.3.2 Visualisation of results 47
4.3.3 Comparison between actual labels and model predictions 51
CHAPTER 5: DISCUSSION 54
5.1 Analysis of Poor Detection 54
5.2 Method comparison on NEU-DET dataset 56
5.3 Method comparison on GC10-DET dataset 57
5.4 Discussion of Findings 58
CHAPTER 6: CONCLUSION 59
6.1 Summary 59
6.2 Research contributions 59
6.3 Future research and development 59
6.4 Personal reflections 59
REFERENCE 60
APPENDIX A: ETHICAL APPROVAL 67
APPENDIX B: CODE 68

LIST OF FIGURES

Figure 1: Object detector anatomy (Hussain, 2023) 3
Figure 2: YOLO detection process (Redmon et al., 2016) 5
Figure 3: YOLOv1 architecture (Redmon et al., 2016) 7
Figure 4: Calculation of loss function (Terven et al., 2023) 8
Figure 5: Multiple anchor boxes for each grid cell (Terven et al., 2023) 9
Figure 6: Structure of YOLOv3 (Terven et al., 2023) 10
Figure 7: Architecture detail of YOLOv3 (Terven et al., 2023) 10
Figure 8: YOLOv4 architecture (Terven et al., 2023) 11
Figure 9: YOLOv5 architecture (Terven et al., 2023) 13
Figure 10: YOLOv6 architecture 14
Figure 11: YOLOv7 architecture (Terven et al., 2023) 15
Figure 12: YOLOv8 architecture (Terven et al., 2023) 16
Figure 13: PGI and related methods (Chien et al., 2024) 17
Figure 14: Architecture of GELAN (Chien et al., 2024) 18
Figure 15: Dual label assignments (Wang et al., 2024) 19
Figure 16: Structure of SSD network (Liu et al., 2016) 20
Figure 17: Lower-resolution feature maps (right) detect larger-scale objects. 20
Figure 18: Original SSD (top) vs DSSD (bottom) (Fu et al., 2017) 22
Figure 19: Proposed residual block for prediction module (Fu et al., 2017) 22
Figure 20: Detection process of R-CNN (Girshick et al., 2014) 23
Figure 21: Selective search process (Uijlings et al., 2013) 23
Figure 22: Fast R-CNN architecture (Girshick, 2015) 24
Figure 23: Faster R-CNN (Ren et al., 2016) 25
Figure 24: Process of generating region proposals using RPN (Ren et al., 2016) 25
Figure 25: The general architecture of YOLOv10 (Wang et al., 2024) 30
Figure 26: Architecture of CSPNet (Wang et al., 2020) 30
Figure 27: Summary of pre-trained YOLOv10-X model 32

Figure 28: Defect instances on NEU-DET (Lv et al., 2020) 34
Figure 29: Defect instances on GC10-DET (Lv et al., 2020) 35
Figure 30: Recall and mean average precision over the training process on NEU-DET 38
Figure 31: Confusion matrix for NEU-DET 39
Figure 32: Normalised confusion matrix for NEU-DET 40
Figure 33: Precision confidence curve for NEU-DET 40
Figure 34: Recall-confidence curve for NEU-DET 41
Figure 35: F1-confidence curve for NEU-DET 42
Figure 36: Precision-recall curve for NEU-DET 42
Figure 37: Defect Labels on testing batch 0 43
Figure 38: Model predictions on testing batch 0 43
Figure 39: Defect labels on testing batch 1 44
Figure 40: Model's predictions on testing batch 1 44
Figure 41: Defect labels on testing batch 2 45
Figure 42: Model's predictions on testing batch 2 45
Figure 43: Recall and mean average precision over the training process on GC10-DET 46
Figure 44: Confusion matrix for GC10-DET 47
Figure 45: Normalised confusion matrix for GC10-DET 48
Figure 46: Precision-confidence curve for GC10-DET 48
Figure 47: Recall-confidence curve for GC10-DET 49
Figure 48: F1-confidence curve for GC10-DET 49
Figure 49: Precision recall curve for GC10-DET 50
Figure 50: Defect labels on testing batch 0 51
Figure 51: Model prediction on testing batch 0 51
Figure 52: Defect labels on testing batch 1 52
Figure 53: Model prediction on testing batch 1 52
Figure 54: Defect labels on testing batch 2 53
Figure 55: Model prediction on testing batch 2 53
Figure 56: Ground truth labels for inclusion on the GC10-DET dataset 54
Figure 57: Ground truth labels (left) vs model prediction (right) on the NEU-DET dataset 54
Figure 58: Overlapping ground truth labels for crazing defect 55

LIST OF TABLES

Table 1: Timeline of YOLO evolution 6
Table 2: YOLOv2 architecture (Terven et al., 2023) 9
Table 3: Difference between exploratory research design and conclusive research design (CS5704, Lecture 3) 29
Table 4: Performance of YOLO model on COCO dataset (Ultralytics, 2024) 31
Table 5: No. of images in each set before vs after data augmentation 32
Table 6: Training hyperparameter 33
Table 7: No. of images for each defect class in the NEU-DET dataset 34
Table 8: No. of images in each defect class in the GC10-DET dataset 35
Table 9: Defect detection performance on NEU-DET 38
Table 10: Performance on GC10-DET defect detection 46
Table 11: mAP and FPS comparison between different methods on NEU-DET 56
Table 12: Recall the comparison between different methods on NEU-DET 56
Table 13: mAP and FPS comparison between different methods on GC10-DET 57
Table 14: Recall comparison between different methods on GC10-DET 57

CHAPTER 1: INTRODUCTION

Defect inspection is a crucial function in the manufacturing industry to ensure that the manufacturing process and product quality are under control. On the other hand, steel strip is a fundamental element across various industries including automotive, aerospace, architecture etc. (Qi et al., 2020). The manufacturing of steel strips is a complex process and is susceptible to defects due to instrument, human, mechanical and environmental factors (Qi et al., 2020). Typical surface defects like inclusion, scratches and crazing degrade the mechanical performance and quality of steel strips. Before the invention of computer vision, defect inspection is completed by human inspectors. However, manual inspection by human inspectors is expensive, inefficient, and inaccurate especially when tiny surface defects occur (Zheng et al., 2021). Besides that, manual inspection cannot satisfy increasing manufacturing demands and quality standards of industrial manufacturing (Zheng et al. 2021).

Recent advancements in deep learning have led to the widespread application of computer vision based on convolutional neural networks (CNN) in the manufacturing industry, particularly in automating the defect detection process (Qi et al., 2020). Object detection frameworks like single-shot multi-box detectors, you-only-look-once (YOLO), faster R-CNN, and RetinaNet have been proposed for defect detection problems. However, detecting tiny defects is still challenging because object detection methods don't fully mine latent information like stronger semantic features from the feature map and more accurate location (Zeng et al., 2022). Methods like RetinaNet and faster R-CNN demonstrate competitive defect detection accuracy, but insufficient processing speed cannot satisfy real-time defect detection.

YOLO, a state-of-the-art real-time object detection system has undergone numerous iterations from 2015 until now. Previous studies implemented YOLOv3 on defect detection, but it demonstrates poor precision and recall. In May 2024, the latest YOLOv10 is released by researchers at Tsinghua University (Wang et al., 2024). The latest YOLOv10 shows exceptional performance and outperforms state-of-the-art object detectors on COCO object detection (Wang et al., 2024). However, empirical studies of YOLOv10 in defect detection are currently limited, as the YOLOv10 was just released at the onset of this research.

Motivated by the discussions above, this study aims to explore a YOLOv10-based model for metal surface defect detection through transfer learning. The YOLOv10-based defect detection model is trained and evaluated on two benchmark metal surface detection datasets, 'NEU-DET' and 'GC10-DET'. A research question is also defined as 'Explore the capability of the latest YOLOv10 in metal surface defect detection through transfer learning'. The contribution of this study includes a comprehensive review of object detection methods, a critical analysis of current defect detection methods, and a novel YOLOv10-based metal surface defect detection model.

1.1 Research aim and objectives

This study aims to explore an advanced deep learning model for detecting surface defects on image data accurately. To achieve this aim, several objectives are defined as follows:

- Objective 1: Discover deep learning approach for surface defect detection.
- Objective 2: Prepare image data of metal surface defects.
- Objective 3: Pre-process the data.
- Objective 4: Build and train the defect detection model.
- Objective 5: Test and verify the effectiveness of the proposed model.

1.2 Research approach

To propose a metal surface defect detection model, image data of metal surface defects is analysed and modelled through computational techniques. Therefore, this study is categorized as quantitative research, involving systematic empirical investigation through statistical, mathematical, numerical data or computational methods (Given, 2008).

- To achieve objective 1, a comprehensive review of the object detection framework and current surface defect detection methodologies is conducted.
- To achieve objective 2, benchmark datasets for metal surface defect detection, 'NEU-DET' and 'GC10-DET' are downloaded from kaggle.com.
- To achieve objective 3, data augmentation including rotating, flipping, brightness contrast and Gaussian blur are implemented. The dataset is divided into 70% training set, 15% validation set, and 15% testing set through random sampling.
- To achieve objective 4, the pre-trained YOLOv10 from COCO object detection is retrained on the defect detection dataset.
- To achieve objective 5, the trained YOLOv10-based model is evaluated on the testing set and compared with the state-of-the-art defect detection model.

1.3 Dissertation outline

The remaining sections are organized as follows. Chapter 2 provides a comprehensive review of object detection framework and defect detection methodologies. Chapter 3 detailed the methodology and step-by-step procedure for building, training, and testing the YOLO v10-based defect detection model. In Chapter 4, the performance of the YOLO v10-based defect detection model is presented through quantitative tables and visualisation of detection outcomes. Chapter 5 offers a critical analysis of the detection results and compares the proposed model with state-of-the-art defect detection models. Finally, Chapter 6, concludes the dissertation by summarising the research process, critically assessing the limitation of the proposed model, and identify future research directions to enhance detection performance.

CHAPTER 2: LITERATURE REVIEW

2.1 Overview of defect detection research

Defect detection research can be categorized into traditional computer vision detection and deep learning detection (Li et al., 2022). The difference between traditional computer vision detection and deep learning detection lies on the feature recognition process from images. Traditional computer vision relies on feature engineering and manually crafted algorithms to identify and extract relevant features like edges, geometric shapes, and textures from images (Li et al., 2022). However, irregular defect distribution in the real world like complex background textures, noise and different light conditions makes manually crafted algorithms difficult to extract features, which leads to poor detection accuracy (Li et al., 2022). Conversely, deep learning detection uses convolutional neural networks (CNN) to automatically recognize and extract feature information from images. Compared with traditional computer vision detection, deep learning detection achieves higher detection accuracy, better generalisation, and faster speed (Li et al., 2022). As CNN technology advances, deep learning detection becomes the predominant method in defect detection (Bhatt et al., 2021).

2.2 Typical object detection framework

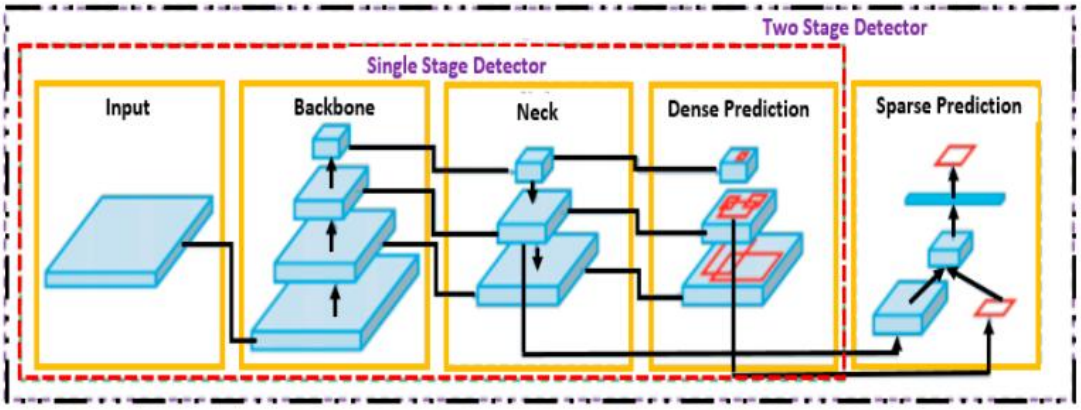

Figure 1: Object detector anatomy (Hussain, 2023)

Defect detection is a specific category of object detection tasks, and it is powered by an object detection framework. Figure 1 shows that a typical object detection framework consists of an input layer, a backbone, a neck, and a detection head. The input layer augments the input images by applying spatial transformation cropping, flipping, scaling, and rotating (Zeng et al., 2022).

The backbone network extracts hierarchical features from the input image, with low-level features extracted in initial layers and high-level features extracted in deeper layers. (Terven et al., 2022). Convolutional neural networks (CNN) are extensively employed as backbone networks owing to their superior capability in feature extraction, feature recognition, and parameter reduction (Fu et al., 2022). CNNs attained remarkable performance in image classification, object detection and

image segmentation by autonomously learning significant hierarchical representations (Li et al., 2021). A standard CNN architecture has at least one convolutional layer for feature extraction, one pooling layer for subsampling, and one fully connected later for output (Fu et al., 2022). There are different CNN architectures which are used as backbone networks such as the visual geometry group (VGG) network (Simonyan and Zisserman, 2014), the AlexNet (Krizhevsky et al., 2012), the residual network (He et al., 2016), the Inception Net (Szegedy et al., 2015), and the dark network (Redmon and Farhadi, 2018).

The neck network like the path aggregation network (Liu et al., 2018), feature pyramid network (Lin et al., 2017), and bidirectional feature pyramid network (Zhu et al., 2018) carry out feature fusion to merge the extracted features of each layer and study the latent features (Zeng et al., 2022).

The detection head detects object inside each regional proposal, localizes the bounding box to the detected object and generate class label for classification. The postprocessing module in the detection head includes the non-maximum suppression (NMS) method (Neubeck and Van Gool, 2006), the soft NMS method (Bodla et al., 2017) and the weighed NMS method (Ning et al., 2017).

Object detectors can be further classified into one-stage detectors and two-stage detectors. One-stage detector directly predicting bounding boxes, computing the category probability and localize coordinate of object in a single forward pass (Zeng et al., 2022). Some representative one-stage detectors including single-shot multi-box detector (Liu et al., 2016), you-only-look-once model (YOLO) (Redmon et al., 2016), the corner network (Law and Deng, 2018) and the RetinaNet (Cheng and Yu, 2020). Unlike one-stage detectors, two-stage detectors generate region proposals with approximate location information in the first stage, classify candidate regions into specific groups and refine the bounding box coordinates in the second stage. Region-based convolutional neural networks (R-CNN) family such as the R-CNN (Girshick et al., 2014), the fast R-CNN (Girshick, 2015), the faster R-CNN (Ren et al., 2016), and the mask R-CNN (He et al., 2017) are representative two-stage detectors. Experiments show that two-stage detector is generally more accurate than one-stage detector but require more processing time and inefficient sometimes, especially in real-time application (Liu et al., 2016).

2.3 One-stage object detection method

2.3.1 you-only-look-once (YOLO) series

In 2016, Redmon et al. proposed a novel one-stage object detector called you-only-look-once (YOLO). Unlike two-stage object detector, YOLO reformulates object detection as a single pass regression problem, directly converting image pixels to bounding box coordinates and class probabilities (Redmon et al., 2016). The YOLO uses a single neural network and features from the entire image to predict each bounding box and the corresponding confidence score.

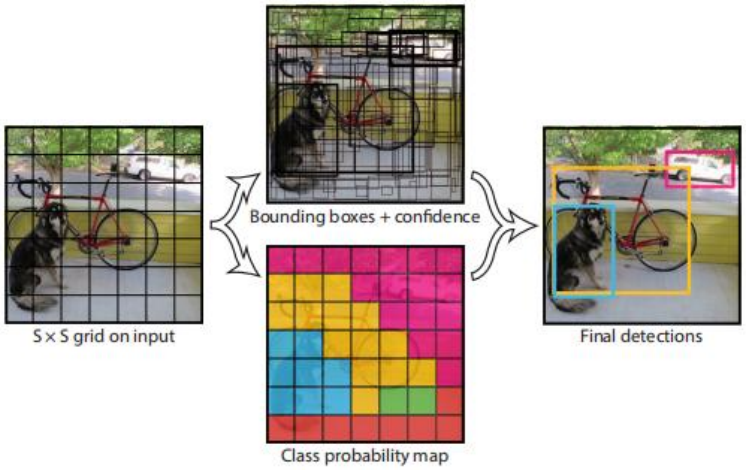

Figure 2: YOLO detection process (Redmon et al., 2016)

The YOLO detection process starts by dividing the input image into S x S grid cells. If the centre of the target object falls into a grid cell, that grid cell is responsible for detecting the target object (Redmon et al., 2016). Each grid cell predicts a fixed number of bounding boxes, corresponding confidence score, and class probabilities (Redmon et al., 2016). The prediction process can be further broken down into bounding box prediction and class prediction.

The bounding box prediction is represented by (x, y, w, h, c), where (x, y) coordinates represent the centre of the bounding box, w and h represents the width and height of the bounding box, and c represents the confidence score for the bounding box (Redmon et al., 2016). The confidence reflects how confident the object is within the bounding box, and it can be expressed as $\Pr(Object) * IOU_{pred}^{truth}$ (Redmon et al., 2016). The confidence score includes the probability of the object being present in the bounding box, denoted by Pr (object) and the precision of the bounding box. The $\Pr(Object)$ ranges from 0 to 1, with 0 indicating no target object, and 1 indicating the target object is in the bounding box. The precision of the bounding box is denoted by intersection over union (IOU), which is the ratio of intersection area and union area between predicted boxes and ground truth boxes (Zhang et al., 2021).

$$\Pr(Class_i|Object|) * confidence = \Pr(Class_i|Object|) * \Pr(Object) * IOU_{pred}^{truth}$$
$$= \Pr(Class_i) * IOU_{pred}^{truth} \quad (1)$$

After bounding box prediction, each grid cell also predicts C conditional class probabilities, $\Pr(Class_i|Object|)$ to classify the object in the bounding box (Redmon et al., 2016). As shown in Formula 1, conditional class probabilities are then multiplied by the individual box confidence predictions to produce class-specific confidence scores for each box. The class-specific confidence score indicates the likelihood of the class in the box and how well the predicted box matches the object (Redmon et al., 2016).

Given the S x S grid and each grid cell predicts B bounding boxes, confidence for those bounding boxes and C class probabilities, final predictions are denoted by S x S x (B*5 +C) tensor (Redmon et al., 2016). B is multiplied by 5 because each box includes *(x, y, w, h, c)*.

Publish Year	YOLO version	Author
2016	YOLOv1	Redmon et al.
2017	YOLOv2/YOLO9000	Redmon et al.
2018	YOLOv3	Redmon et al.
2020	YOLOv4	Bochkovskiy et al.
2020	YOLOv5	Jocher
2022	YOLOv6	Li et al.
2023	YOLOv7	Wang et al.
2023	YOLOv8	Jocher et al.
2024	YOLOv9	Chien et al.
2024	YOLOv10	Wang et al.

Table 1: Timeline of YOLO evolution

The YOLO series is revolving at a rapid pace from 2016 until 2024, and the latest YOLO has come to YOLO-v10. All versions adhere YOLO detection process, only modifications are made to optimize the detection performance. Each YOLO version is briefly explained as follows.

YOLOv1

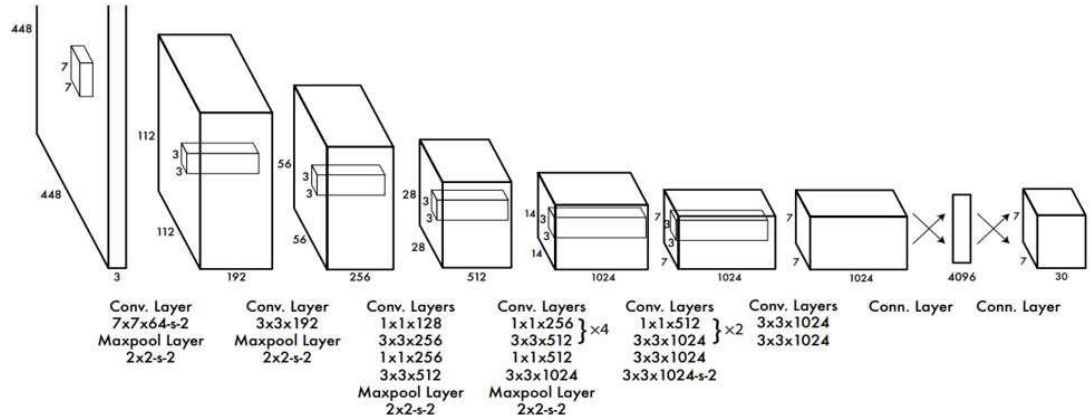

Figure 3: YOLOv1 architecture (Redmon et al., 2016)

The original YOLOv1 network is a custom convolutional neural network inspired by GoogLeNet architecture (Szegedy et al., 2015) for image classification. Unlike the original GoogLeNet, Redmon et al. (2016) replaced the inception module with 1x1 reduction layers followed by 3x3 convolutional layers. The YOLOv1 consists of 24 convolutional layers for feature extractions, followed by 2 fully connected layers for class probabilities and bounding box coordinates predictions (Redmon et al., 2016). Redmon et al. (2016) also proposed a fast version of YOLOv1 (Fast YOLO), in which convolutional layers are deducted to 9 layers with fewer filters to achieve faster object detection.

Before the training process, Redmon et al. (2016) pre-train the first 20 convolutional layers, followed by an average pooling layer and a fully connected layer on the 1000-class competition dataset (Russakovsky et al., 2015). Four convolutional layers and two fully connected layers afterwards are initialised with random weights (Redmon et al., 2016). The resolution of input images is halved to 224x224 pixels when pre-training convolutional layers, and the resolution is doubled to 448x448 pixels for detection (Redmon et al., 2016) (Zhang et al., 2021). The activation function of the final layer is a linear activation function while all other layers are leaky rectified linear activation functions (leaky-ReLU) (Redmon et al., 2016). The final layer uses non-maximum suppression algorithms (Neubeck and Van Gool, 2006) to predict the class probability and bounding box coordinates (Redmon et al., 2016).

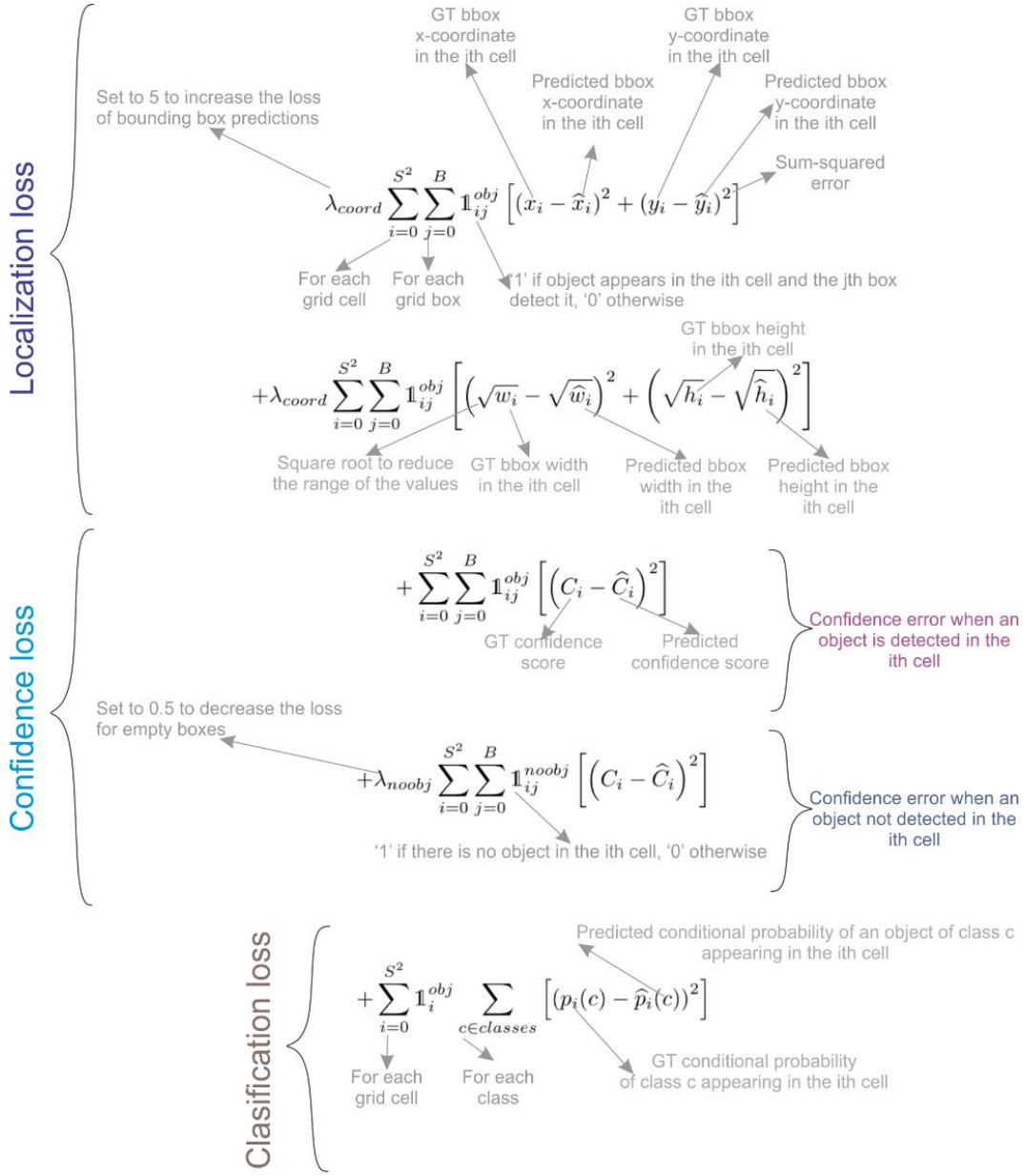

Figure 4: Calculation of loss function (Terven et al., 2023)

The training loss function is a mean square error, which includes localization loss, confidence loss, and classification loss. l_i^{obj} represent whether the target is in the cell i , l_{ij}^{obj} indicates jth bounding box predictor is responsible for predicting bounding boxes in cell i , C_i indicates the confidence score, w_i and h_i are normalized width and height. The loss function penalizes classification error when the object is located within that grid cell (Redmon et al., 2016). Additionally, the loss function penalizes bounding boxes' coordinate error if that predictor is accountable for the ground truth box (Redmon et al., 2016).

The YOLOv1 demonstrates exceptional speed and can be trained end-to-end owing to predictions by a single neural network. However, the YOLOv1 struggles in detecting small objects that are clustered and objects of varying scales.

YOLOv2/YOLO9000

YOLOv2, also called YOLO9000 is introduced by Redmon et al. in 2017. Building upon YOLOv1, YOLOv2 made several enhancements which achieved a substantial improvement in detection speed and accuracy.

Num	Type	Filters	Size/Stride	Output
1	Conv/BN	32	3 × 3/1	416 × 416 × 32
2	Max Pool		2 × 2/2	208 × 208 × 32
3	Conv/BN	64	3 × 3/1	208 × 208 × 64
4	Max Pool		2 × 2/2	104 × 104 × 64
5	Conv/BN	128	3 × 3/1	104 × 104 × 128
6	Conv/BN	64	1 × 1/1	104 × 104 × 64
7	Conv/BN	128	3 × 3/1	104 × 104 × 128
8	Max Pool		2 × 2/2	52 × 52 × 128
9	Conv/BN	256	3 × 3/1	52 × 52 × 256
10	Conv/BN	128	1 × 1/1	52 × 52 × 128
11	Conv/BN	256	3 × 3/1	52 × 52 × 256
12	Max Pool		2 × 2/2	52 × 52 × 256
13	Conv/BN	512	3 × 3/1	26 × 26 × 512
14	Conv/BN	256	1 × 1/1	26 × 26 × 256
15	Conv/BN	512	3 × 3/1	26 × 26 × 512
16	Conv/BN	256	1 × 1/1	26 × 26 × 256
17	Conv/BN	512	3 × 3/1	26 × 26 × 512
18	Max Pool		2 × 2/2	13 × 13 × 512
19	Conv/BN	1024	3 × 3/1	13 × 13 × 1024
20	Conv/BN	512	1 × 1/1	13 × 13 × 512
21	Conv/BN	1024	3 × 3/1	13 × 13 × 1024
22	Conv/BN	512	1 × 1/1	13 × 13 × 512
23	Conv/BN	1024	3 × 3/1	13 × 13 × 1024
24	Conv/BN	1024	3 × 3/1	13 × 13 × 1024
25	Conv/BN	1024	3 × 3/1	13 × 13 × 1024
26	Reorg layer 17			13 × 13 × 2048
27	Concat 25 and 26			13 × 13 × 3072
28	Conv/BN	1024	3 × 3/1	13 × 13 × 1024
29	Conv	125	1 × 1/1	13 × 13 × 125

Table 2: YOLOv2 architecture (Terven et al., 2023)

Darknet-19 is adopted as a backbone network (from layer 1 to layer 23 in Table 2) for YOLOv2. Darknet-19 is a fully convolutional architecture, which has 19 convolutional layers followed by five max-pooling layers. Batch normalisation is applied after each convolutional layer to improve the convergence rate and reduce overfitting. To extract finer-grained features, YOLOv2 uses a passthrough layer that reorganize a 26 x 26 x 512 feature map by stacking adjacent features into different channels (Redmon et al., 2017) (Terven et al., 2023). This generates 13 × 13 × 2048 feature maps concatenated in the channel dimension with the lower resolution 13 × 13 × 1024 maps to obtain 13 × 13 × 3072 feature maps (Redmon et al., 2017) (Terven et al., 2023). Absence of fully connected layers in the YOLOv2 architecture allows multi-scale training, in which inputs can be of different sizes from 320 x320 up to 608 x 608. This further improves the YOLOv2's robustness to different-size inputs.

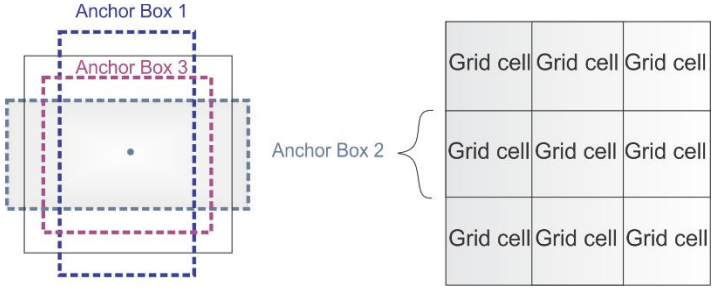

Figure 5: Multiple anchor boxes for each grid cell (Terven et al., 2023)

Unlike YOLOv1 which predicts the coordinates of bounding boxes directly, YOLOv2 uses a set of anchor boxes with pre-defined shapes and sizes to predict bounding boxes (Redmon et al., 2017). YOLOv2 first define multiple anchor boxes per grid cell, then predicts the anchor boxes' coordinates and class (Terven et al., 2023). During training, YOLOv2 adopts k-means clustering on training bounding boxes to search suitable anchor boxes and the number of anchor boxes per grid cell automatically.

YOLOv3

One year after the publishment of YOLOv2, YOLOv3 is introduced by Redmon et al. (2018) with several enhancements in the backbone network, bounding boxes prediction and class prediction.

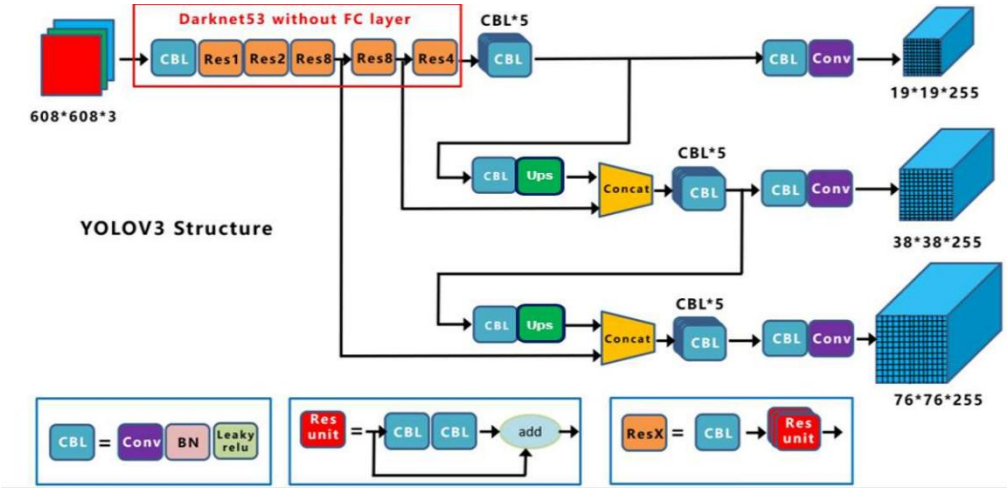

Figure 6: Structure of YOLOv3 (Terven et al., 2023)

Layer	Filters	size	Repeat		Output size
Image					416 × 416
Conv	32	3 × 3/1	1		416 × 416
Conv	64	3 × 3/2	1		208 × 208
Conv	32	1 × 1/1	Conv		208 × 208
Conv	64	3 × 3/1	Conv	×1	208 × 208
Residual			Residual		208 × 208
Conv	128	3 × 3/2	1		104 × 104
Conv	64	1 × 1/1	Conv		104 × 104
Conv	128	3 × 3/1	Conv	×2	104 × 104
Residual			Residual		104 × 104
Conv	256	3 × 3/2	1		52 × 52
Conv	128	1 × 1/1	Conv		52 × 52
Conv	256	3 × 3/1	Conv	×8	52 × 52
Residual			Residual		52 × 52
Conv	512	3 × 3/2	1		26 × 26
Conv	256	1 × 1/1	Conv		26 × 26
Conv	512	3 × 3/1	Conv	×8	26 × 26
Residual			Residual		26 × 26
Conv	1024	3 × 3/2	1		13 × 13
Conv	512	1 × 1/1	Conv		13 × 13
Conv	1024	3 × 3/1	Conv	×4	13 × 13
Residual			Residual		13 × 13

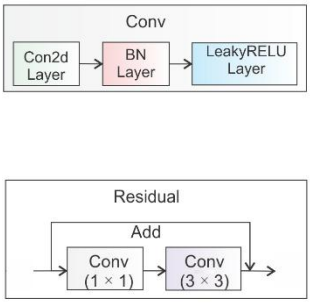

Figure 7: Architecture detail of YOLOv3 (Terven et al., 2023)

Darknet-53 is introduced as a backbone network of YOLOv3, which is more powerful than the Darknet-19 used in YOLOv2 (Redmon et al., 2018). Figure 7 shows that Darknet-53 consists of 53 convolutional layers with residual connections. Unlike

Darknet-19, all max-pooling layers are replaced by stride convolutions and residual connections to preserve fine-grained features (Terven et al., 2023).

For bounding boxes prediction, YOLOv3 simplifies the calculation of objectness score/confidence score by using only a logistic regression function (Redmon et al., 2018). Besides that, YOLOv3 introduce multi-scale prediction by predicting bounding boxes at three different scales with three boxes per scale (Redmon et al., 2018). To address multilabel classification problems, YOLOv3 uses binary cross-entropy loss function to train independent logistic classifiers instead of SoftMax used in YOLOv2 (Redmon et al., 2018).

YOLOv4

Two years later the YOLOv3, Bochkovskiy (2020) released YOLOv4. Although the author is not Redmon, the YOLO v4 retains the same YOLO philosophy and the improvement is satisfactory that the computer vision community acknowledges this work as YOLOv4 (Terven et al., 2023). Starting from this version, the network architecture is described in the backbone, the neck, and the detection head (Terven et al., 2023). Main changes of the YOLOv4 include an enhanced network architecture, advanced training approaches and hyperparameter optimization techniques.

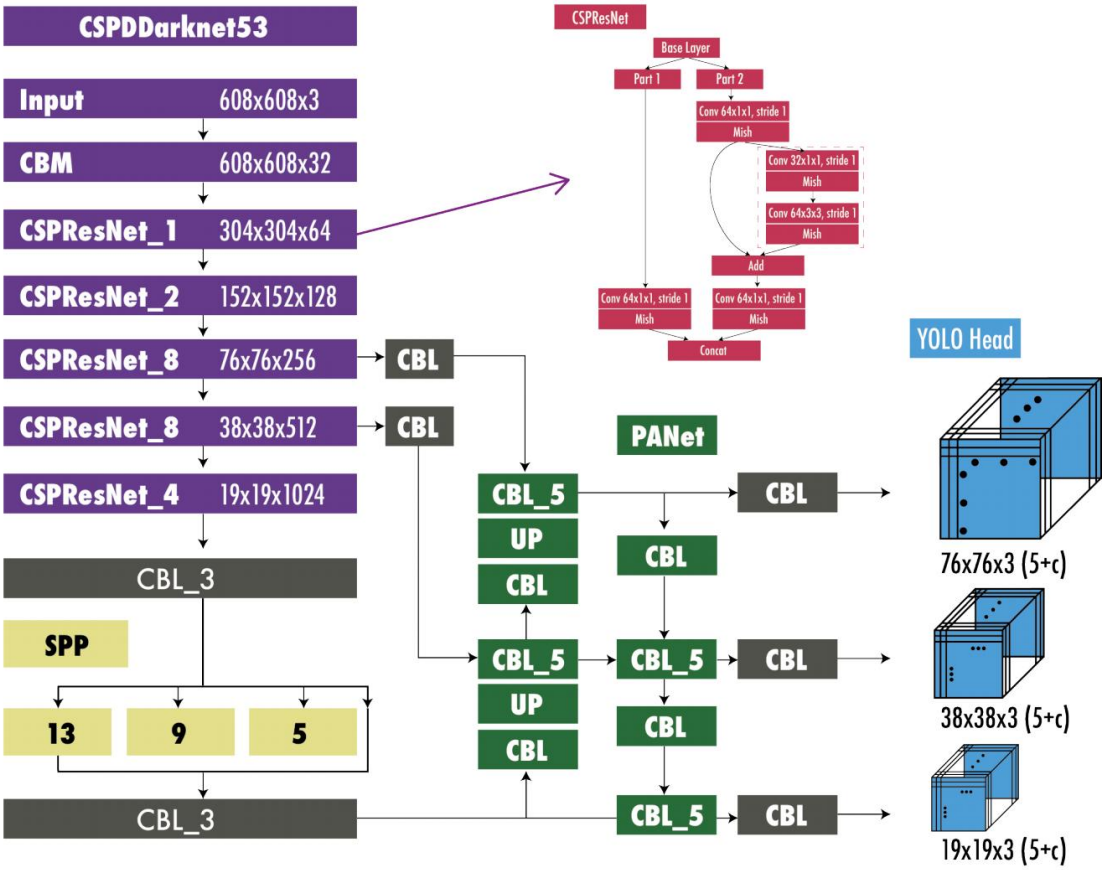

Figure 8: YOLOv4 architecture (Terven et al., 2023)

YOLOv4 uses CSPDarknet-53 as its backbone network. CSPDarknet-53 is an enhanced version of Darknet with bag-of-specials, which introduced cross-stage

partial connection (Wang et al., 2020) for computational reduction and Mish activation functions (Misra, 2019) for smoother gradient. For the neck, multi-scale prediction is retained as in YOLOv3 but with a modified path aggregation network (PANet) (Liu et al.,2018) and modified spatial attention module (Woo et al., 2018). Besides that, modified spatial pyramid pooling (SPP) (He et al., 2015) is used to increase the receptive field while maintaining the inference speed (Bochkovskiy, 2020). Figure 8 shows the YOLOv4 architecture, where the CMB module consists of convolution layers with batch normalisation and Mish activation functions, the CBL module consists of convolutional layers with batch normalisation and Leaky- ReLU activation functions, UP indicates upsampling, SPP indicates spatial pyramid pooling, PANet represents path aggregation network (Terven et al., 2023).

YOLOv4 integrates Bag of Freebies (BoF) techniques, which introduces mosaic data enhancement for data augmentation, DropBlock (Ghiasi et al., 2018) for regularization, Complete intersection over union (CIoU) loss function (Zheng et al., 2020) for the detector, and cross-mini-batch normalisation (CmBN) for statistic collection from the entire batch (Bochkovskiy, 2020) (Terven et al., 2023). To improve model's robustness to perturbation, self-adversarial training is adopted to avoid model from being fooled by adversarial perturbations (Bochkovskiy, 2020).

For hyperparameter optimization, genetic algorithms are adopted on the first 10% of periods to find the best training hyperparameter. Besides that, the author uses a cosine annealing scheduler (Loshchilov and Hutter, 2016) to control the learning rate. The learning rate gradually decreases, then quickly reduces through the training process, and ends with a small reduction (Bochkovskiy, 2020) (Terven et al., 2023).

YOLOv5

A few months after the release of YOLOv4, YOLOv5 was released by Jocher (2020), the founder and CEO at Ultralytics. YOLOv5 is quite like YOLOv4 but it is developed in Pytorch rather than Darknet. YOLOv4 exhibits superior accuracy compared to YOLOv5; nevertheless, YOLOv5 offers greater flexibility and a more compact network size. The main differences between YOLOv4 and YOLOv5 include anchor initialisation techniques and the network architecture. YOLOv5 incorporates the AutoAnchor algorithm to automatically check and adjust ill-fitted anchor boxes and training settings (Jocher,2020) (Terven et al., 2023).

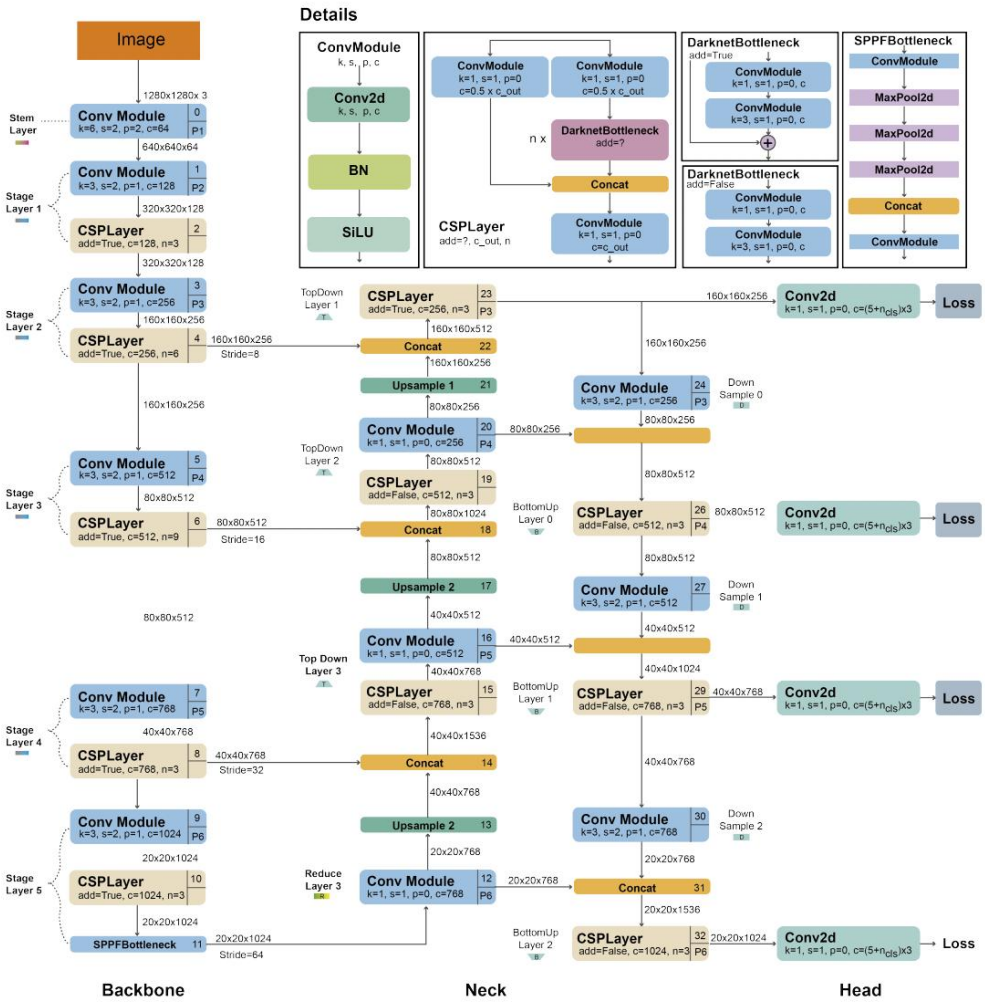

Figure 9: YOLOv5 architecture (Terven et al., 2023)

The backbone architecture of YOLOv5 is a modified CSPDarknet-53. The architecture starts with a stem layer, where stride convolution with an extensive window size is applied to reduce the memory and computational complexity prior to convolution layers (Terven et al., 2023). Each convolutional module consists of a convolutional layer succeeded by batch normalisation and SiLU activation function (Terven et al., 2023). The spatial pyramid pooling fast (SPPF) layer, positioned on the end of the backbone, along with the subsequent convolutional layers, effectively processes features at varying scales, pooling them into a fixed-size feature map and

thereby speed up the computation (Terven et al., 2023). The neck employs SPPF and a modified CSP-PAN, while the head resembles YOLOv3.

YOLOv6

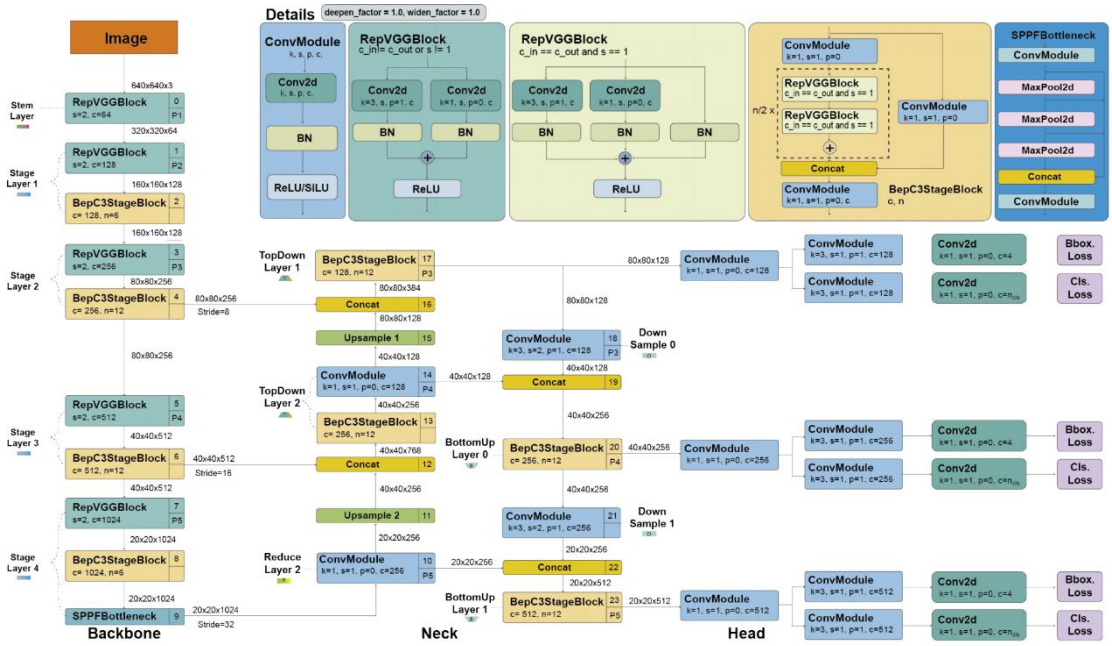

Figure 10: YOLOv6 architecture

Two years after the YOLOv5, YOLOv6 was released by Li et al. (2022). YOLOv6 made changes in architecture and loss function, adopted quantization technique, self-distillation strategy and task alignment learning.

The YOLOv6 adopts RepVGG (Ding et al., 2021) as a backbone network. For the neck, the path aggregation network is enhanced with RepBlocks (Ding et al., 2021) or CSPStackRep (Wang et al., 2020) to improve feature aggregation and processing. In the detection head, an efficient decoupled detection head with a hybrid-channel strategy is developed to separate classification and regression tasks for better performance (Terven et al., 2023).

YOLOv6 employs VariFocal loss (Zhang et al., 2021) for calculating classification loss, skew intersection over union (SIoU) (Gevorgyan, 2022) or generalized intersection over union (GIoU) (Rezatofighi et al., 2019) for calculating regression loss. To achieve faster and more accurate detection, Li et al. (2022) introduced enhanced quantization techniques using RepOptimizer (Ding et al., 2022) and channel-wise distillation (Shu et al., 2021) (Terven et al., 2023). Besides that, Li et al. (2022) use task alignment learning introduced in task-oriented object detection (Feng et al., 2021) to achieve better label assignment.

Li et al. (2022) present eight model variants from YOLOv6-N to YOLOv6-L6. The largest model achieved 57.2% average precision (AP) at 29 frames per second (FPS) on the MS COCO dataset test-dev 2017.

YOLOv7 (need to check)

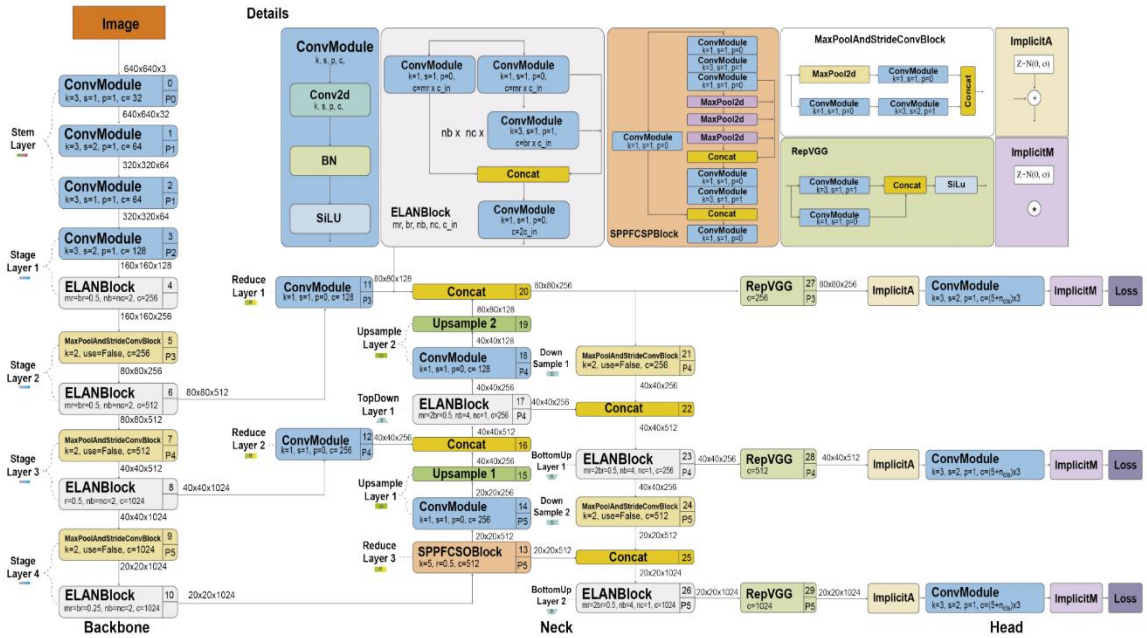

Figure 11: YOLOv7 architecture (Terven et al., 2023)

In 2023, YOLOv7 is released by the author of YOLOv4 (Wang et al., 2023). Compared to YOLOv6, Wang et al. (2023) made changes to the architecture and adopted bag-of-freebies methods.

The backbone network of YOLOv7 is an extended efficient layer aggregation network (E-ELAN) (Wang et al., 2022). The original ELAN (efficient later aggregation network) optimizes the learning and convergence of a deep model by controlling the gradient path (Terven et al., 2023). The E-ELAN enhances the network's learning by integrating features from different groups through shuffling and merging cardinality, without disrupting the original gradient path (Terven et al., 2023). The E-ELAN is specifically designed to work with models that have an unlimited number of stacked computational blocks, results in significant improvement over the original ELAN. Besides that, YOLOv7 proposed a new scaling technique which scales the depth and width of the block to maintain optimal structure of concatenation-based architectures (Terven et al., 2023).

Bag-of-freebies methods include planned re-parameterized convolution (RepConvN), batch normalisation with conv-bn-activation, implicit knowledge inspired in YOLOR (Wang et al., 2021), exponential moving average as the final inference model, sparse label assignment for the auxiliary head, and fine label assignment for the lead head (Terven et al., 2023). The RepConvN is used in RepVGG (see figure 11), in which the identity connection is removed to prevent disruption of residual connections in ResNet and the concatenation operations in DenseNet (Terven et al., 2023).

The experiment shows that YOLOv7 outperforms YOLOv4 with 75% parameter reduction, 36% computational reduction, and an improved average prevision (AP) B 1.5% (Wang et al., 2023).

YOLOv8

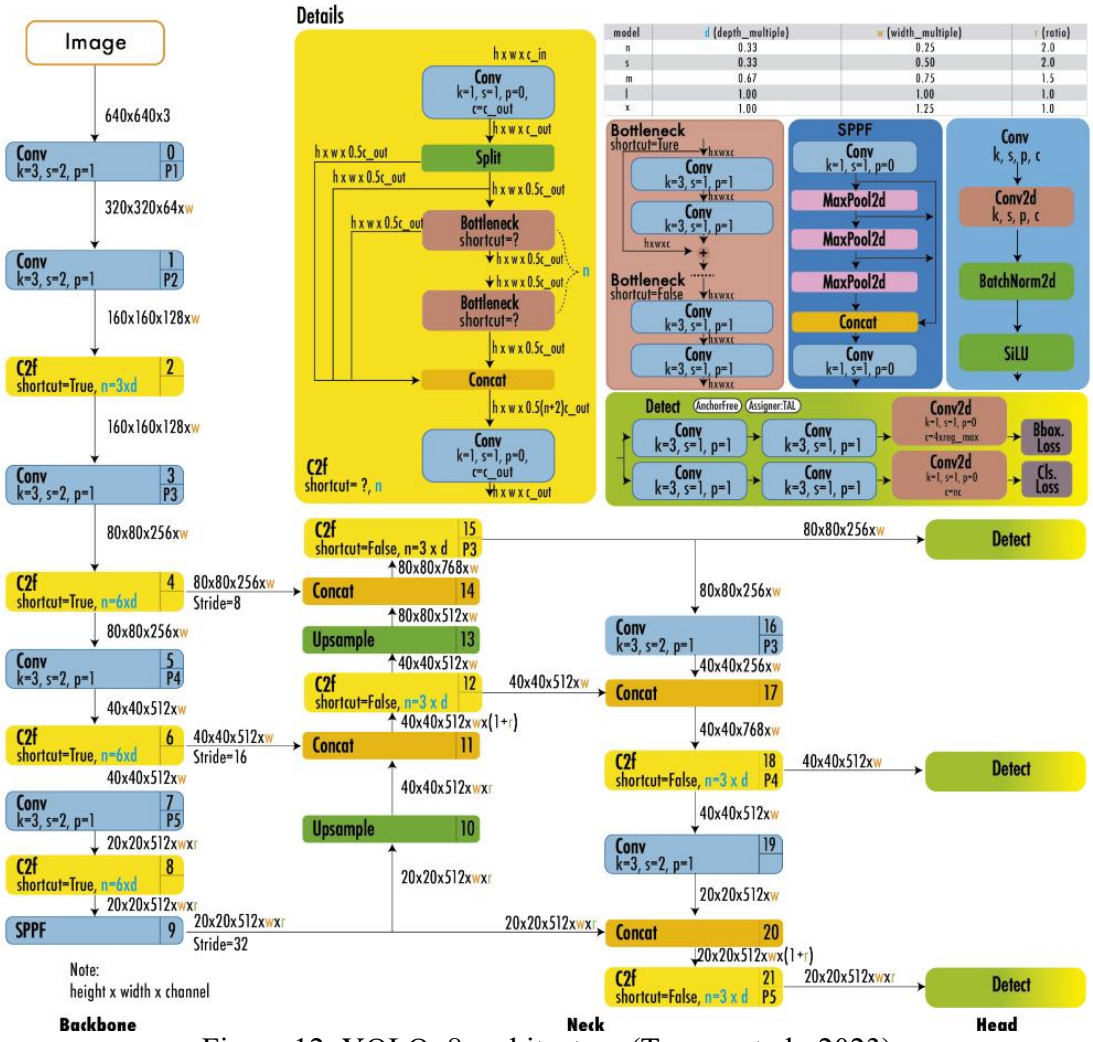

Figure 12: YOLOv8 architecture (Terven et al., 2023)

YOLOv8 was released by Jocher et al. (2023) from Ultralytics, who is also the author of YOLOv5. As there is no official YOLOv8 paper by Jocher et al. (2023), an explanation of YOLOv8 is referenced from a review paper by Terven et al. (2023). The YOLOv8 have five scaled version including YOLOv8n, YOLO8s, YOLOv8m, YOLOv8l, and YOLOv8x.

The backbone of YOLOv8 architecture comes back to CSPDarknet-53 used in the YOLOv5, but the CSPLayer is now replaced by the C2f module (Terven et al., 2023). The C2f module consists of cross-stage partial bottleneck with two convolutions which integrates high-level features with contextual information to enhance detection accuracy (Terven et al., 2023).

Besides that, YOLOv8 introduces a decoupled head, where the tasks of predicting objectness, classification, and regression are handled by separate branches in the network (Terven et al., 2023). This allows each branch to focus on its specific task. YOLOv8 also adopts an anchor-free approach to simplify the model and reduces the computational complexity (Terven et al., 2023).

YOLOv8 applies batch normalisation and SiLU activation functions after each convolution. In the output layer, YOLOv8 uses a sigmoid activation function for objectness score prediction and a softmax activation function for class probabilities prediction (Terven et al., 2023). For loss computation, YOLOv8 uses CIoU (Zheng et al., 2020) for bounding box regression, distribution focal loss (DFL) (Li et al., 2020) for bounding box localisation, and binary cross-entropy loss for classification (Terven et al., 2023). Experiments show that those loss functions have improved object detection performance, especially on small object detection (Terven et al., 2023). The YOLOv8x outperforms YOLOv5 (50.7%AP) with 53.9%AP at 280 FPS on the MS COCO dataset test-dev 2017. (Terven et al., 2023).

YOLOv9

In 2024, Chien et al. released YOLOv9, which focuses on mitigating information loss over deep neural processing to enhance overall efficiency and accuracy. YOLOv9 introduces a programmable gradient information (PGI) framework and the generalized efficient layer aggregation network (GELAN) architecture.

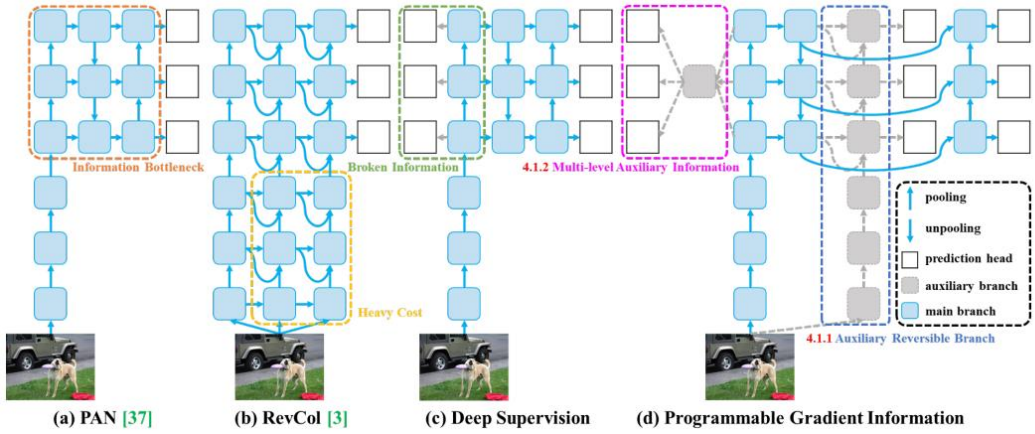

Figure 13: PGI and related methods (Chien et al., 2024)

PGI is used to maintain data integrity through the processing layers. PGI consists of a main branch, an auxiliary reversible branch and multi-level auxiliary information. The main branch is the primary architecture used for inference. The auxiliary reversible branch generates gradient information to support the main branch during backward transmission (Chien et al., 2024). When the main branch deep features lose important information due to an information bottleneck, they can obtain valid gradient information from the auxiliary reversible branch (Chien et al., 2024). These gradient information drive parameters learning to assist in extracting correct

and important information, so the main branch can obtain effective features for the target task (Chien et al., 2024). Multi-level auxiliary information is a plannable multi-level of semantic information that controls main branch learning (Chien et al., 2024). It works by aggregating the information containing all target objects and send it to the main branch and then update parameter (Chien et al., 2024).

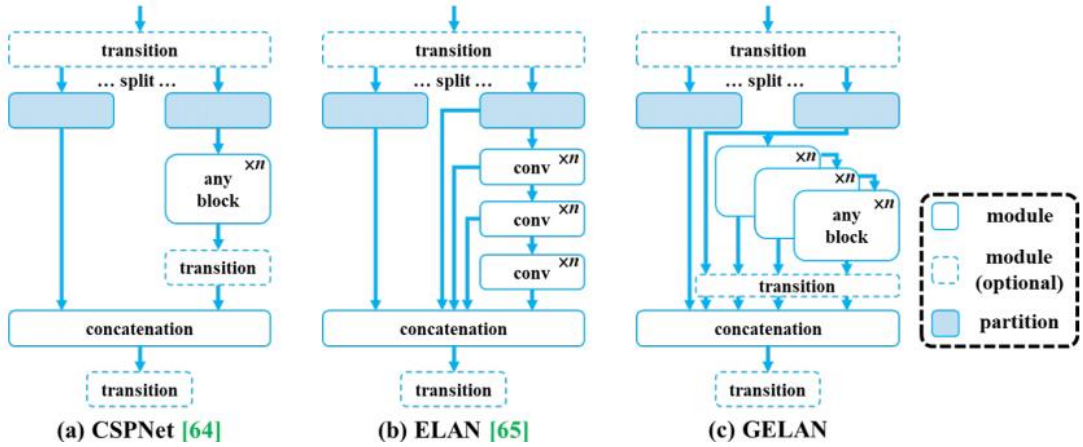

Figure 14: Architecture of GELAN (Chien et al., 2024)

The GELAN architecture integrates CSPNet and ELAN to form a new network architecture which considers lightweight, inference speed and accuracy (Chien et al., 2024). While the original ELAN only used stacking of convolutional layers, the new ELAN architecture can use any computational block (Chien et al., 2024). Compared to YOLOv8, YOLOv9 achieves 49% parameter reduction, 43% computational reduction and 0.6%AP improvement on the MS COCO dataset (Chien et al., 2024).

YOLOv10

Also in 2024, YOLOv10 (Wang et al., 2024) is released by researchers at Tsing Hua University. Previous YOLO versions perform a one-to-many label assignment strategy during training, where one ground truth object subjected to multiple positive bounding boxes (Wang et al., 2024). Non-maximum suppression (NMS) is then used to select the best positive bounding boxes during inference. However, NMS affects the inference speed and the detection performance is highly dependent on NMS hyperparameters (Wang et al., 2024). To address this issue, YOLOv10 introduces an NMS-free training approach using dual label assignments and consistent matching metrics.

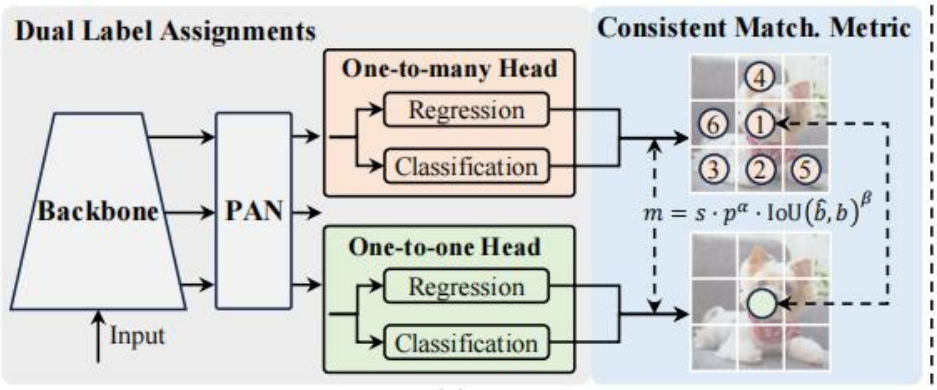

Figure 15: Dual label assignments (Wang et al., 2024)

Dual label assignments integrate the advantages of one-to-one head and one-to-many head, where the one-to-one head can avoid NMS post-processing while the one-to-many head can reduce performance degradation caused by the one-to-one head (Wang et al., 2024). During training, two heads are jointly optimized with the model, where the backbone and neck can benefit from rich supervision by the one-to-many assignment (Wang et al., 2024). During inference, the one-to-many head is discarded, and only the one-to-one head is used for prediction (Wang et al., 2024).

$$m(\alpha, \beta) = s \cdot p^{\alpha} \cdot IoU(b^{\wedge}, b)^{\beta} \qquad (2)$$

To enhance the prediction accuracy during inference, Wang et al. (2024) employs consistent metrics that align the supervision between both heads, where p is the classification score, b^ and b represent the predicted bounding box and instance respectively, and s denotes the spatial prior whether the predicted anchor point is within the instance. α and β are hyperparameters that control the impact between the semantic prediction task and the location regression task.

In addition to post-processing, Wang et al. (2024) also perform efficiency-driven model design. To reduce the computational overhead of the classification head, Wang et al. (2024) adopt a lightweight classification head consisting of two depth-wise separable convolutions (Chollet, 2017) with 3x3 kernel size followed by a 1x1 convolution. Besides that, Wang et al. (2024) decouple spatial reduction and channel modulation to maximise information retention during down sampling while reducing computational cost. Furthermore, Wang et al. (2024) propose a rank-guided block design based on intrinsic stage redundancy to ensure optimal parameter utilization. To maximize the performance under minimal cost, Wang et al. (2024) further explore the large-kernel depth-wise convolution and self-attention (Vaswani et al., 2017) for accuracy-driven design. Large kernel depth-wise convolution enlarges the receptive field while self-attention modules improve global representation learning with minimal overhead (Wang et al., 2024).

2.3.2 Single shot multi box detector

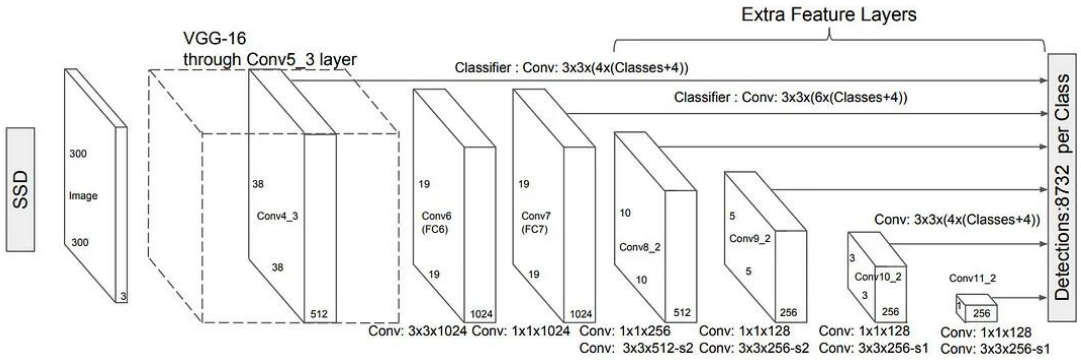

Figure 16: Structure of SSD network (Liu et al., 2016)

In 2016, Liu et al. (2016) proposed a single-shot multi-box detector for real-time object detection. SSD is a feed-forward convolutional neural network that produces fixed-size bounding boxes and scores to detect target instances in those boxes, followed by non-maximum suppression to produce the most relevant detections (Liu et al., 2016).

As shown in Figure 16, the SSD uses the VGG-16 network (Simonyan and Zisserman, 2014) as a backbone network to extract features from images and generate a set of feature maps for object detection (Liu et al., 2016). VGG-16 network consists of a series of convolutional and pooling layers that progressively reduce the spatial dimension of the input image while increasing the depth of feature maps (Simonyan and Zisserman, 2014).

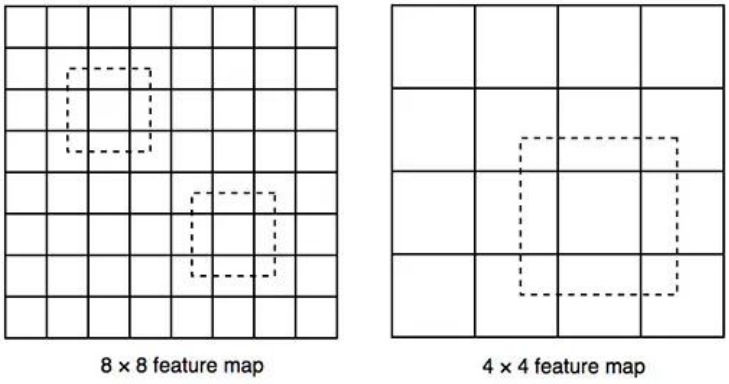

Figure 17: Lower-resolution feature maps (right) detect larger-scale objects. (Hui, 2018)

After the backbone network, an additional six convolutional feature layers with progressively decreasing size are designed to capture multi-scale features, thereby allowing multi-scale prediction from large objects to small objects (Liu et al., 2016). Each additional convolutional feature layer produces detection predictions in different scales using a set of 3x3 convolutional filters. A convolutional filter, also known as the kernel, is the basic element that produces either a categorical score or a

bounding box offsets value from the default box coordinates (Liu et al., 2016). With an input image, six convolutional feature layers produce 8732 predictions (Hui, 2018).

When the kernel is applied to each feature map location, it produces an output value. These predictions are based on "default boxes" (anchor boxes) that the SSD associates with each feature map cell (Liu et al., 2016). To clarify, default boxes are pre-defined bounding boxes of varied sizes and aspect ratios which are used as initial guesses for objects' locations in an image. Default boxes are applied across multiple feature maps with different resolutions to detect varying-size objects. The convolutional filter predicts offset to default boxes to fit the target object more accurately.

$$L(x, c, l, g) = \frac{1}{N}\left(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)\right) \qquad (3)$$

During training, each ground truth box is matched with the highest IoU default box. To encourage the model for predicting high scores for multiple overlapping default boxes instead of the only best matchbox, default boxes with IoU greater than 0.5 are classified as positive matches (Liu et al., 2016). The loss function is a weighted sum of localisation loss (L_{loc}) and confidence loss (L_{conf}), where N is the number of matched default boxes, L_{loc} is the smooth L1 loss between predicted box (l) and the ground truth box (g), L_{conf} is the SoftMax loss over multiple classes confidences (c) and weight term (α) is set to 1 by cross-validation (Liu et al., 2016). After the matching process, most of default boxes are negatives matches and this leads to an imbalance between positive and negative instances. Liu et al. (2016) introduce hard negative mining to select negative default boxes with the highest confidence losses for training and the negatives and positives ratio is maintained at 3:1.

By combining predictions from all feature maps and default boxes, the SSD produces a final set of bounding boxes, each associated with a class score (Liu et al., 2016). After obtaining these predictions, SSD applies NMS to select the best-matching boxes with the highest confidence score (Liu et al., 2016). However, SSD struggles in detecting small targets.

2.3.3 Deconvolutional single shot detector

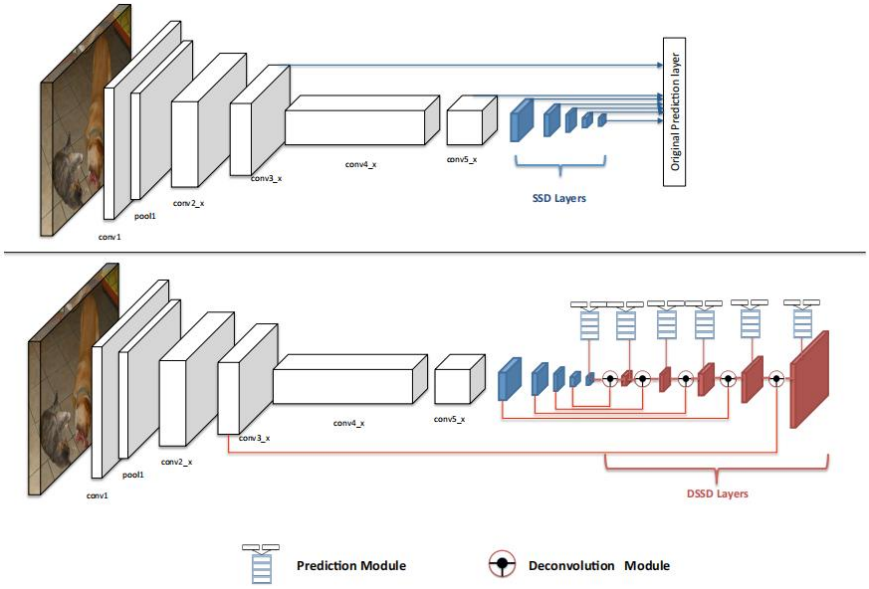

Figure 18: Original SSD (top) vs DSSD (bottom) (Fu et al., 2017)

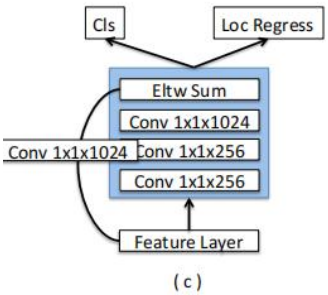

Figure 19: Proposed residual block for prediction module (Fu et al., 2017)

To address poor small object detection in SSD, Fu et al. (2017) introduce deconvolutional single shot detector (DSSD). Firstly, DSSD introduces Residual-101 (He et al., 2016) as its backbone network, which replaces the VGG (Visual Geometry Group) network used in the original SSD (Fu et al., 2017). Besides that, extra deconvolutional layers (red layers in Figure 18) with skip-connection are added on the base of the original SSD to up-sample the feature map, effectively enhancing the resolution of feature map layers before they are used for detection (Fu et al., 2017). To integrate information from earlier feature maps and deconvolutional layers, the deconvolutional module is added after the last feature layer from SSD (last blue layer in Figure 18) and each deconvolutional layer (red layer in figure 18) (Fu et al., 2017). Furthermore, a residual block is added for each prediction layer as ablation studies show that it can improve accuracy (Fu et al., 2017).

With modifications above, DSSD outperforms SSD with 81.5% mAP on VOC2007 test, 80.0% on VOC2012 test, and 33.2%mAP on COCO (Fu et al., 2017). It achieves significant improvement on small object detection, outperforming SSD, faster R-CNN, and region-based fully convolutional networks (R-FCN) on each dataset (Fu et al., 2017).

2.4 Two-stage object detection method

2.4.1 R-CNN

By combining region proposals with convolutional neural networks (CNNs), Girshick et al. (2014) introduced regions with CNN features (R-CNN) for object detection. The R-CNN consists of three modules, the first module generates category-independent region proposals, the second module is a pre-trained AlexNet based CNNs that extracts features from each region, and the third module is a set of class-specific linear SVMs for object classification within each region (Girshick et al., 2014).

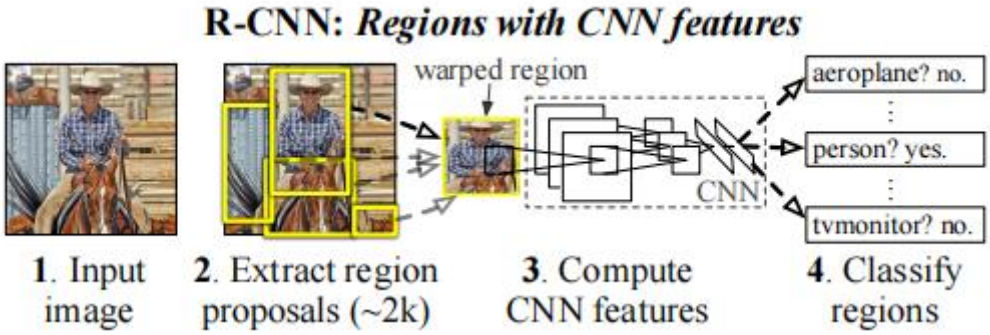

Figure 20: Detection process of R-CNN (Girshick et al., 2014)

Figure 21: Selective search process (Uijlings et al., 2013)

The R-CNN firstly generates around 2000 region proposals that are likely to contain objects for each image through a selective search algorithm (Girshick et al., 2014). Selective search (Uijlings et al., 2013) integrates a bottom-up grouping strategy with saliency cues to create region proposals of varying size and aims at reducing search space in object detection (Zhao et al., 2019). Generated region proposals are resized to 277x277 pixel size and passed through the pre-trained CNNs for feature extractions (Girshick et al., 2014). The CNNs output 4096-dimensional features as the final representation (Girshick et al., 2014). After feature extractions, the R-CNN adopts a category-specific linear supporting vector machine (SVM) to classify each region proposal into different object classes and background (non-object) regions (Girshick et al., 2014) (Zhao et al., 2019). Afterwards, the bounding box

regression is applied to adjust bounding box coordinates to better fit the detected object (Girshick et al., 2014). Finally, a greedy NMS algorithm filters redundant bounding boxes to retain the most confident bounding boxes, producing final bounding boxes for detected objects (Girshick et al., 2014).

While R-CNN shows significant improvement in detection accuracy, the limitation comes obviously when the R-CNN extracts and classifies 2000 region proposals per image, resulting in expensive computational cost and long processing time (around 50s per image) (Girshick et al., 2014).

2.4.2 Fast R-CNN

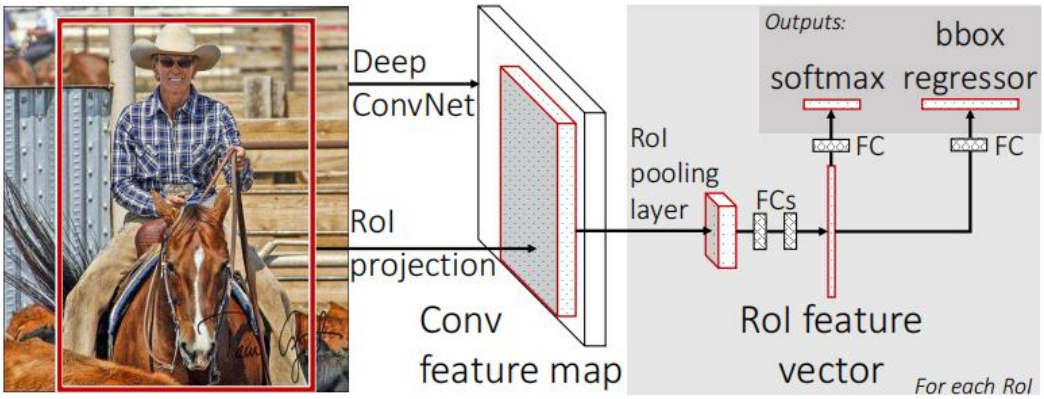

Figure 22: Fast R-CNN architecture (Girshick, 2015)

To address the limitation of R-CNN, Girshick (2015) proposed fast region-based convolutional neural networks (Fast R-CNN) for object detection. Unlike R-CNN which processes each region proposal independently with CNNs, fast R-CNN processes the entire image with CNNs in a single forward pass to generate feature maps.

Fast R-CNN initially generates approximately 2000 region proposals using selective search algorithm. Subsequently, the entire image is processed with CNNs to generate a convolutional feature map (Girshick, 2015). Each generated region proposal is applied and mapped onto the corresponding area in the feature map. Then, a region of interest (RoI) pooling layer uses max pooling to extract a fixed-length feature vector for each region proposal (Girshick, 2015). Finally, each feature vector is passed through fully connected layers that finally separated into two sibling output layers, softmax layer and bounding box regression layer (Girshick, 2015). The softmax layer predicts the softmax probability over K object classes plus one background class. The bounding box regression layer predicts bounding box coordinates for each object class (Girshick, 2015).

The loss function in fast R-CNN is a combination of classification loss and localization loss. Compared to R-CNN, Fast R-CNN is nine times faster at training time, 213 times faster at test time, and achieves higher mean average precision on PASCAL VOC 2012 (Girshick, 2015). However, the region proposals generation

process in fast R-CNN is slow and tedious due to the nature of the selective search algorithm, resulting in around 2s per image (Hmidani and Alaoui, 2022).

2.4.3 Faster R-CNN

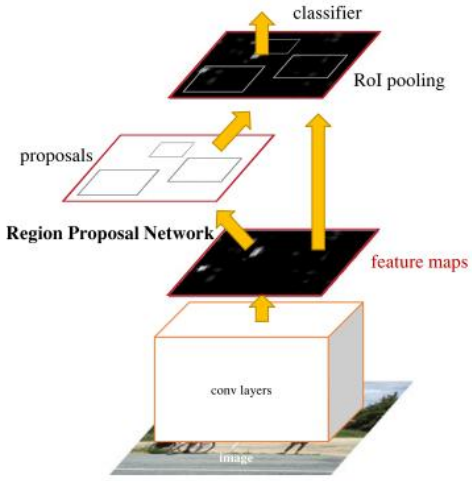

Figure 23: Faster R-CNN (Ren et al., 2016)

To address the region proposal computation bottleneck, Ren et al. (2016) introduce a region proposal network (RPN) that shares full-image convolutional features with the detection network. RPN is a fully convolutional network (FCN) that simultaneously predicts region proposals and objectness scores within an image (Ren et al., 2016). RPN eliminates the need for selective search in generating region proposals, resulting in faster detection. By integrating RPN with a fast R-CNN detector, faster R-CNN is invented.

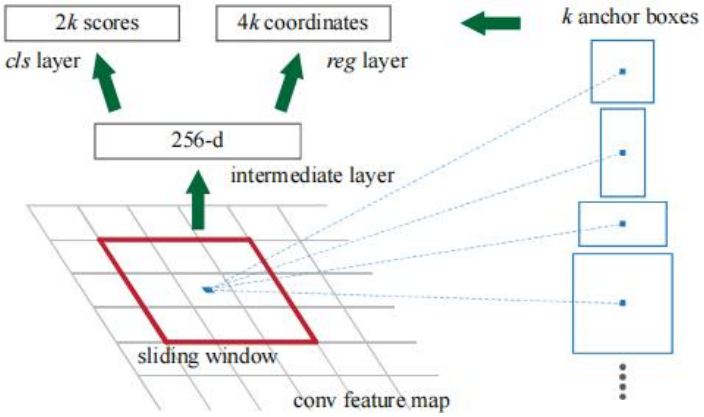

Figure 24: Process of generating region proposals using RPN (Ren et al., 2016)

The architecture of RPN is a n x n convolutional layer with ReLU activation function, followed by two siblings 1 x 1 convolutional layers for box regression and box classification respectively (Ren et al., 2016). Firstly, the input image is passed through a VGG16 CNN to extract a feature map. To generate region proposals, a small network which is fully connected to a n x n spatial window slides over the convolutional feature map (Ren et al., 2016). At each sliding window position on the feature map, RPN predicts multiple region proposals using a set of anchor boxes.

Each sliding window position is mapped to a low-dimensional vector, which is 512-dimensional for VGG16 (Ren et al., 2016). Then, those feature vectors are passed through a box regression layer and a box classification layer (Ren et al., 2016). The box regression layer outputs the coordinates for each region proposal while the box classification layer outputs two scores that estimate the probability of object/non-object for each proposal (Ren et al., 2016). The experiment shows that the faster R-CNN achieves detection speed of 5FPS with only 300 proposals per image, surpassing fast R-CNN (Ren et al., 2016). Although it is huge improvements on R-CNN families, 5fps is still insufficient for real-time detection.

2.5 Related works

In recent years, tremendous efforts have been made to integrate deep learning with defect detection problems across electronics, the manufacturing industry etc. The proposed solutions below can be further classified into defect classification, defect detection, and defect segmentation.

For instance, Scime et al. (2018) proposed an AlexNet-based CNN for defect classification in the additive manufacturing process through transfer learning. Scime et al. (2018) retrained the AlexNet CNN (Krizhevsky et al., 2012) to adapt AlexNet for power bed abnormalities classification. Although the CNN model achieves 97% classification accuracy, it can only classify images into specific defect categories instead of locating defects within images. It should be pointed out there is a potential class imbalance issue, in which normal cases are far more than defect cases. This also implies the lack of defect data at that time.

Yang et al. (2019) proposed a single-shot multi-box detector (SSD) for real-time tiny part defect detection on 0.8cm darning needles. Lv et al. (2020) also proposed an SSD-based end-to-end defect detection network to detect multiscale metallic surface defects. However, SSD tends to struggle with detecting tiny defects despite fast detection speed. Additionally, Lv et al. (2020) contributes a novel metallic surface defect dataset named GC10-DET, which includes 10 types of defect categories.

Cheng and Yu (2020) developed a new deep neural network named RetinaNet with different channel attention and adaptive spatial feature fusion methods to detect defects on steel surfaces. However, the experiment shows that RetinaNet struggling in detecting tiny defects, defects with huge size differences, and overlapping defects (Cheng and Yu, 2020). However, the performance of RetinaNet is affected by low-resolution and low-contrast image conditions (Cheng and Yu, 2020). The experiment shows that RetinaNet only offers 12.2 FPS of inference speeds, which cannot satisfy real-time defect detection.

The study by Westphal and Seitz (2021) proposed an Xception-based CNN and a VGG 16-based CNN for defect classification problems. The Xception CNN

architecture introduces depth-wise separable convolution (DWSC) layers which not only deal with spatial dimensions in neural networks but also with the number of colour channels of an input (Chollet, 2017). Both Xception-based CNN and VGG 16-based CNN are pre-trained on the ImageNet dataset and retrained on powder bed surface defect datasets (Westphal and Seitz, 2021). The experiment shows that the VGG-based CNN architecture achieves better defect classification performance than the Xception-based CNN architecture. (Westphal and Seitz, 2021). This implies that VGG-based CNN architecture has better feature extraction capability.

Zeng et al. (2022) proposed a multiscale feature fusion method called atrous spatial pyramid pooling-balanced-feature pyramid network (ABFPN) to detect multi-scale tiny defects on the printed circuit board (PCB). The ABFPN uses atrous convolutional operators with varying dilation rates and balanced modules to maximize latent information during feature fusion. The ablation study shows that the proposed feature fusion method improves the detection accuracy of faster R-CNN by 2.3%mAP on PCB defect detection.

Huang et al. (2022) proposed a YOLOv4-based model (SO-YOLO) with a shallow feature fusion network to detect tiny defects on the chip surface. Unlike the original YOLOv4, Huang et al. (2022) modified the path aggregation network and fusion network for SO-YOLO to retain shallow feature information. This prevents shallow feature loss during feature extraction from affecting detection accuracy. The experiment shows that modification improves the mAP by 4.6% compared to the original YOLOv4.

Duan et al. (2022) introduced an improved faster R-CNN for metal surface defect detection. In this work, the original VGG16 backbone network of faster R-CNN is replaced by a residual network with deformable convolution and a multi-feature layer to generate more informative feature maps. In the region proposal network, region of interest (ROI) alignment is replaced by ROI pooling to avoid matching errors between the feature map and ROI (Duan et al., 2022). Besides that, Duan et al. (2022) replaced the fully connected layer with the ResNet layer in the classification network to improve feature extraction. The experiment shows that the modifications above improve the defect detection performance by 6.3% of mean average precision compared to the original faster R-CNN (Duan et al., 2022). However, faster R-CNN is relatively slow when compared to one-stage object detection like YOLO and SSD despite better precision.

To achieve faster defect detection with high precision, Wang et al. (2022) proposed an improved YOLOv7 for steel strip surface defect detection. Based on YOLOv7, several modifications are made. Firstly, a de-weighted bidirectional feature pyramid network is combined with YOLOv7 to improve feature fusion and reduce information loss during the convolutional process (Wang et al., 2022). Secondly, efficient channel attention (Wang et al., 2020) is combined with the YOLOv7

backbone to focus on important feature channels. Thirdly, the original bounding box loss function is replaced by the SIoU (Gevorgyan, 2022) loss function, in which the penalty term takes the vector angle between regression into account. Experiments by Wang et al. (2022) show that improved YOLOv7 outperforms faster R-CNN, SSD, and YOLOv5 in both processing speed and mean average precision. However, the model struggles to detect defects with lighter colours.

To detect internal defects, the research conducted by Wong et al. (2022) introduced a U-Net-based deep convolutional network for defect segmentation on X-ray computed tomography (XCT) image. U-Net is a CNN architecture initially developed for biomedical image segmentation (Ronneberger et al., 2015). With transfer learning techniques, the U-Net is adapted to defect segmentation problems.

Zhou et al. (2023) made improvements to the YOLOv5 model for better metal surface defect detection, focusing on enhancing the detection accuracy of small defects. In their work, the C3 module in the original YOLOv5 structure is replaced with the CSPlayer module to improve the neural network's flexibility while maintaining the light weight of the model. The global attention mechanism is applied by combining channel and spatial attention to improve feature expression, achieve better small target detection, and better capture the complicated features of images.

To improve object detection on multiple targets, Yu et al. (2023) proposed a multiple-attentional path aggregation network (APAN) for marine object detection in degraded underwater conditions. The multi-attention mechanism (Vaswani et al., 2017) further improves the detection accuracy of multiple marine object detection, and such an attention mechanism is promising in detecting multiple defects.

2.6 Identification of research gap

Although deep learning detection made significant advancements in feature recognition, detecting tiny defects is still challenging because current small object detection methods don't fully mine latent information like stronger semantic features from the feature map and more accurate location (Zeng et al., 2022). Although some methods demonstrate competitive accuracy in tiny defect detection, insufficient latent speed cannot satisfy real-time defect detection. After literature reviews, it is found that the latest YOLOv10 shows exceptional performance on COCO but there is a lack of empirical experiments in defect detection. Motivated by the discussions above, this study aims to develop a YOLOv10-based model for defect detection. A research question is also defined as follows:

'Explore the capability of the YOLOv10 model in metal surface defect detection.'

CHAPTER 3: METHODOLOGY

3.1 Overall Approach

The choice of research approach can be categorized into quantitative research and qualitative research. Quantitative research involves systematic empirical investigation via statistical, mathematical, numerical or computational techniques (Given, 2008). On the other side, qualitative research is a scientific method of observation to gather non-numerical data (Babbie, 2013). Qualitative research answers why and how a certain phenomenon may occur (Lune and Berg, 2016). To propose a deep learning model which learns discriminative features from metal surfaces and detects defects, image data of metal surface defects is analysed and modelled through computational techniques. Therefore, this study is classified as quantitative research.

3.2 Methods

	Exploratory Research	**Conclusive Research**
Objective	To provide insights and understanding	To test specific hypotheses and examine the relationship
Information required	Loosely defined	Clearly defined
Research process	Flexible and unstructured	Formal and structured
Findings/Results	Tentative	Conclusive

Table 3: Difference between exploratory research design and conclusive research design (CS5704, Lecture 3)

The choice of research design includes exploratory research design and conclusive research design. Table 3 explains the differences between the two research designs. There are related defect detection research and formal processes to propose a deep learning model. For instance, data collection, data pre-processing, discovering suitable object detection framework, adopting framework into model, training model, testing model, fine-tuning model etc. Therefore, this study implements a conclusive research design to discover whether the latest YOLO v10-based defect detection model can achieve better defect detection performance through experiments.

The choice of data collection can be categorized into qualitative data and qualitative data. Qualitative data collection involves interviews, diaries, participant observation, field notes, policy documents, Twitter feeds, blogs, newspapers, and speeches (CS5704, Lecture 3). As opposed, quantitative data collection involves surveys, questionnaires, experiments, clinical trials, and instrument-based measurement (CS5704, Lecture 3). Quantitative data collection is not limited to qualitative data, but they are converted into numerical form for statistical analysis (CS5704, Lecture 3). To build the defect detection model, image data of metal surface

defects with labelled annotation are required. This study collects quantitative data, as metal surface defects are captured by an industrial camera and annotation is labelled using the computer.

The source of data includes primary data and secondary data. Primary data refers to data collected by the author, while secondary data refers to data collected by other researchers (CS5704, Lecture 3). The collection of primary images of defective parts is challenging due to the time-consuming image data acquisition process and expensive annotation labelling process by industrial experts. As there are several public benchmark datasets available on the Kaggle platform, this study uses secondary quantitative data sources.

3.2.1 Mapping objective to methods

Objective 1: Discovering suitable deep learning approaches for defect detection.

The YOLO series has achieved a balance between processing speed and detection accuracy (Tan et al., 2024). In May 2024, Wang et al. released YOLOv10 for real-time object detection. To explore the capability of the latest YOLOv10 on defect detection, this study adopts a pre-trained YOLOv10 model on COCO object detection to metal surface defect detection through transfer learning.

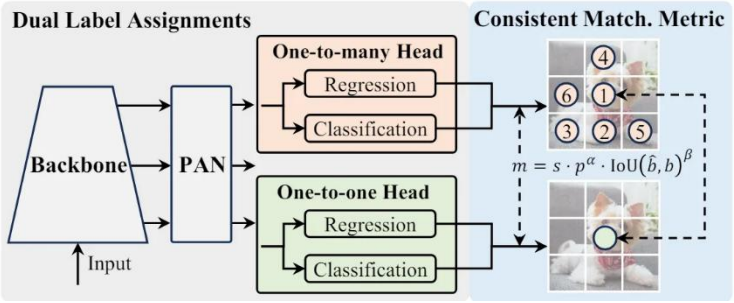

Figure 25: The general architecture of YOLOv10 (Wang et al., 2024)

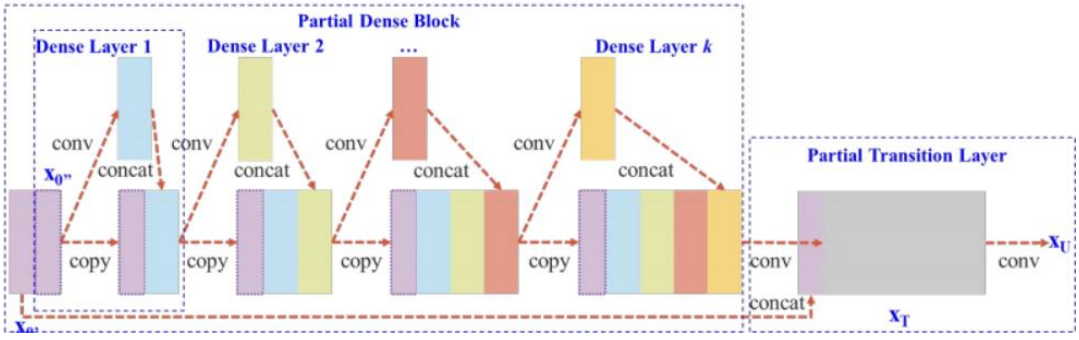

Figure 26: Architecture of CSPNet (Wang et al., 2020)

The architecture of YOLOv10 is like the previous YOLO framework. The main difference between YOLOv10 and the previous YOLO version are use of dual assignment. The backbone network of YOLOv10 is a CSPNet. The neck network of YOLOv10 is a path aggregation network, which combines features from different scales. During training, the one-to-many head predicts multiple boxes for each object

to provide rich supervisory signals. During inference, the one-to-one head produces the best prediction for each object, replacing NMS used in previous YOLO versions (Wang et al., 2024).

Model	Test Size	#Params	FLOPs	AP^{val}	Latency
YOLOv10-N	640	2.3M	6.7G	38.5%	1.84ms
YOLOv10-S	640	7.2M	21.6G	46.3%	2.49ms
YOLOv10-M	640	15.4M	59.1G	51.1%	4.74ms
YOLOv10-B	640	19.1M	92.0G	52.5%	5.74ms
YOLOv10-L	640	24.4M	120.3G	53.2%	7.28ms
YOLOv10-X	640	29.5M	160.4G	54.4%	10.70ms

Table 4: Performance of YOLO model on COCO dataset (Ultralytics, 2024)

Table 4 shows the performance of various YOLOv10 model scales on the Microsoft COCO dataset. Microsoft COCO is a well-known benchmark dataset in object detection (Lin et al., 2014). Considering the average precision and latency, this study transfers the YOLOv10-X model to the metal surface defect detection problem. The YOLOv10-X model offers the highest average precision with a competitive latency speed, of only 10.70ms per image.

Objective 2: Collecting defect image data

Primary image data collection is expensive, time-consuming, and requires industrial experts to label and annotate defects. As there are public benchmark datasets for defect detection, the 'NEU-DET' dataset and the 'GC10-DET' dataset are downloaded from kaggle.com. More detailed explanations can be found in Section 3.4 Dataset.

Objective 3: Pre-processing defect image data

This study trains YOLOv10 on the NEU-DET dataset and GC10-DET dataset respectively. Both datasets undergo the same data pre-processing pipeline. Annotations of both the 'NEU-DET' and 'GC10-DET' datasets are in COCO format, but the YOLO model expects annotation in YOLO format. Hence, annotations are converted to YOLO format to match the YOLO model.

NEU-DET	**No. of images before data augmentation**	**No. of images after data augmentation**
Training set	1260	3780
Validation set	270	270
Testing set	270	270
GC10-DET		
Training set	1610	2859
Validation set	345	345
Testing set	345	345

Table 5: No. of images in each set before vs after data augmentation

After format conversion, both the 'NEU-DET' and 'GC10-DET' datasets are split into 75% training set, 15% validation set, and 15% testing set through random sampling. To improve the robustness and generalisation ability of the defect detection model, the training data is expanded through data augmentation techniques, which include random rotation 90 degrees, random vertical and horizontal flipping, random brightness contrast, and Gaussian blur. Each image undergoes data augmentation twice. After data augmentation, the training set is expanded to 3 times its original size. The validation set and testing set remain unchanged, so the trained model is validated and tested on real data.

Objective 4: Training the defect detection model

```
                 from  n    params  module                                       arguments
  0                -1  1      2320  ultralytics.nn.modules.conv.Conv             [3, 80, 3, 2]
  1                -1  1    115520  ultralytics.nn.modules.conv.Conv             [80, 160, 3, 2]
  2                -1  3    436800  ultralytics.nn.modules.block.C2f             [160, 160, 3, True]
  3                -1  1    461440  ultralytics.nn.modules.conv.Conv             [160, 320, 3, 2]
  4                -1  6   3281920  ultralytics.nn.modules.block.C2f             [320, 320, 6, True]
  5                -1  1    213120  ultralytics.nn.modules.block.SCDown          [320, 640, 3, 2]
  6                -1  6   4604160  ultralytics.nn.modules.block.C2fCIB          [640, 640, 6, True]
  7                -1  1    417920  ultralytics.nn.modules.block.SCDown          [640, 640, 3, 2]
  8                -1  3   2712960  ultralytics.nn.modules.block.C2fCIB          [640, 640, 3, True]
  9                -1  1   1025920  ultralytics.nn.modules.block.SPPF            [640, 640, 5]
 10                -1  1   1545920  ultralytics.nn.modules.block.PSA             [640, 640]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 12            [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 13                -1  3   3122560  ultralytics.nn.modules.block.C2fCIB          [1280, 640, 3, True]
 14                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 15            [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 16                -1  3   1948800  ultralytics.nn.modules.block.C2f             [960, 320, 3]
 17                -1  1    922240  ultralytics.nn.modules.conv.Conv             [320, 320, 3, 2]
 18           [-1, 13]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 19                -1  3   2917760  ultralytics.nn.modules.block.C2fCIB          [960, 640, 3, True]
 20                -1  1    417920  ultralytics.nn.modules.block.SCDown          [640, 640, 3, 2]
 21           [-1, 10]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 22                -1  3   3122560  ultralytics.nn.modules.block.C2fCIB          [1280, 640, 3, True]
 23       [16, 19, 22]  1   4404300  ultralytics.nn.modules.head.v10Detect        [10, [320, 640, 640]]
YOLOv10x summary: 688 layers, 31674140 parameters, 31674124 gradients, 171.1 GFLOPs
```

Figure 27: Summary of pre-trained YOLOv10-X model

Before training, the YOLOv10-X model is imported from Ultralytics and the pre-trained weight on COCO object detection is downloaded from the GitHub repository. The pre-trained weight is used to initialise the YOLOv10-X model. Figure 27 shows the model summary, the proposed model consists of 688 layers, 31674140 parameters, 31674124 gradients and 171.1 GFLOPs.

Hyperparameter	Value
Learning rate	0.001
Momentum	0.937 (Default)
Weight decay	0.0005 (Default)
Batch size	8
Patience	100
Training epochs	300
freeze	None
Optimizer	Stochastic gradient descent (SGD)
Image size	640*640 (Default)

Table 6: Training hyperparameter

As the NEU-DET dataset and GC10-DET dataset are similar, the model is trained on the same hyperparameters setting. Momentum is used to accelerate gradient descent while the weight decay helps prevent overfitting. Batch size is set to 8 to boost training while ensuring the stability of gradient descent. Patience is set to 100, indicating that if no improvement over the last 100 epochs, training is early stopped to avoid the overfitting issue. Transfer learning involves freezing certain layers of the pre-trained model to prevent them from being updated during the training process. As those layers have been trained on large and diverse datasets, freezing helps to keep the knowledge learned by these layers to extract features. In this work, the freezing is set to false because testing shows that the model performs better without freezing layers. Input images are resized to 640*640 before feeding to the model, as larger images cost more training time and GPU resources.

Objective 5: Test and verify the effectiveness of the defect detection model.

After training, the YOLOv10-based defect detection model is evaluated on a testing set. Performance in precision, recall, mean average precision (mAP) and frame per second (FPS) are recorded in Chapter 4 Result. The proposed model is compared with state-of-the-art metal surface defect detection model such as RetinaNet, SSD, faster R-CNN, improved faster R-CNN, improved YOLOv5, and improved YOLOv7.

3.3 Instruments

This study is implemented on the Google Collab platform with Python 3.0. The hardware configuration includes one NVIDIA T4 graphic card with 22.5 GB of VRAM.

3.4 Dataset

The experiment is implemented on the 'GC-10 DET' dataset (Lv et al., 2020) and the 'NEU-DET' dataset (Song and Yan, 2013), which are public benchmark datasets for metal surface defect detection.

3.4.1 NEU-DET

Defect types	No. of images
Crazing	300
Inclusion	300
patches	300
Pitted surface	300
Rolled in scale	300
Scratches	300

Table 7: No. of images for each defect class in the NEU-DET dataset

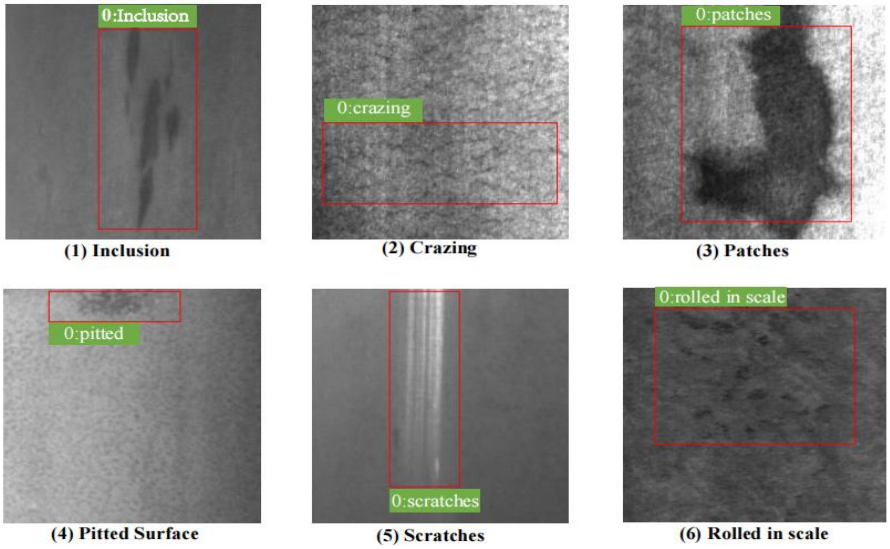

Figure 28: Defect instances on NEU-DET (Lv et al., 2020)

The 'NEU-DET' dataset was contributed by the team at Northeastern University (He et al., 2019). This dataset consists of 1800 grey-scale images from 6 defect classes: inclusion, crazing, patches, pitted surface, scratches, and rolled-in scale. The resolution of the image is 200*200. Each defect class contains 300 images, this dataset doesn't have a data imbalance issue. Detailed explanations of each defect are shown as follows.

- **Inclusion:** Small spots, fish scale morphology, strip morphology, and irregular block distribution on the strip surface.
- **Crazing:** Cracks on the metal surface.
- **Patches:** A segment of metal distinguished from others.
- **Pitted surface:** Tiny but deep corrosion that penetrates the metal surface.
- **Scratches:** Mark of abrasion on the metal surface.
- **Rolled-in scale:** Mill scale on the metal surface.

3.4.2 GC10-DET

Defect types	No. of images
punching	329
Welding line	513
Crescent gap	265
Water spot	354
Oil spot	569
Silk spot	884
Inclusion	347
Rolled pit	85
Crease	74
Waist folding	150

Table 8: No. of images in each defect class in the GC10-DET dataset

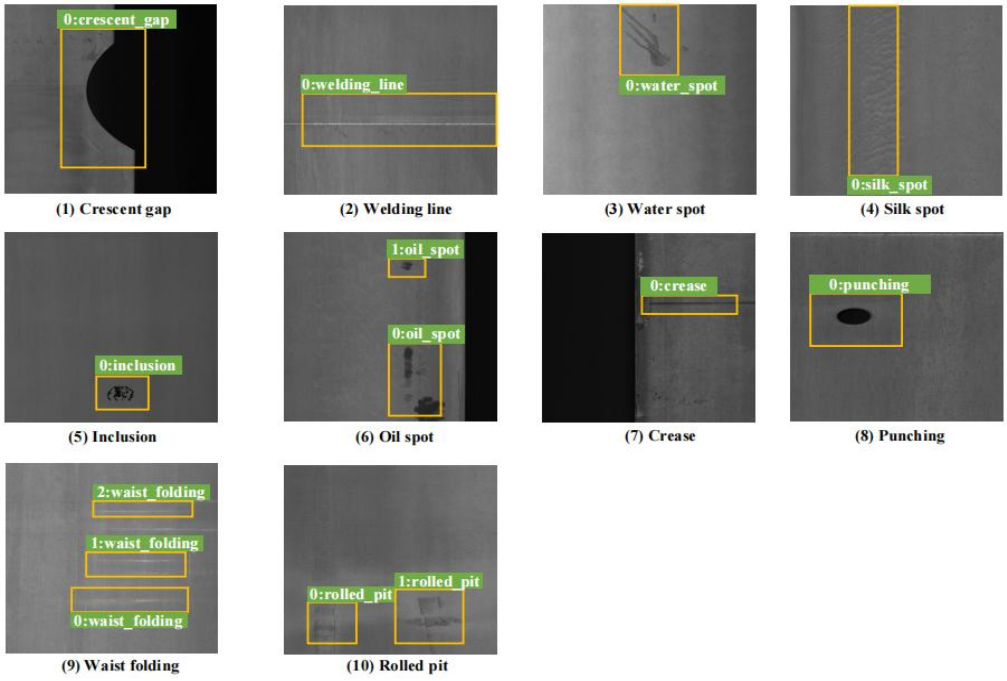

Figure 29: Defect instances on GC10-DET (Lv et al., 2020)

The "GC10-DET" dataset consists of 2312 grey-scale images with 10 classes of metal surface defect (Lv et al., 2020). The resolution of the image is 4096*1000. There is a potential class imbalance issue in the GC-10 DET dataset, especially for rolled pit and crease. The number of images in those defect classes is significantly lower than in other classes. Detailed explanations for each defect class are shown as follows.

- **Punching:** Unwanted punching hole on steel strip.
- **Welding line:** Produced when two strips are welded together. This is not a defect but requires automatic detection and tracking to be avoided in subsequent cuts.

- **Crescent gap:** Cutting defects with a half-circle shape.
- **Water spot:** Spot occurs during the drying process.
- **Oil spot:** Spot resulting from mechanical lubricant contamination
- **Silk spot:** A localised or continuous wave-like defect on a strip surface, caused by irregular temperature and pressure of the roller.
- **Inclusion:** Small spots, fish scale morphology, strip morphology, and irregular block distribution on the strip surface.
- **Rolled pit:** Intermittent bulges or pits on the strip surface that are punctate, linear or flaky. They are dispersed over the strip length or section, mainly due to damaged work roll or tension roll.
- **Crease:** Vertical transverse fold with spacing along the strip, resulting by localised yielding in the direction of movement during the uncoiling process.
- **Waist folding:** Pronounced folds resembling wrinkles, indicating excessive local distortion of the defect. The cause is attributed to low carbon levels.

3.5 Ethics

This research is conducted without using sensitive or confidential information. Ethical approval is granted by the Brunel Research Ethics Committee. The letter of confirmation can be found in Appendix A.

3.6 Limitation

The limited data scale and diversity of defect categories in existing datasets constrain the deployment of the defect detection model. To develop a robust and comprehensive defect detection model, an extensive and diverse dataset for surface defects is essential. Such a surface defect dataset should cover a wide range of defect types and rich instances to improve the model's generalisation ability in real-world scenarios.

CHAPTER 4: RESULTS

4.1 Evaluation metrics

When the defect class prediction is correct and the intersection over union (IoU) meets or exceeds 0.6 thresholds, the detection is deemed accurate. To evaluate the defect detection performance of the YOLOv10 model, performance metrics like recall, precision, mean average precision, frame per second, and F1 score are adopted. Precision measures the proportion of true positive detections among all positive detections made by the model. A high precision indicates when the model predicts a defect, it is typically correct. The precision can be expressed as:

$$\text{Precision}=\frac{TP}{TP+FP} \qquad (4)$$

Where TP and FP represent the number of true positives and false positives, respectively. Recall represents the ratio of true positive detections out of all actual defects in the dataset. A high recall indicates that the model can detect most of the actual defects. The recall can be expressed as:

$$\text{Recall}=\frac{TP}{TP+FN} \qquad (5)$$

Where TP and FN represent the number of true positives and false negatives, respectively. Average precision represents the average detected precision for each defect class and can be computed by the area under the precision-recall curve.

$$\text{Average precision (AP)}=\int_0^1 p(r)dr \qquad (6)$$

Where p(r) indicates precision as a function of recall r. Mean average precision (mAP) is the average detected precision for all defect classes and it can be expressed as:

$$\text{Mean average precision (mAP)}=\frac{1}{N}\sum_{i=1}^{N} AP_i \qquad (7)$$

Where N is the number of defect classes and AP_i indicates average precision for the i-th class. Frame per second (FPS) measures how many images the model can perform defect detection in one second. A high FPS indicates the model is fast at defect detection and it can be expressed as:

$$\text{Frame per second (FPS)}=\frac{1}{Time\ per\ image} \qquad (8)$$

The F1 score is the harmonic mean of precision and recall. A high F1 score indicates a well-balanced performance, where the model has the best trade-off between recall and precision. As opposed, a low F1 score indicates that the model struggles with precision, recall, or both. The F1 score can be expressed as:

$$\text{F1}=2 \text{ x } \frac{\text{Pr}\,e\,cision \times Recall}{\text{Pr}\,e\,cision+Recall} \qquad (9)$$

4.2 Results on NEU-DET dataset

4.2.1 Results

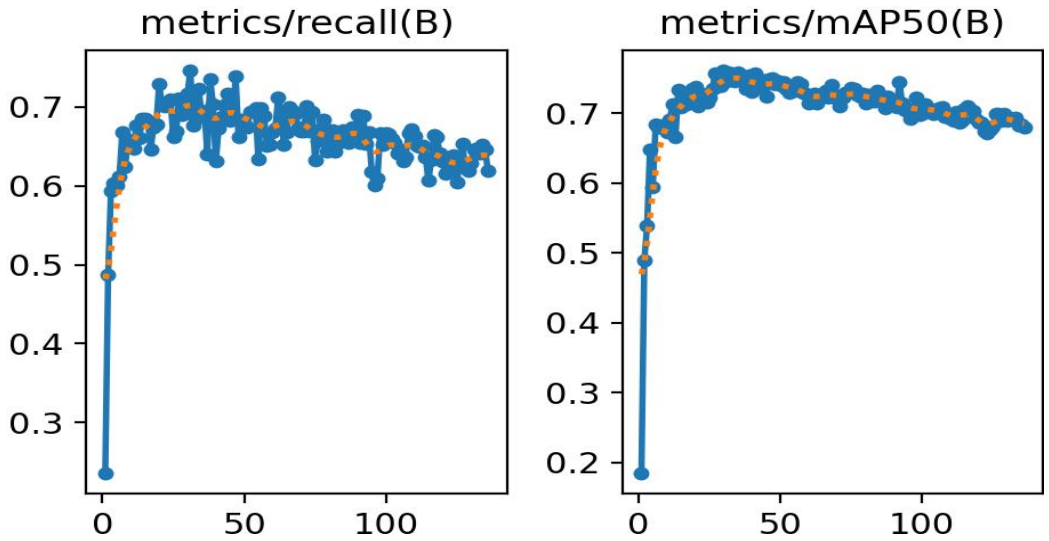

Figure 30: Recall and mean average precision over the training process on NEU-DET

After 6.8 hours of training, the process terminates at epoch 136 as no improvement is observed over the past 100 epochs. Figure 30 shows that the model converges very fast at earlier epochs despite the learning rate is set to 0.001. Such a situation might be due to a very high learning rate. However, my testing shows that using a smaller learning rate can achieve more stable convergence, but it didn't help to improve the mean average precision and recall. Hence, the learning rate is kept at 0.001, offering a balance between convergence speed and performance.

Defect types	Instances	Precision	Recall	mAP50	mAP50-95
Crazing	101	0.583	0.374	0.457	0.189
Inclusion	153	0.801	0.817	0.854	0.502
Patches	136	0.914	0.864	0.928	0.615
Pitted_surface	44	0.851	0.841	0.875	0.633
Rolled-in_scale	111	0.589	0.593	0.653	0.316
Scratches	76	0.797	0.829	0.897	0.562
All class	**621**	**0.756**	**0.72**	**0.777**	**0.469**
FPS	**32.89**				
F1 score	**0.74**				

Table 9: Defect detection performance on NEU-DET

The trained model is evaluated on a testing set with 270 images and the result is recorded in Table 4.1. Instances are the actual number of instances in each defect class. MAP50 indicates the mean average precision at 50% intersection over union (IoU) while mAP50-95 represents the mean average from 50% to 95% IoU. The result shows that the model tends to struggle in detecting crazing and rolled-in scale, resulting in low recall and mAP compared to other defect classes.

4.2.2 Visualisation of results

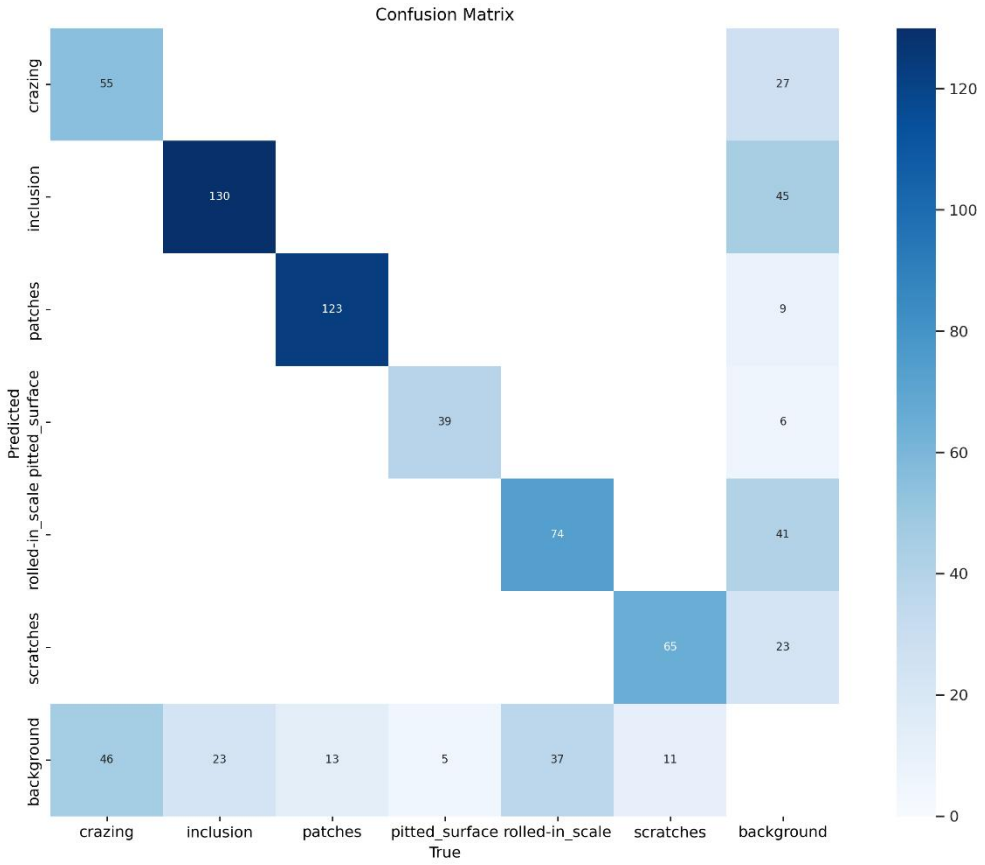

Figure 31: Confusion matrix for NEU-DET

The confusion matrix visualizes how well the model's predictions match the true labels for each class in counts. The diagonal elements show the number of instances where the predicted label matches the true label. The y-axis represents defect classes predicted by the model while the x-axis represents the actual true defect classes in the dataset. The colour intensity represents the number of instances, where darker shades indicating higher number of instances.

In summary, the model correctly detected 55 instances of crazing, 130 instances of inclusion, 123 instances of patches, 39 instances of pitted surface, 74 instances of rolled-in scale, and 65 instances of scratches. The model effectively distinguishes between defect classes, as there a no instances mis-detected as different defect classes.

However, 46 instances of crazing, 23 instances of inclusion, 13 instances of patches, 5 instances of pitted surface, 37 instances of rolled-in scale, and 11 instances of scratches are mis-detected as background. Also, a total of 151 instances of background are mis-detected as defect classes.

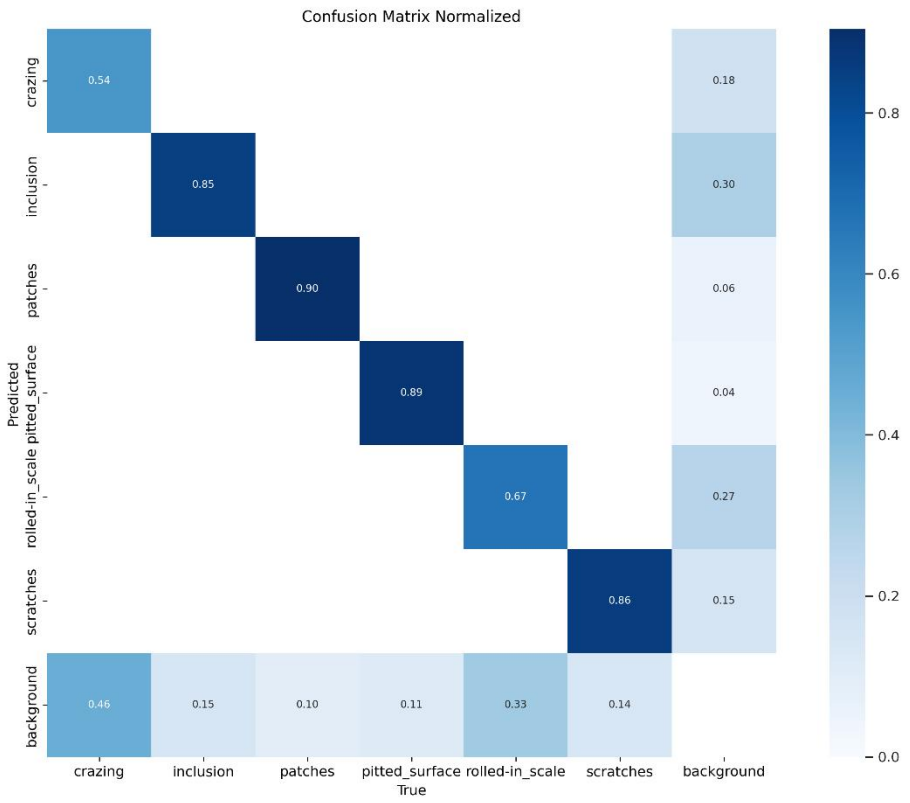

Figure 32: Normalised confusion matrix for NEU-DET

The normalised confusion matrix shows how well the model's predictions match the true labels for each class in proportions. The model performs very well on inclusion, patches, pitted surface, and scratches detection, with high proportions on the diagonal (higher than 80%) and low misdetection. However, the model struggles to distinguish background from defect classes, especially for crazing, rolled-in scale, and inclusion.

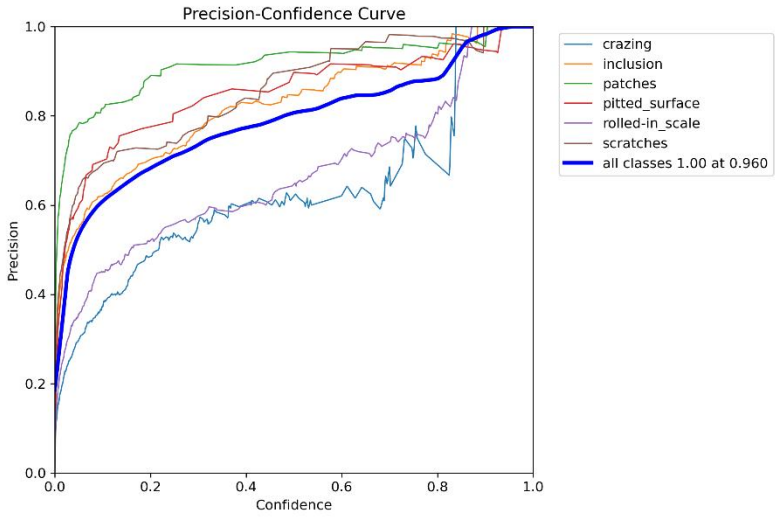

Figure 33: Precision confidence curve for NEU-DET

The precision confidence curve shows the precision value at different confidence thresholds. At a high confidence threshold, the precision is high because the model is very selective and predicts only when the model is greatly confident in its predictions, resulting in more accurate predictions. However, recall is low at high

confidence thresholds because the model becomes more selective, makes fewer predictions and might miss actual positive instances, so it is important to achieve the best trade-off between precision and recall. Classes like ‘patches’, ‘inclusion’, ‘pitted surface’ and ‘scratches’ show high precision across most confidence levels. This indicates that the model is relatively accurate for these defect classes even at a low confidence level.

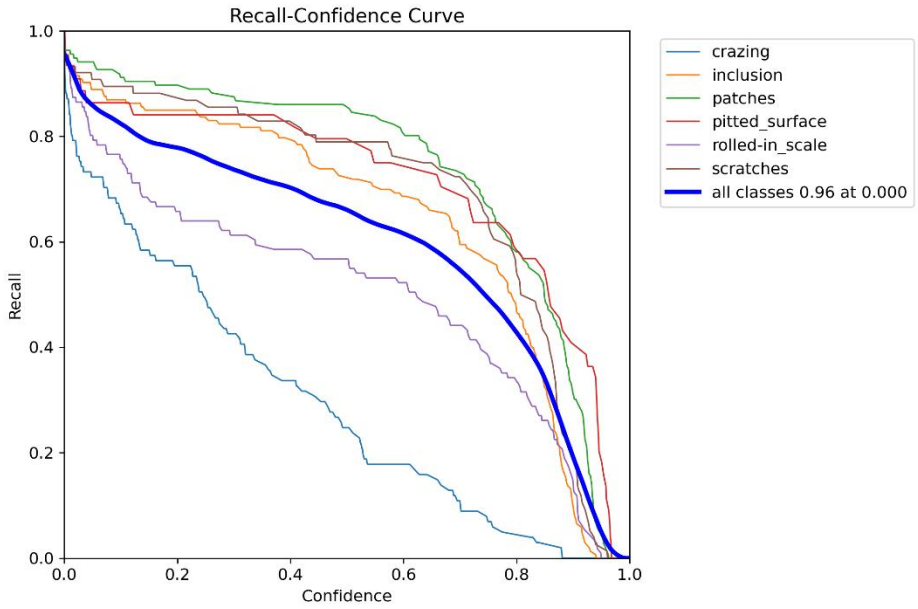

Figure 34: Recall-confidence curve for NEU-DET

The recall-confidence curve plots the recall value across different confidence thresholds. At low confidence thresholds, the recall is generally high because the model is less selective and makes more prediction, which help the model to capture more actual positive instances. However, the precision will be affected at low confidence threshold because the model makes more prediction and tends to allow more false positive instances in its prediction. Classes like 'patches', 'inclusion', and 'pitted- surface' maintain high recall even as confidence increases, indicating that the model can capture these defect classes correctly even the model become more selective and makes less predictions.

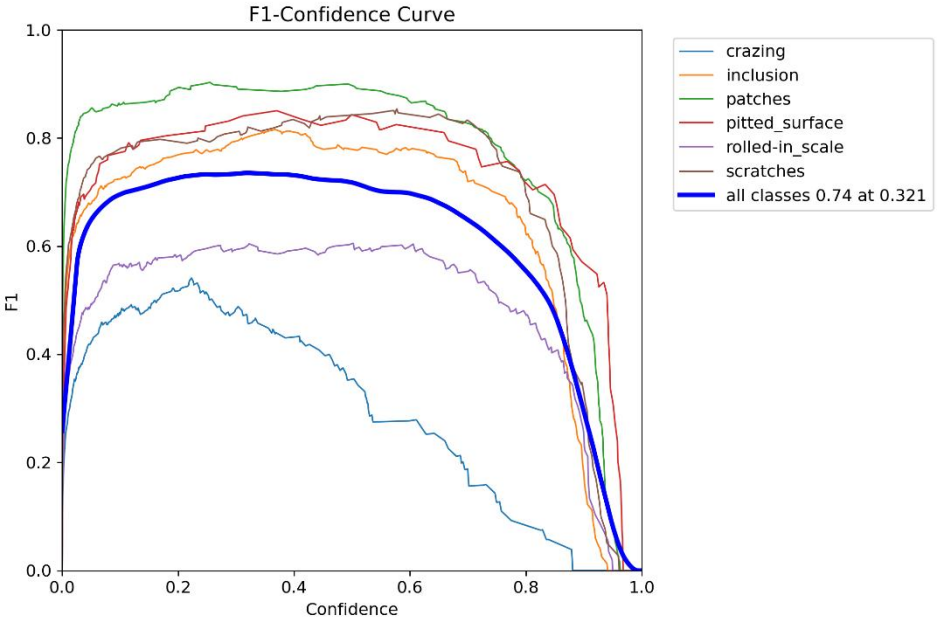

Figure 35: F1-confidence curve for NEU-DET

The F1-confidence curve plots the F1 score across different confidence thresholds. The peak F1 score of 0.74 occurs at 0.321 confidence threshold, where the model has the best overall performance on precision and recall across all defect class.

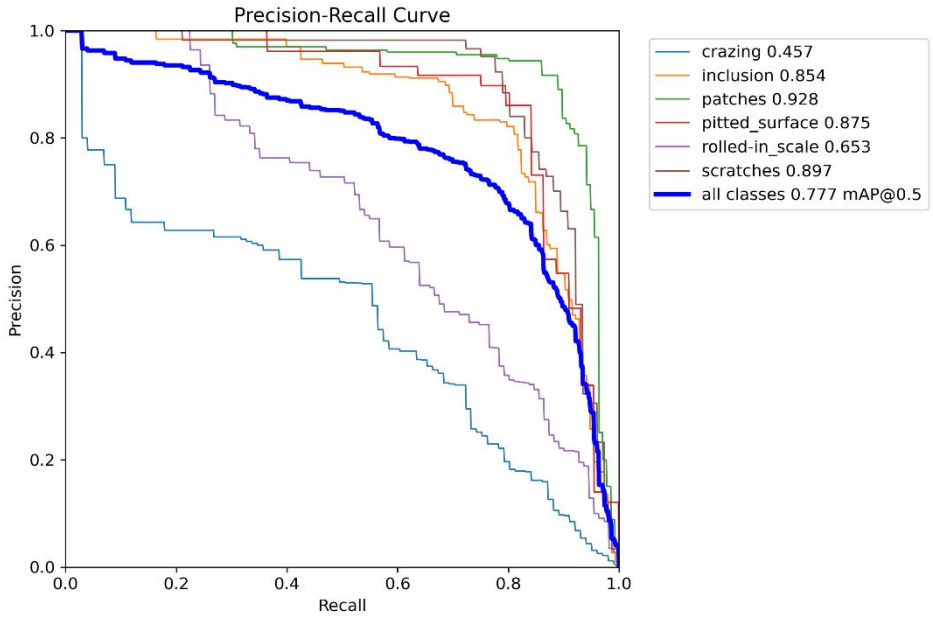

Figure 36: Precision-recall curve for NEU-DET

The precision-recall curve shows that the model achieves better detection performance on defect classes like patches, scratches, inclusion, and pitted surfaces as their curves are closer to the top-right corner of the graph. This indicates that the model is both sensitive and accurate in detecting those defects. However, the model struggles in capturing and detecting crazing and rolled-in scale. A mean average precision of 0.777 is achieved at 50% IoU thresholds.

4.2.3 Comparison between actual labels and model predictions

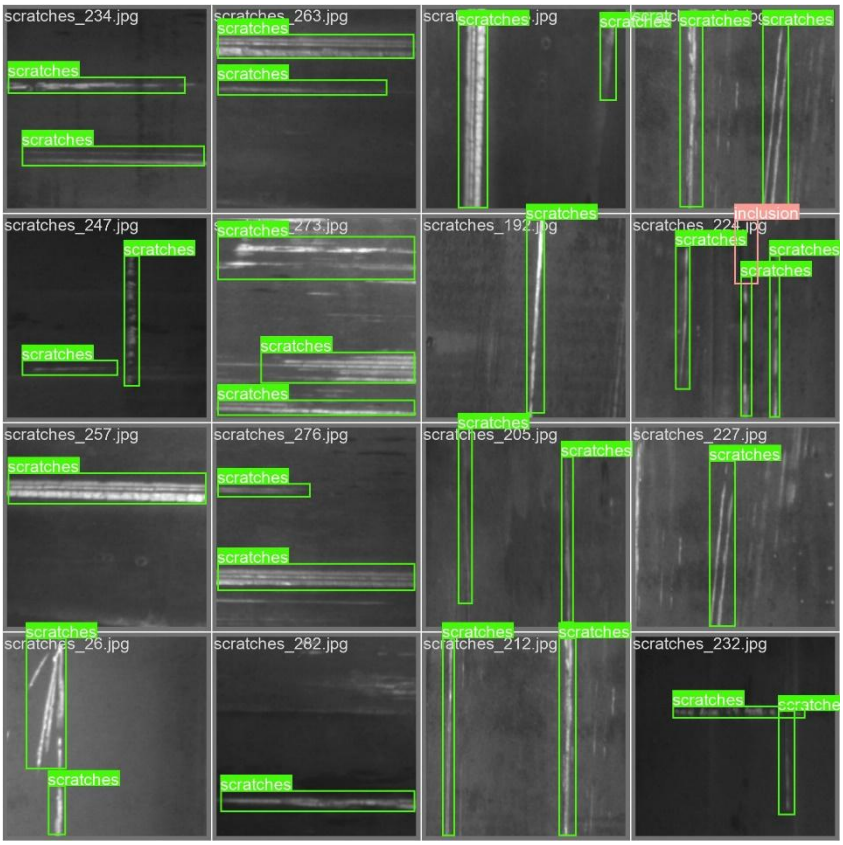

Figure 37: Defect Labels on testing batch 0

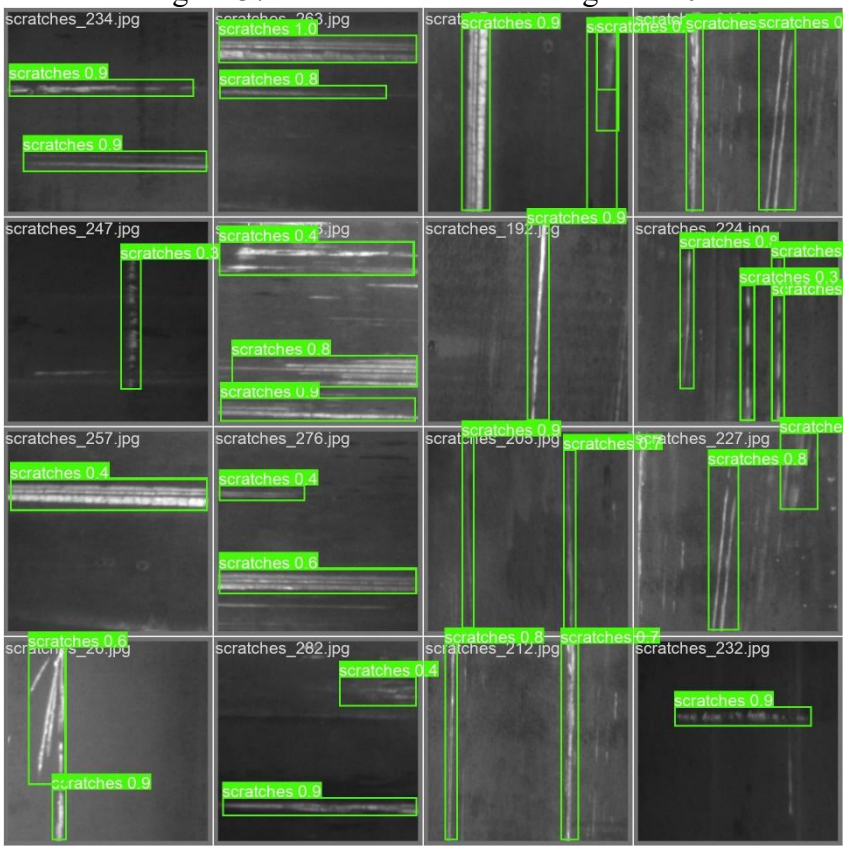

Figure 38: Model predictions on testing batch 0

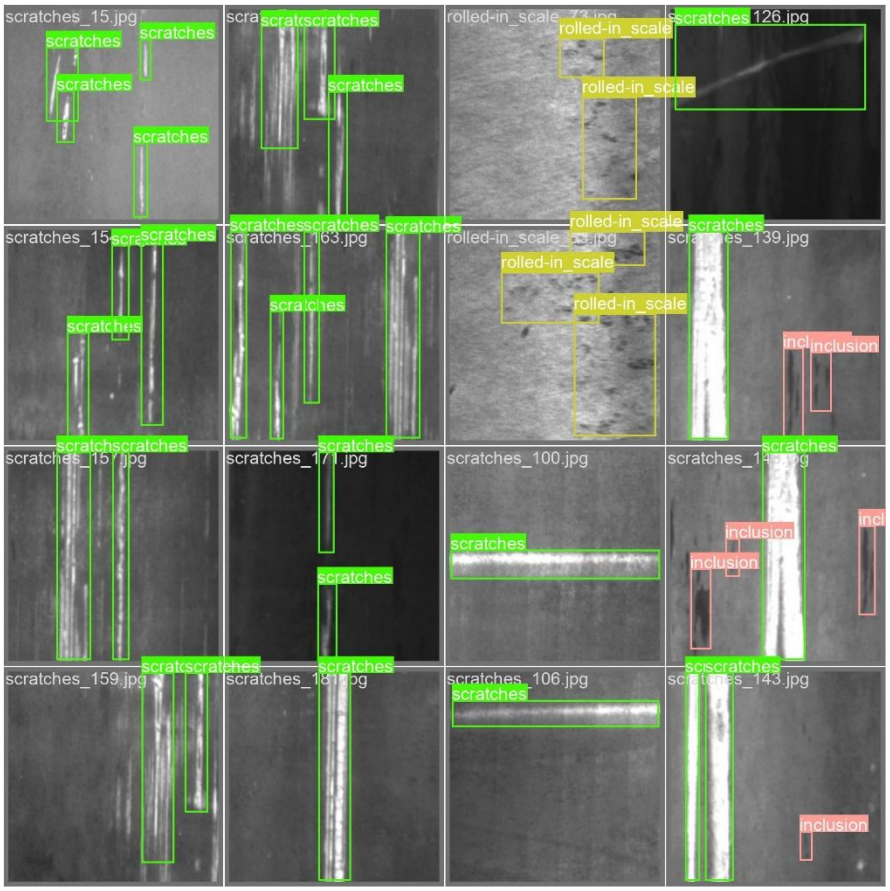

Figure 39: Defect labels on testing batch 1

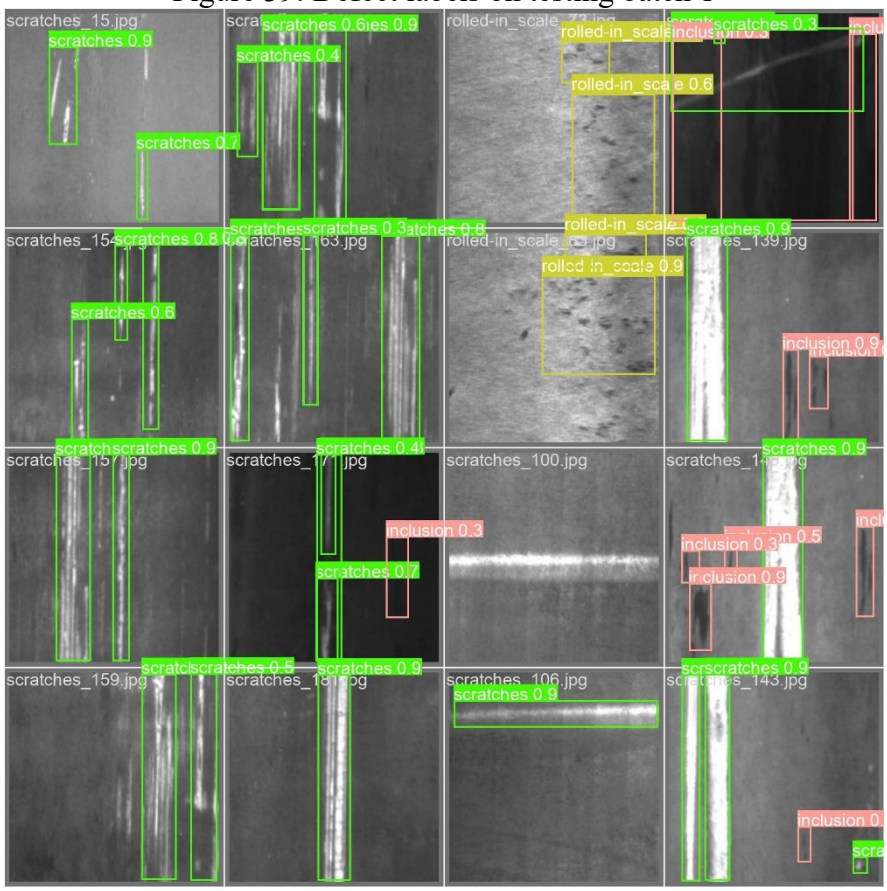

Figure 40: Model's predictions on testing batch 1

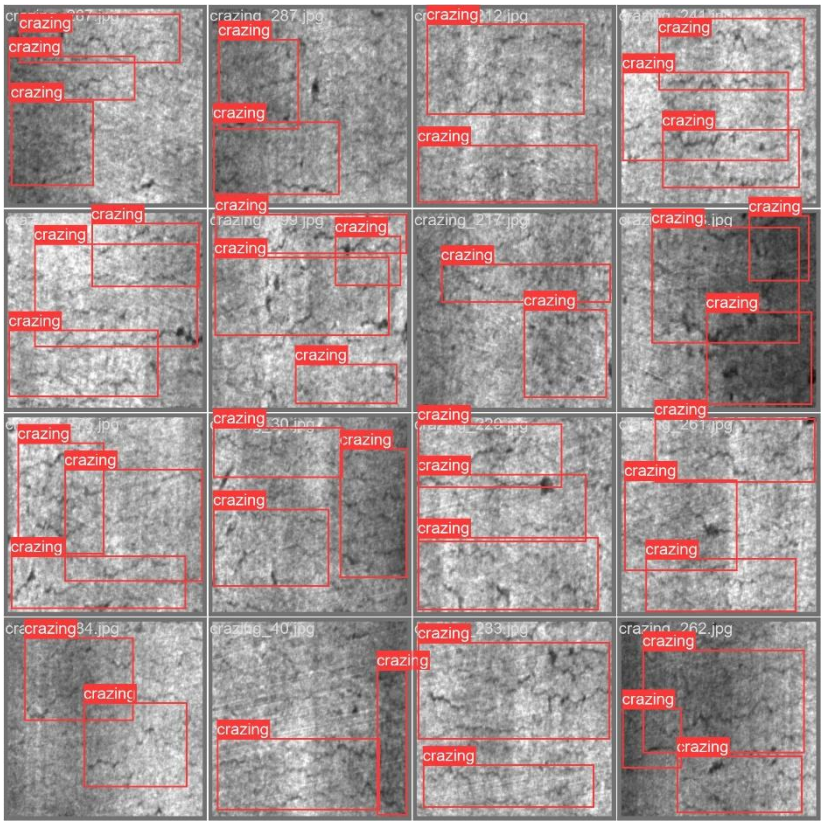

Figure 41: Defect labels on testing batch 2

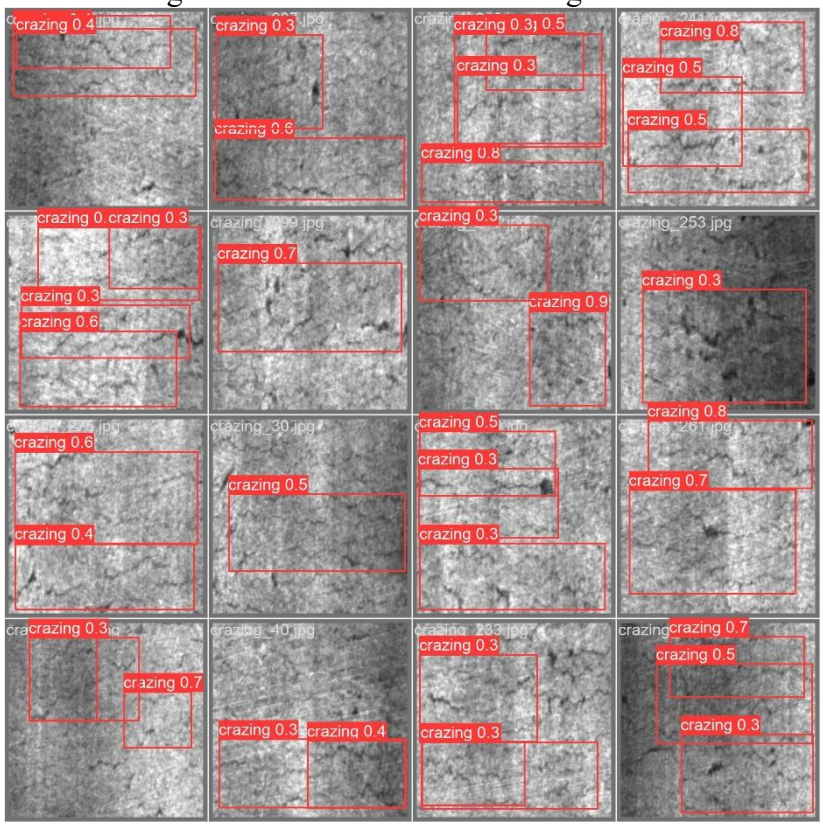

Figure 42: Model's predictions on testing batch 2

The comparison shows that the model detects and locates most defects, but it tends to struggle in detecting crazing defects.

4.3 Results on GC10-DET dataset

4.3.1 Results

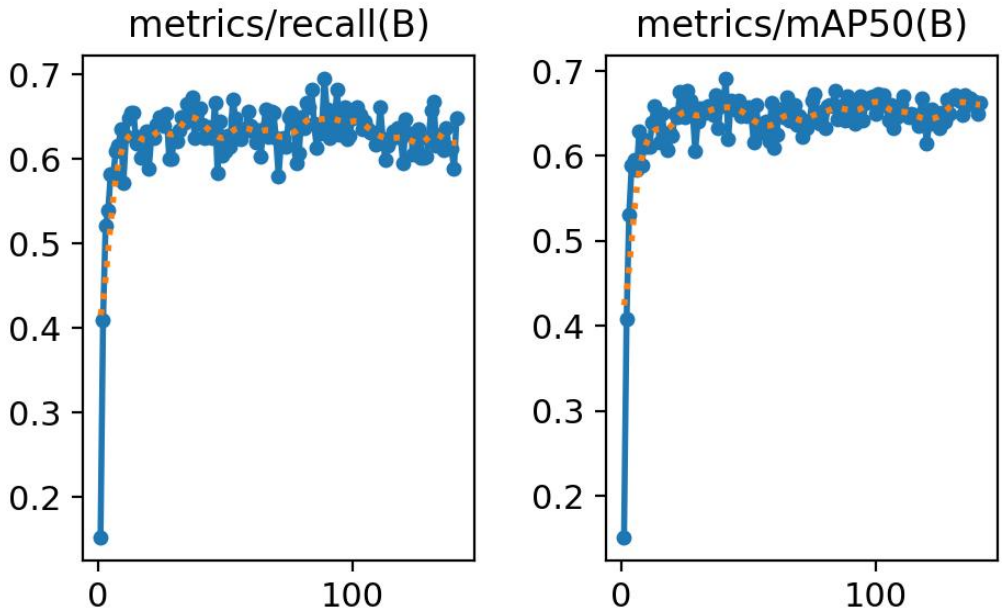

Figure 43: Recall and mean average precision over the training process on GC10-DET

After 5.3 hours of training, the process terminates at epoch 141 as no improvement is observed over the last 100 epochs. Similar situation with NEU-DET training, the model exhibits rapid convergence during the initial epochs despite a 0.001 learning rate.

Defect types	**Instances**	**Precision**	**Recall**	**mAP50**	**mAP50-95**
Punching hole	42	0.933	0.976	0.984	0.554
Welding line	71	0.841	0.819	0.85	0.397
Crescent gap	49	0.918	0.98	0.932	0.588
Water spot	50	0.786	0.84	0.908	0.494
Oil spot	84	0.515	0.571	0.578	0.219
Silk spot	136	0.595	0.562	0.528	0.202
Inclusion	52	0.38	0.404	0.320	0.112
Rolled pit	9	0.727	0.222	0.318	0.205
Crease	6	0.565	0.5	0.529	0.12
Waist folding	26	0.652	0.649	0.655	0.314
All class	**525**	**0.691**	**0.652**	**0.660**	**0.321**
FPS	**64.52**				
F1 score	**0.66**				

Table 10: Performance on GC10-DET defect detection

The trained model is evaluated on a testing set with 345 images. Table 10 shows that the model performs generally well in defect detection, but it tends to struggle in detecting inclusion, oil spot, silk spot, crease and rolled pit. The model's ability to detect rolled pits and creases is constrained due to the extremely small number of instances in these classes, leading to insufficient training.

4.3.2 Visualisation of results

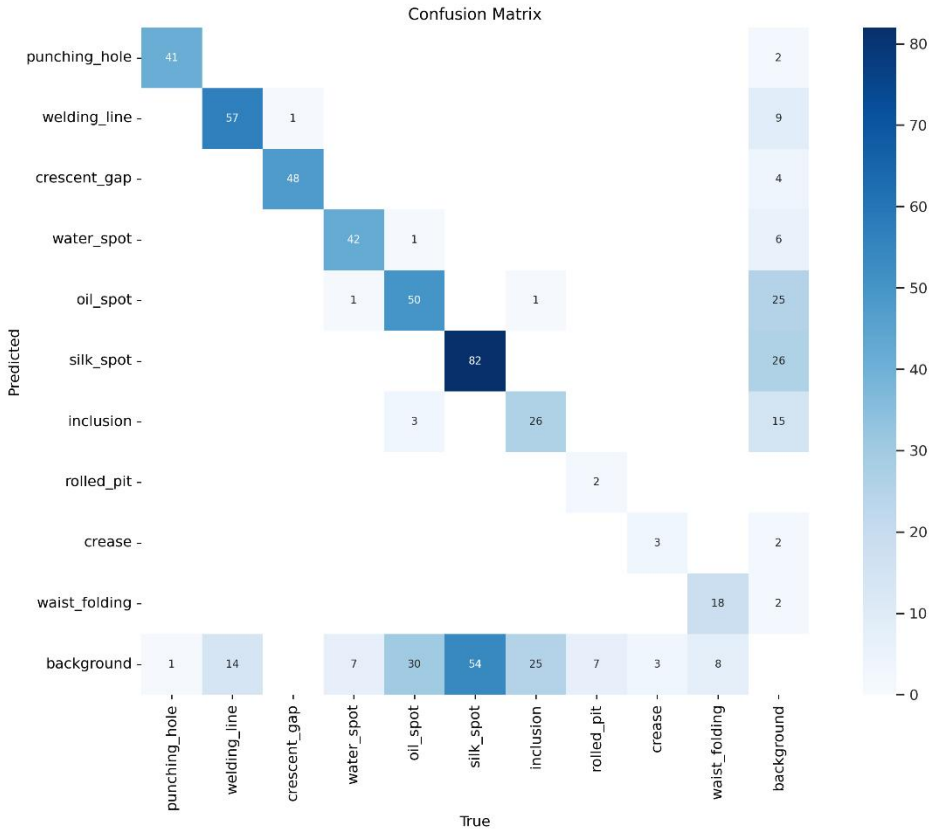

Figure 44: Confusion matrix for GC10-DET

As a summary of the confusion matrix, the model correctly detected 41 instances of punching holes, 57 instances of welding lines, 48 instances of crescent gaps, 42 instances of water spots, 50 instances of oil spots, 82 instances of silk spots, 26 instances of inclusion, 2 instances of rolled pit, 3 instances of crease, and 18 instances of waist folding. The model performs generally well in distinguishing defect classes from other defect classes, only few misdetections observed between defect classes.

However, 1 instance of punching hole, 14 instances of welding line, 7 instances of water spot, 30 instances of oil spot, 54 instances of silk spot 25 instances of inclusion, 7 instances of rolled pit, 3 instances of crease, 8 instances of waist folding are mis-detected as background. Also, a total of 91 instances of background are mis-detected as a defect class. This implies the poor performance of the model in distinguishing defects from the background.

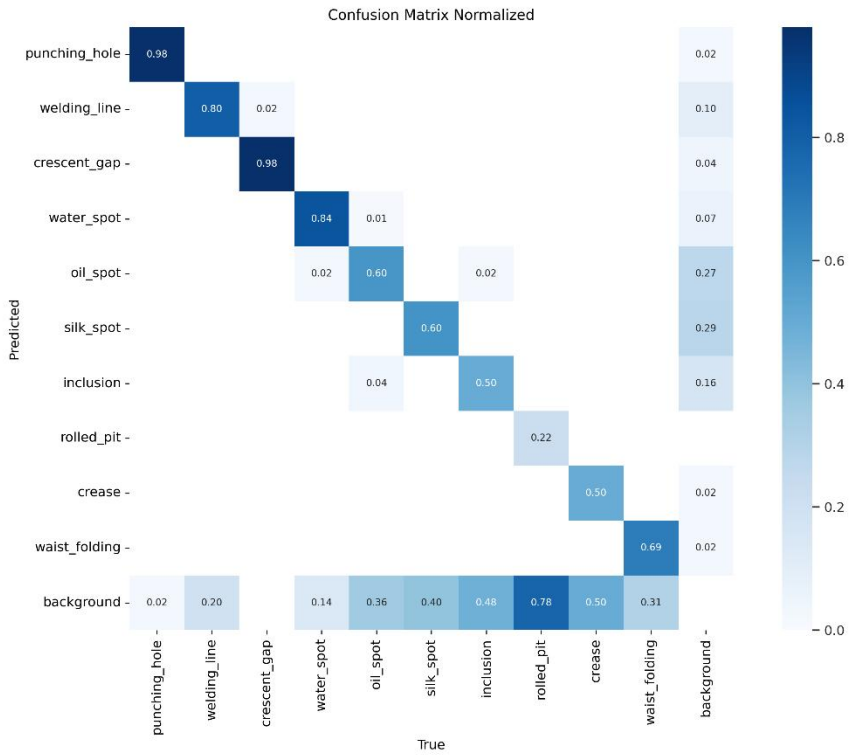

Figure 45: Normalised confusion matrix for GC10-DET

From the normalised confusion matrix, we can infer that the model is reliable in detecting punching holes, welding lines, crescent gaps, and water spots. The model achieves over 80% accuracy in detecting those classes. Other defect classes are often mis-detected as background, resulting in high detection errors.

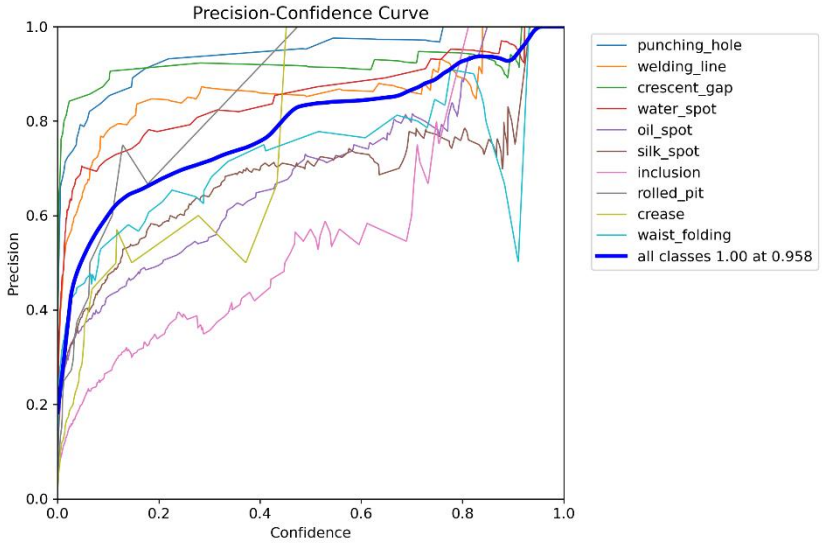

Figure 46: Precision-confidence curve for GC10-DET

The thick blue line plots the aggregated precision of all defect classes across different confidence thresholds. Classes like punching holes, welding lines, crescent gaps, and water spots show high precision across most confidence thresholds, indicating the model is reliable for detecting these defects correctly. Other defect classes require higher confidence thresholds to achieve better precision.

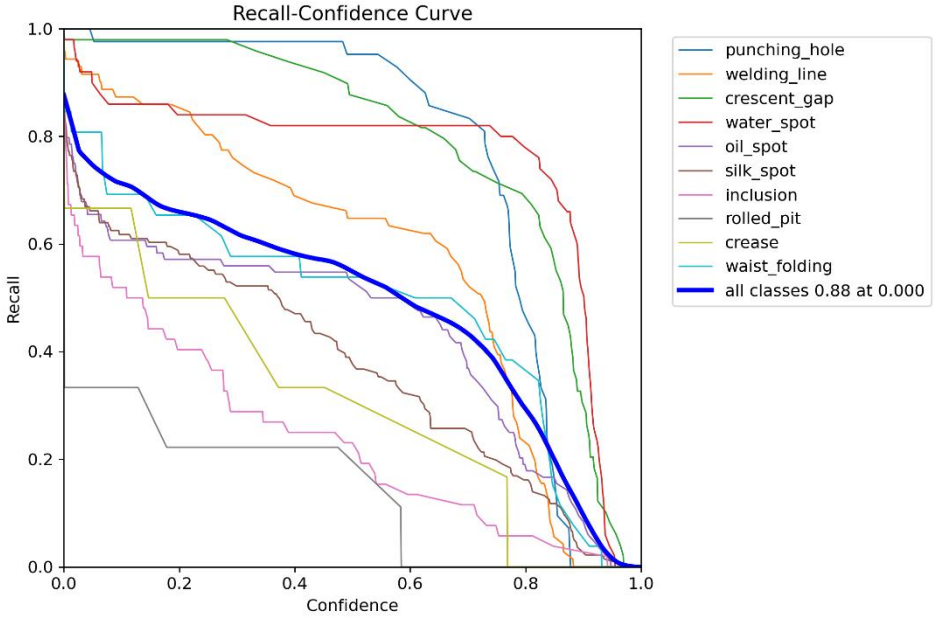

Figure 47: Recall-confidence curve for GC10-DET

Classes like punching hole, crescent gap, water spot, and welding line maintain high recall even when confidence thresholds increase. This indicates that the model is more sensitive in capturing these defects even when the model makes fewer predictions. Other defect classes are likely to be missed by the model.

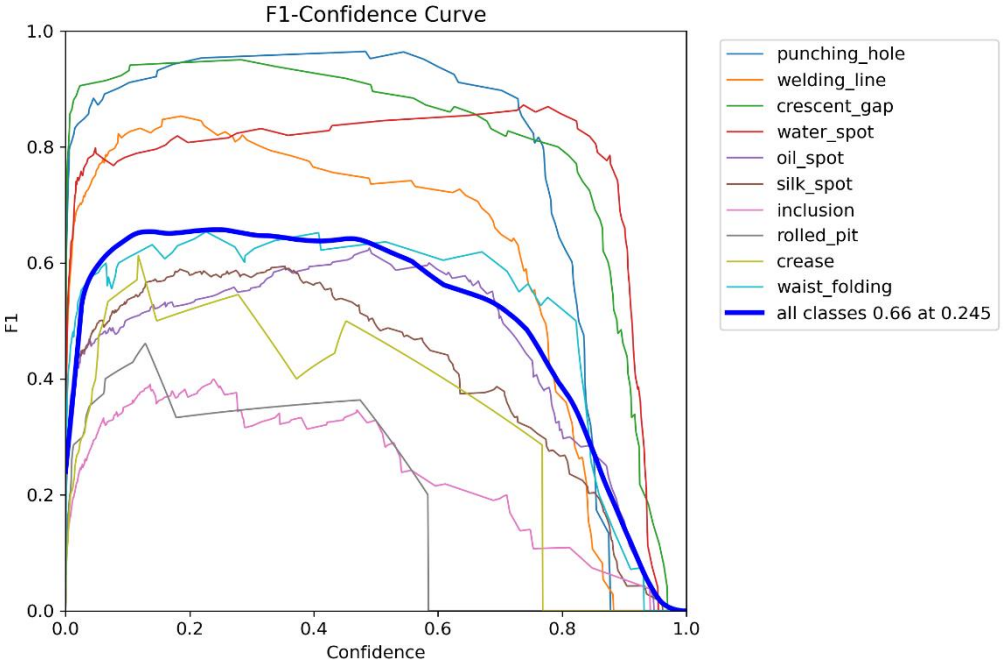

Figure 48: F1-confidence curve for GC10-DET

The peak F1 score of 0.74 occurs at a 0.321 confidence threshold, indicating that the model has the best overall performance on precision and recall across all defect classes at this confidence threshold.

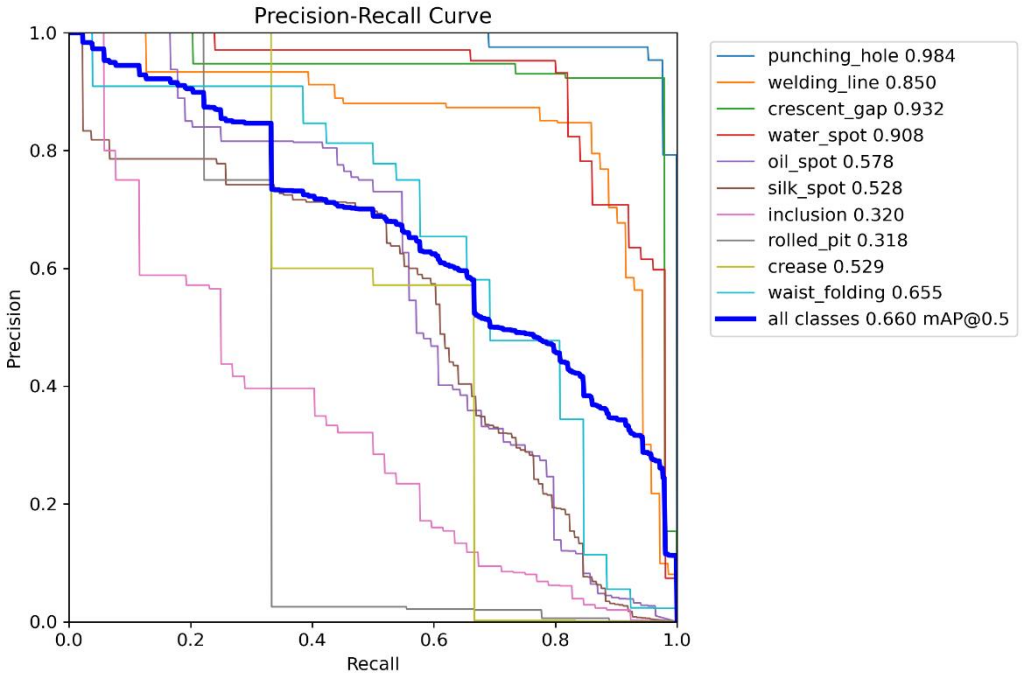

Figure 49: Precision recall curve for GC10-DET

The graph of the precision-recall curve shows that the model is more accurate and sensitive at detecting defect classes like punching hole, welding line, crescent gap and water spot as their curves are closer to the top-right corner. However, the model is relatively poor in detecting the other 6 defect classes. A mean average precision of 0.66 is achieved at 50% IoU thresholds.

4.3.3 Comparison between actual labels and model predictions

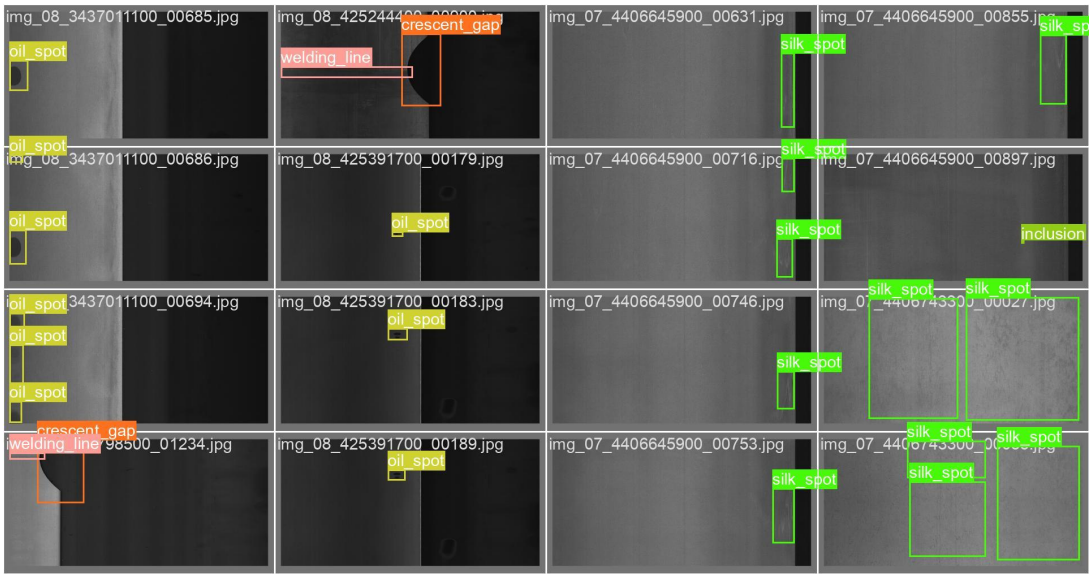

Figure 50: Defect labels on testing batch 0

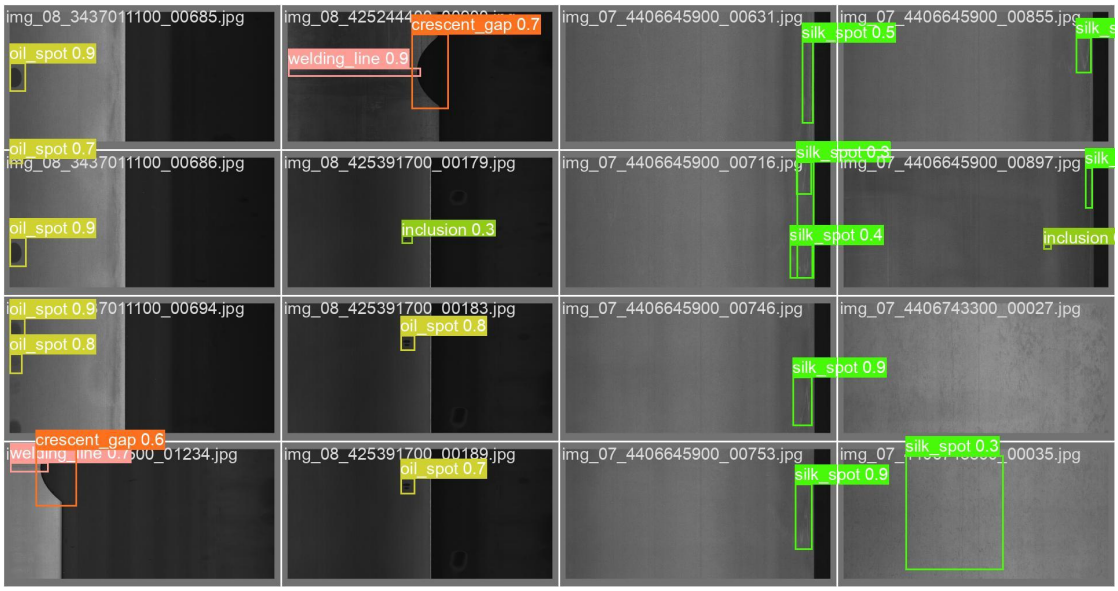

Figure 51: Model prediction on testing batch 0

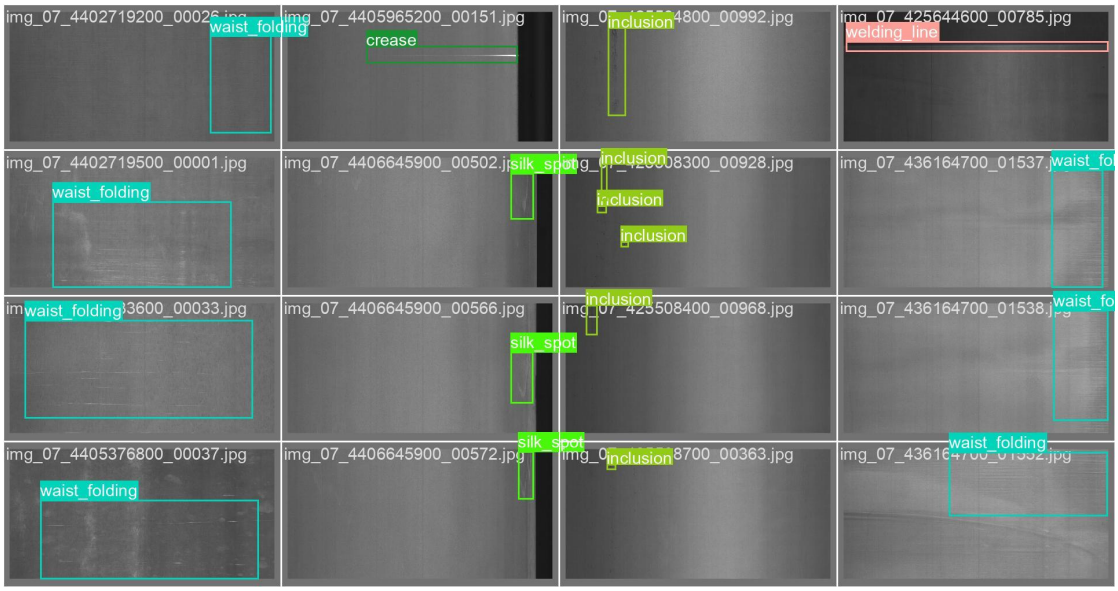

Figure 52: Defect labels on testing batch 1

Figure 53: Model prediction on testing batch 1

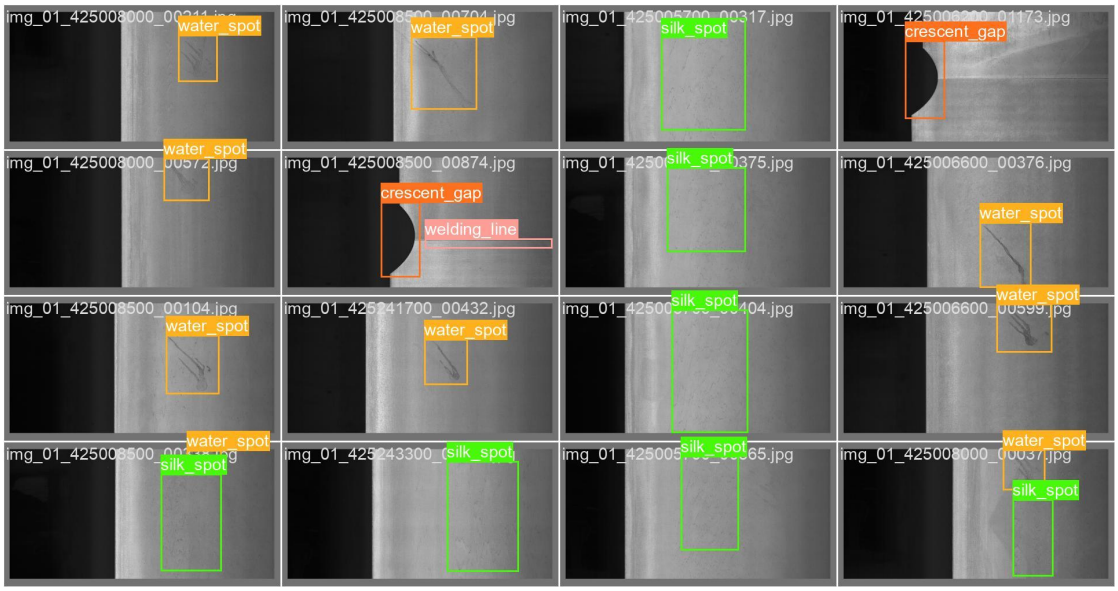

Figure 54: Defect labels on testing batch 2

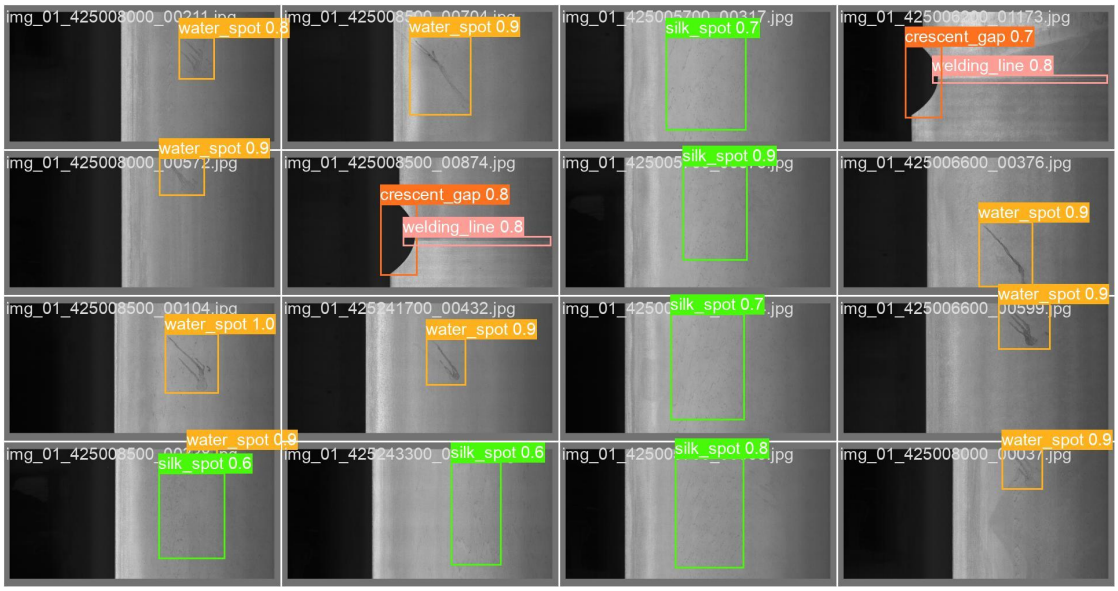

Figure 55: Model prediction on testing batch 2

The comparison shows that the model successfully detected most defects. Minor misdetections were observed on the silk spot, welding line, and waist folding. Major misdetections were observed on inclusion.

CHAPTER 5: DISCUSSION

5.1 Analysis of Poor Detection

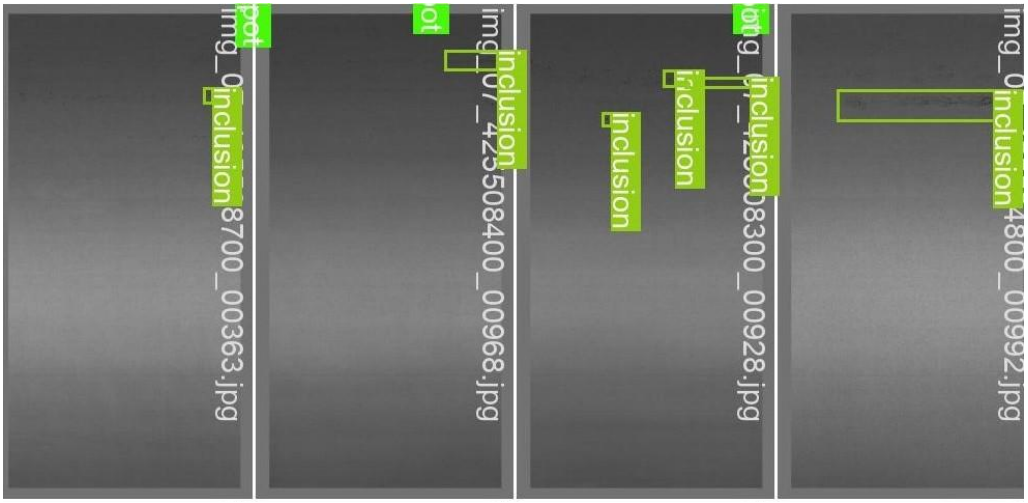

Figure 56: Ground truth labels for inclusion on the GC10-DET dataset

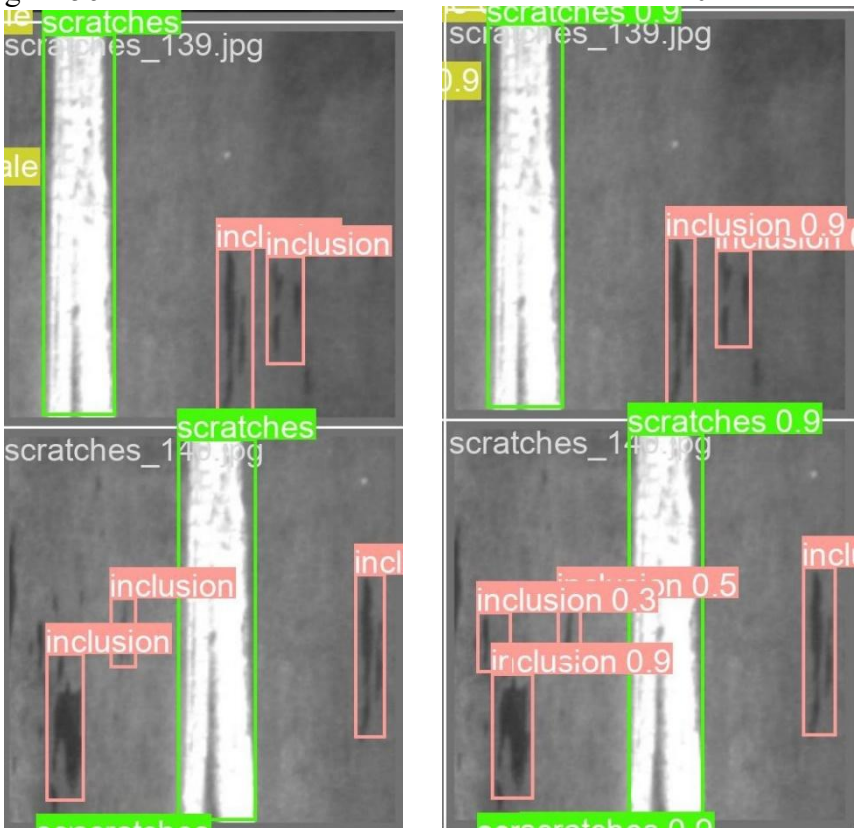

Figure 57: Ground truth labels (left) vs model prediction (right) on the NEU-DET dataset

The result on GC10-DET defect detection shows that the model struggles in detecting inclusion and rolled pit. Figure 56 shows that the inclusion defect is indistinguishable from the background on the GC10-DET dataset. Conversely, the model accurately detects inclusion defects on the NEU-DET dataset. Hence, the ground truth label of NEU-DET's inclusion is analysed and it is found that increasing contrast might help improve inclusion detection on GC10-DET. Poor detection on

rolled pits is due to the extremely small number of instances, leading to insufficient training.

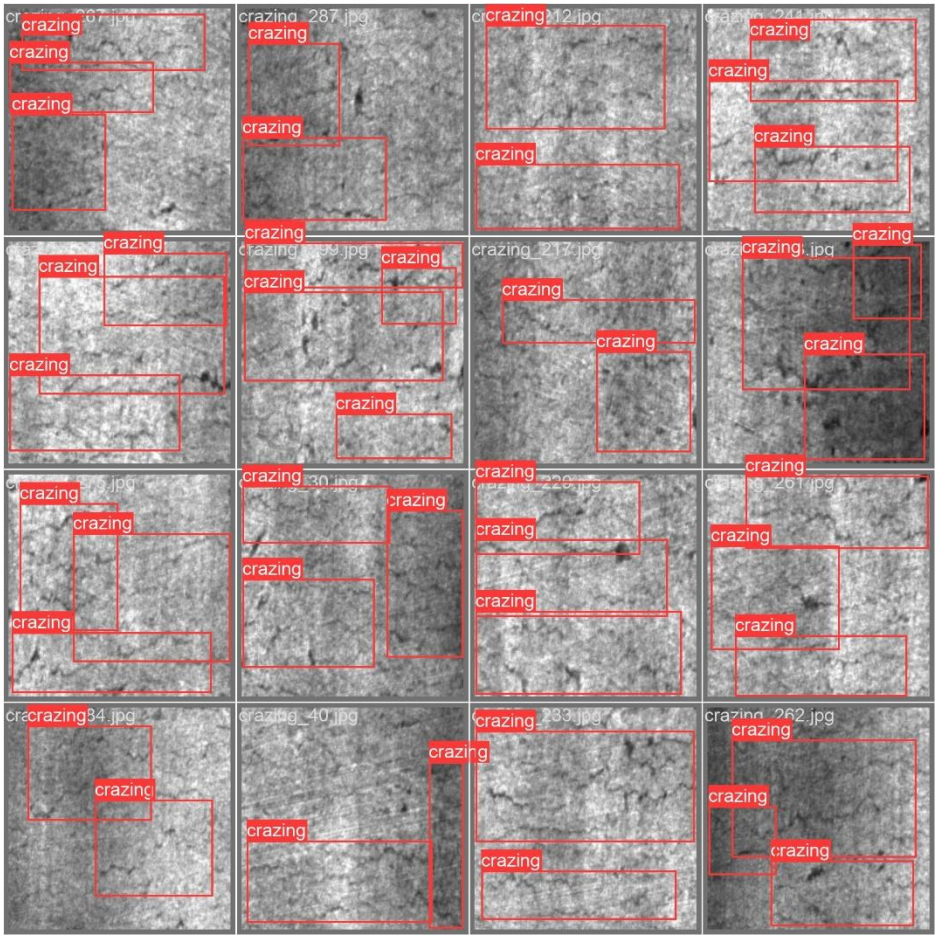

Figure 58: Overlapping ground truth labels for crazing defect

The result on NEU-DET defect detection shows that the model struggles in detecting crazing, resulting in very low mAP and recall. This might be due to the presence of overlapping ground truth labels, which impacts the model training process.

5.2 Method comparison on NEU-DET dataset

The performance of the proposed YOLOv10 model is compared with several state-of-the-art model on NEU-DET defect detection. There are single-shot multi-box detector (SSD) based model (Lv et al., 2020), faster R-CNN-based model (Lv et al., 2020), improved faster R-CNN based model (Duan et al., 2022), RetinaNet-based model (Cheng and Yu, 2020), and improved YOLOv7 based model (Wang et al., 2022). The highest metrics are bolded. '-' indicates metrics not mentioned in the paper.

Defect types	**SSD**	**Faster R-CNN**	**Improved Faster R-CNN**	**RetinaNet**	**Improved YOLOv7**	**Proposed YOLOv10**
Crazing	0.417	0.374	0.571	0.609	**0.690**	0.457
Inclusion	0.763	0.794	0.768	0.825	0.853	**0.854**
Patches	0.863	0.853	0.894	**0.943**	0.907	0.928
Pitted surface	0.851	0.815	0.785	**0.958**	0.843	0.875
Rolled in scale	0.581	0.545	0.683	0.672	**0.727**	0.653
Scratches	0.856	0.882	**0.942**	0.741	0.896	0.897
mAP	0.724	0.711	0.774	0.791	**0.819**	0.777
FPS	37.04	27.03	-	12.20	**55.56**	32.89

Table 11: mAP and FPS comparison between different methods on NEU-DET

Defect types	**SSD**	**Faster R-CNN**	**Improved Faster R-CNN**	**RetinaNet**	**Improved YOLOv7**	**Proposed YOLOv10**
Crazing	**0.965**	0.874	-	-	-	0.374
Inclusion	**0.974**	0.923	-	-	-	0.817
Patches	**0.987**	0.981	-	-	-	0.864
Pitted surface	**1.000**	0.943	-	-	-	0.841
Rolled in scale	**0.966**	0.881	-	-	-	0.593
Scratches	**0.981**	0.971	-	-	-	0.829
Average recall	**0.978**	0.929	-	-	0.765	0.72

Table 12: Recall the comparison between different methods on NEU-DET

The YOLOv10-based model outperforms SSD by 5.3% mAP, faster R-CNN by 6.6% mAP, and improved faster R-CNN by 0.3% mAP. Besides that, the processing speed of YOLOv10 outperforms faster R-CNN by 5.86FPS, and RetinaNet by 20.69FPS. The YOLOv10 is 1.4% mAP lower than RetinaNet but it outperforms in processing speed by 20.69FPS. Only improved YOLOv7 outperforms the proposed YOLOv10 in mAP, FPS and average recall. However, the recall of YOLOv10 is relatively low compared to SSD and faster R-CNN.

5.3 Method comparison on GC10-DET dataset

Performance of the proposed YOLOv10 model is compared with related works on GC10-DET defect detection, which includes single-shot multi-box detector (SSD) based model (Lv et al., 2020), faster R-CNN based model (Lv et al., 2020), improved YOLOv5 based model (Zhou et al., 2023), and improved YOLOv7 based model (Wang et al., 2022). The highest metrics are bolded. '-' indicates the metric is not mentioned in the paper.

Types	**SSD**	**Faster R-CNN**	**Improved YOLOv5**	**Improved YOLOv7**	**Proposed YOLOv10**
Punching hole	0.900	0.899	0.957	0.969	**0.984**
Welding line	0.885	0.554	**0.958**	0.798	0.85
Crescent gap	0.848	0.872	0.989	**0.991**	0.932
Water spot	0.558	0.599	0.832	**0.908**	**0.908**
Oil spot	0.622	0.653	**0.901**	0.885	0.578
Silk spot	0.650	0.579	0.901	**0.91**	0.528
Inclusion	0.256	0.194	0.663	**0.869**	0.32
Rolled pit	0.364	0.364	0.589	**0.826**	0.318
Crease	0.521	0.736	0.624	**0.868**	0.529
Waist folding	**0.919**	0.818	0.870	0	0.655
mAP	0.651	0.627	**0.828**	0.802	0.66
FPS	30.30	23.26	79.4	**105.20**	64.52

Table 13: mAP and FPS comparison between different methods on GC10-DET

Types	**SSD**	**Faster R-CNN**	**Improved YOLOv5**	**Improved YOLOv7**	**Proposed YOLOv10**
Punching hole	0.965	0.964	-	-	**0.976**
Welding line	**0.967**	0.623	-	-	0.819
Crescent gap	0.969	0.968	-	-	**0.98**
Water spot	0.739	0.696	-	-	**0.84**
Oil spot	**0.891**	0.761	-	-	0.571
Silk spot	**0.988**	0.708	-	-	0.562
Inclusion	**0.667**	0.551	-	-	0.404
Rolled pit	**0.333**	**0.333**	-	-	0.222
Crease	0.857	**1.000**	-	-	0.5
Waist folding	**1.000**	0.800	-	-	0.649
average recall	**0.838**	0.74	**-**	0.703	0.652

Table 14: Recall comparison between different methods on GC10-DET

The YOLOv10-based defect detection model outperforms SSD by 0.9% mAP and 34.22FPS, faster R-CNN by 3.3% mAP at 64.52FPS. However, the recall of YOLOv10 is still relatively poor compared to SSD and faster R-CNN. It seems likely that the limitation of the YOLO series as improved YOLOv7 also demonstrates poor recall.

Surprisingly, the improved YOLOv5 outperforms its successor like the improved YOLOv7 and YOLOv10 in mAP. The analysis is conducted on improved YOLOv5, and it is found that the author augments the data first and splits them into training, validation, and testing sets. This causes data leakage problems where the model saw similar images during training already. The correct step should be to split the data into training, validation, and testing sets first, then perform data augmentation in training data.

5.4 Discussion of Findings

The method comparison in NEU-DET and GC10-DET shows that the improved YOLOv7 have the best overall performance in mAP and FPS, but produce low recall compared to SSD and faster R-CNN. The proposed YOLOv10 also demonstrates competitive performance, outperforming SSD and faster R-CNN in mAP and processing speed. However, the YOLOv10-based model struggles to distinguish tiny defects like rolled pit and inclusion from the background, resulting in lower recall and mAP on these defects.

As the original YOLOv10 framework was initially designed for object detection, the defect detection performance might be constrained. To surpass the improved YOLOv7, future research should focus on modifying the YOLOv10 framework specifically for defect detection, aiming at improving the recall and mAP. Additionally, a more extensive and diverse dataset of metal surface defects is essential to improve model robustness and generalisation, as the reliance on transfer learning may limit performance due to inherent differences between the pre-trained tasks and the target application.

CHAPTER 6: CONCLUSION

6.1 Summary

The latest YOLOv10 demonstrates exceptional performance on COCO object detection, outperforming state-of-the-art object detectors in average precision with fewer parameters and faster latency speed. However, there is a lack of empirical study of YOLOv10 on metal surface defect detection. To address this gap, this study explores a YOLOv10-based model for metal surface defect detection through transfer learning. The YOLOv10 for COCO object detection is adapted to metal surface detection. The pre-trained YOLOv10 on COCO object detection is trained on two benchmark metal surface defect datasets, 'NEU-DET' and 'GC10-DET'. After training, the model is evaluated and compared with the state-of-the-art metal surface defect detection model to verify its effectiveness. As the YOLOv10-based metal surface defect detection model is successfully built, trained, tested and verified, we can conclude that all objectives are achieved.

6.2 Research contributions

This study contributes a novel YOLOv10-based model for metal surface detection. The performance comparison shows that YOLOv10 based model demonstrates competitive performance, outperforming state-of-the-art defect detection models such as SSD, faster R-CNN, improved faster R-CNN and RetinaNet in both mean Average Precision (mAP) and frame per second (FPS). Additionally, this study provides a comprehensive review of defect detection methodologies and object detection frameworks, includes the evolution of the YOLO series from v1 to v10, SSD, deconvolutional SSD, and R-CNN families.

6.3 Future research and development

The experiment shows that the YOLO series such as YOLOv7 and YOLOv10 exhibits lower recall on defect detection compared to other methods. As the original YOLOv10 framework was initially designed for object detection, the defect detection performance might be constrained. To enhance mAP and recall of defect detection, future research should focus on modifying the YOLOv10 framework specifically for defect detection tasks. Furthermore, the development of a more extensive and diverse dataset of metal surface defects is essential to improve model robustness and generalisation.

6.4 Personal reflections

The dissertation journey has been both challenging and rewarding. The exploration of cutting-edge object detector and their application to metal surface defect detection has deepened my understanding of machine learning and object detection. The process required continuous learning, problem-solving, and adaptation, which has contributed significantly to my personal and professional growth. Moving forward, addressing the limitations identified in this research will be crucial for further advancements, hopefully leading to impactful outcomes in the defect detection field.

REFERENCE

[1] Fu, Y., Downey, A.R., Yuan, L., Zhang, T., Pratt, A. and Balogun, Y., 2022. Machine learning algorithms for defect detection in metal laser-based additive manufacturing: A review. *Journal of Manufacturing Processes*, *75*, pp.693-710.

[2] Scime, L. and Beuth, J., 2018. A multi-scale convolutional neural network for autonomous anomaly detection and classification in a laser powder bed fusion additive manufacturing process. *Additive Manufacturing*, *24*, pp.273-286.

[3] Goh, G.D., Sing, S.L. and Yeong, W.Y., 2021. A review on machine learning in 3D printing: applications, potential, and challenges. *Artificial Intelligence Review*, *54*(1), pp.63-94.

[4] Li, Z., Liu, F., Yang, W., Peng, S. and Zhou, J., 2021. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, *33*(12), pp.6999-7019.

[5] Wang, R. and Cheung, C.F., 2022. CenterNet-based defect detection for additive manufacturing. *Expert Systems with Applications*, *188*, p.116000.

[6] Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q. and Tian, Q., 2019. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 6569-6578).

[7] Newell, A., Yang, K. and Deng, J., 2016. Stacked hourglass networks for human pose estimation. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14* (pp. 483-499). Springer International Publishing.

[8] Westphal, E. and Seitz, H., 2021. A machine learning method for defect detection and visualization in selective laser sintering based on convolutional neural networks. *Additive Manufacturing*, *41*, p.101965.

[9] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

[10] Given, L.M. ed., 2008. *The Sage encyclopedia of qualitative research methods*. Sage publications.

[11] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[12] Wong, V.W.H., Ferguson, M., Law, K.H., Lee, Y.T.T. and Witherell, P., 2022. Segmentation of additive manufacturing defects using U-net. *Journal of Computing and Information Science in Engineering*, *22*(3), p.031005.

[13] Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1251-1258).

[14] Zeng, N., Wu, P., Wang, Z., Li, H., Liu, W. and Liu, X., 2022. A small-sized object detection oriented multi-scale feature fusion approach with application to defect detection. *IEEE Transactions on Instrumentation and Measurement*, *71*, pp.1-14.

[15] Huang, H., Tang, X., Wen, F. and Jin, X., 2022. Small object detection method with shallow feature fusion network for chip surface defect detection. *Scientific reports*, *12*(1), p.3914.

[16] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[17] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

[18] Redmon, J. and Farhadi, A., 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

[19] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B. and Belongie, S., 2017. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2117-2125).

[20] Liu, S., Qi, L., Qin, H., Shi, J. and Jia, J., 2018. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8759-8768).

[21] Zhu, L., Deng, Z., Hu, X., Fu, C.W., Xu, X., Qin, J. and Heng, P.A., 2018. Bidirectional feature pyramid network with recurrent attention residual modules for shadow detection. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 121-136)

[22] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., 2016. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14* (pp. 21-37). Springer International Publishing.

[23] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

[24] Law, H. and Deng, J., 2018. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 734-750).

[25] Cheng, X. and Yu, J., 2020. RetinaNet with difference channel attention and adaptively spatial feature fusion for steel surface defect detection. *IEEE Transactions on Instrumentation and Measurement*, *70*, pp.1-11.

[26] Girshick, R., Donahue, J., Darrell, T. and Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

[27] Girshick, R., 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 1440-1448).

[28] Ren, S., He, K., Girshick, R. and Sun, J., 2016. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, *39*(6), pp.1137-1149.

[29] He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 2961-2969).

[30] Bodla, N., Singh, B., Chellappa, R. and Davis, L.S., 2017. Soft-NMS--improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision* (pp. 5561-5569).

[31] Neubeck, A. and Van Gool, L., 2006, August. Efficient non-maximum suppression. In *18th international conference on pattern recognition (ICPR'06)* (Vol. 3, pp. 850-855). IEEE.

[32] Ning, C., Zhou, H., Song, Y. and Tang, J., 2017, July. Inception single shot multibox detector for object detection. In *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)* (pp. 549-554). IEEE.

[33] Hussain, M., 2023. YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, *11*(7), p.677.

[34] Fu, C.Y., Liu, W., Ranga, A., Tyagi, A. and Berg, A.C., 2017. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*.

[35] Ronneberger, O., Fischer, P. and Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18* (pp. 234-241). Springer International Publishing.

[36] Yang, J., Li, S., Wang, Z. and Yang, G., 2019. Real-time tiny part defect detection system in manufacturing using deep learning. *IEEe Access*, *7*, pp.89278-89291.

[37] Yu, H., Li, X., Feng, Y. and Han, S., 2023. Multiple attentional path aggregation network for marine object detection. *Applied intelligence*, *53*(2), pp.2434-2451.

[38] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. *Advances in neural information processing systems*, *30*.

[39] Lv, X., Duan, F., Jiang, J.J., Fu, X. and Gan, L., 2020. Deep metallic surface defect detection: The new benchmark and detection network. *Sensors*, *20*(6), p.1562.

[40] Zhou, C., Lu, Z., Lv, Z., Meng, M., Tan, Y., Xia, K., Liu, K. and Zuo, H., 2023. Metal surface defect detection based on improved YOLOv5. *Scientific Reports*, *13*(1), p.20803.

[41] Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J. and Ding, G., 2024. Yolov10: Real-time end-to-end object detection. *arXiv preprint arXiv:2405.14458*.

[42] Song, K. and Yan, Y., 2013. A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects. *Applied Surface Science*, *285*, pp.858-864.

[43] Ultralytics (2024) *Yolov10*, *YOLOv10 - Ultralytics YOLO Docs*. Available at: https://docs.ultralytics.com/models/yolov10/ (Accessed: 09 August 2024).

[44] Tan, L., Liu, S., Gao, J., Liu, X., Chu, L. and Jiang, H., 2024. Enhanced Self-Checkout System for Retail Based on Improved YOLOv10. *arXiv preprint arXiv:2407.21308*.

[45] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L., 2014. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13* (pp. 740-755). Springer International Publishing.

[46] Wang, C.Y., Liao, H.Y.M., Wu, Y.H., Chen, P.Y., Hsieh, J.W. and Yeh, I.H., 2020. CSPNet: A new backbone that can enhance learning capability of CNN. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops* (pp. 390-391).

[47] Li, Z., Tian, X., Liu, X., Liu, Y. and Shi, X., 2022. A two-stage industrial defect detection framework based on improved-yolov5 and optimized-inception-resnetv2 models. *Applied Sciences*, *12*(2), p.834.

[48] Bhatt, P.M., Malhan, R.K., Rajendran, P., Shah, B.C., Thakar, S., Yoon, Y.J. and Gupta, S.K., 2021. Image-based surface defect detection using deep learning: A review. *Journal of Computing and Information Science in Engineering*, *21*(4), p.040801.

[49] Hui, J. (2018). *SSD object detection: Single Shot MultiBox Detector for real-time processing*. [online] Medium. Available at: https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06.

[50] Sapkota, R., Qureshi, R., Calero, M.F., Hussain, M., Badjugar, C., Nepal, U., Poulose, A., Zeno, P., Vaddevolu, U.B.P., Yan, H. and Karkee, M., 2024. Yolov10 to its genesis: A decadal and comprehensive review of the you only look once series. *arXiv preprint arXiv:2406.19407*.

[51] Zhang, Y., Li, X., Wang, F., Wei, B. and Li, L., 2021, August. A comprehensive review of one-stage networks for object detection. In *2021 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)* (pp. 1-6). IEEE.

[52] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. and Berg, A.C., 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*, pp.211-252.

[53] Redmon, J. and Farhadi, A., 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7263-7271).

[54] Terven, J., Córdova-Esparza, D.M. and Romero-González, J.A., 2023. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, *5*(4), pp.1680-1716.

[55] Bochkovskiy, A., Wang, C.Y. and Liao, H.Y.M., 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

[56] Misra, D., 2019. Mish: A self-regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*.

[57] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, *37*(9), pp.1904-1916.

[58] Woo, S., Park, J., Lee, J.Y. and Kweon, I.S., 2018. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 3-19).

[59] Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R. and Ren, D., 2020, April. Distance-IoU loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 34, No. 07, pp. 12993-13000).

[60] Ghiasi, G., Lin, T.Y. and Le, Q.V., 2018. Dropblock: A regularization method for convolutional networks. *Advances in neural information processing systems*, *31*.

[61] Loshchilov, I. and Hutter, F., 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

[62] Jocher, G. (2020). *ultralytics/yolov5*. [online] GitHub. Available at: https://github.com/ultralytics/yolov5.

[63] Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W. and Li, Y., 2022. YOLOv6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*

[64] Ding, X., Zhang, X., Ma, N., Han, J., Ding, G. and Sun, J., 2021. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 13733-13742).

[65] Ding, X., Chen, H., Zhang, X., Huang, K., Han, J. and Ding, G., 2022. Re-parameterizing your optimizers rather than architectures. *arXiv preprint arXiv:2205.15242*.

[66] Shu, C., Liu, Y., Gao, J., Yan, Z. and Shen, C., 2021. Channel-wise knowledge distillation for dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 5311-5320).

[67] Zhang, H., Wang, Y., Dayoub, F. and Sunderhauf, N., 2021. Varifocalnet: An iou-aware dense object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 8514-8523).

[68] Gevorgyan, Z., 2022. SIoU loss: More powerful learning for bounding box regression. *arXiv preprint arXiv:2205.12740*.

[69] Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I. and Savarese, S., 2019. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 658-666).

[70] Feng, C., Zhong, Y., Gao, Y., Scott, M.R. and Huang, W., 2021, October. Tood: Task-aligned one-stage object detection. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 3490-3499). IEEE Computer Society.

[71] Wang, C.Y., Bochkovskiy, A. and Liao, H.Y.M., 2023. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 7464-7475).

[72] Wang, C.Y., Liao, H.Y.M. and Yeh, I.H., 2022. Designing network design strategies through gradient path analysis. *arXiv preprint arXiv:2211.04800*.

[73] Wang, C.Y., Yeh, I.H. and Liao, H.Y.M., 2021. You only learn one representation: Unified network for multiple tasks. *arXiv preprint arXiv:2105.04206*.

[74] Jocher, G., Chaurasia, A. and Qiu, J. (2023). *YOLOv8 by Ultralytics*. [online] GitHub. Available at: https://github.com/ultralytics/ultralytics.

[75] Li, X., Wang, W., Wu, L., Chen, S., Hu, X., Li, J., Tang, J. and Yang, J., 2020. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. *Advances in Neural Information Processing Systems*, *33*, pp.21002-21012.

[76] Wang, C.Y., Yeh, I.H. and Liao, H.Y.M., 2024. Yolov9: Learning what you want to learn using programmable gradient information. *arXiv preprint arXiv:2402.13616*.

[77] Howard, A.G., 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[78] Hmidani, O. and Alaoui, E.I., 2022, December. A comprehensive survey of the R-CNN family for object detection. In *2022 5th International Conference on Advanced Communication Technologies and Networking (CommNet)* (pp. 1-6). IEEE.

[79] Uijlings, J.R., Van De Sande, K.E., Gevers, T. and Smeulders, A.W., 2013. Selective search for object recognition. *International journal of computer vision*, *104*, pp.154-171.

[80] Zhao, Z.Q., Zheng, P., Xu, S.T. and Wu, X., 2019. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, *30*(11), pp.3212-3232.

[81] Zheng, X., Zheng, S., Kong, Y. and Chen, J., 2021. Recent advances in surface defect inspection of industrial products using deep learning techniques. *The International Journal of Advanced Manufacturing Technology*, *113*, pp.35-58.

[82] Qi, S., Yang, J. and Zhong, Z., 2020, September. A review on industrial surface defect detection based on deep learning technology. In *Proceedings of the 2020 3rd International Conference on Machine Learning and Machine Intelligence* (pp. 24-30).

[83] Duan, H., Huang, J., Liu, W. and Shu, F., 2022, August. Defective surface detection based on improved faster R-CNN. In *2022 IEEE International Conference on Industrial Technology (ICIT)* (pp. 1-6). IEEE.

[84] Wang, Y., Wang, H. and Xin, Z., 2022. Efficient detection model of steel strip surface defects based on YOLO-V7. *Ieee Access*, *10*, pp.133936-133944.

[85] Wang, Q., Wu, B., Zhu, P., Li, P., Zuo, W. and Hu, Q., 2020. ECA-Net: Efficient channel attention for deep convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 11534-11542).

[86] He, Y., Song, K., Meng, Q. and Yan, Y., 2019. An end-to-end steel surface defect detection approach via fusing multiple hierarchical features. *IEEE transactions on instrumentation and measurement*, *69*(4), pp.1493-1504.

APPENDIX A: ETHICAL APPROVAL

College of Engineering, Design and Physical Sciences Research Ethics Committee
Brunel University London
Kingston Lane
Uxbridge
UB8 3PH
United Kingdom

www.brunel.ac.uk

6 August 2024

LETTER OF CONFIRMATION

Applicant: Mr Meng Heng Chong

Project Title: A deep learning approach for defect detection

Reference: 49394-NER-Jul/2024- 52273-1

Dear Mr Meng Heng Chong

The Research Ethics Committee has considered the above application recently submitted by you.

This letter is to confirm that, according to the information provided in your BREO application, your project does not require full ethical review. You may proceed with your research as set out in your submitted BREO application, using secondary data sources only. You may not use any data sources for which you have not sought approval.

Please note that:

- **You are not permitted to conduct research involving human participants, their tissue and/or their data. If you wish to conduct such research (including surveys, questionnaires, interviews etc.), you must contact the Research Ethics Committee to seek approval prior to engaging with any participants or working with data for which you do not have approval.**
- The Research Ethics Committee reserves the right to sample and review documentation relevant to the study.
- If during the course of the study, you would like to carry out research activities that concern a human participant, their tissue and/or their data, you must submit a new BREO application and await approval before proceeding. Research activity includes the recruitment of participants, undertaking consent procedures and collection of data. Breach of this requirement constitutes research misconduct and is a disciplinary offence.

Good luck with your research!

Kind regards,

Professor Simon Taylor

Chair of the College of Engineering, Design and Physical Sciences Research Ethics Committee

Brunel University London

APPENDIX B: CODE

Appendix B.1 Code For NEU-DET

```
#Mount to google drive
from google.colab import drive
drive.mount('/content/drive')

#Install required package and libraries
!pip install -q git+https://github.com/THU-MIG/yolov10.git
import os
import numpy as np
import cv2
from ultralytics import YOLOv10
import xml.etree.ElementTree as ET
import shutil
import random
import albumentations as A
from albumentations.pytorch import ToTensorV2
from IPython.display import Image
import yaml
HOME = os.getcwd()
print(HOME)

# Converting COCO annotation format to YOLO
# Define function for converting COCO annotation format to YOLO
def convert_voc_to_yolo(xml_file, txt_file, image_width, image_height):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    with open(txt_file, 'w') as f:
        for obj in root.findall('object'):
            class_name = obj.find('name').text
            class_id = class_name_to_id[class_name]  # Convert class name to class ID

            bndbox = obj.find('bndbox')
            xmin = int(bndbox.find('xmin').text)
            ymin = int(bndbox.find('ymin').text)
            xmax = int(bndbox.find('xmax').text)
            ymax = int(bndbox.find('ymax').text)

            # Convert from Pascal VOC format to YOLO format
            x_center = (xmin + xmax) / 2.0 / image_width
            y_center = (ymin + ymax) / 2.0 / image_height
            width = (xmax - xmin) / image_width
            height = (ymax - ymin) / image_height

            f.write(f"{class_id} {x_center:.6f} {y_center:.6f} {width:.6f} {height:.6f}\n")

#Funtion to create YOLO annotation file
def create_yolo_annotations(xml_folder, output_folder, image_folder):
```

```
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    for xml_file in os.listdir(xml_folder):
        if xml_file.endswith('.xml'):
            xml_path = os.path.join(xml_folder, xml_file)
            image_name = os.path.splitext(xml_file)[0] + '.jpg'
            image_path = os.path.join(image_folder, image_name)

            # Implement this function to get image size
            image_width, image_height = get_image_size(image_path)

            txt_file = os.path.join(output_folder, os.path.splitext(xml_file)[0] + '.txt')
            convert_voc_to_yolo(xml_path, txt_file, image_width, image_height)

def get_image_size(image_path):
    image = cv2.imread(image_path)
    return image.shape[1], image.shape[0]

# Define folder path for images, COCO annotation and YOLO annotation
xml_folder = '/content/drive/MyDrive/NEU-DET/annotation'
output_folder = '/content/drive/MyDrive/NEU-DET/YOLO_annotation'
image_folder = '/content/drive/MyDrive/NEU-DET/images'

# Mapping actual class names to corresponding IDs
class_name_to_id = {
    'crazing': 0,
    'inclusion': 1,
    'patches': 2,
    'pitted_surface': 3,
    'rolled-in_scale': 4,
    'scratches': 5

}
#Execute function
create_yolo_annotations(xml_folder, output_folder, image_folder)
```

#Split dataset into Train, test, and validation set.

```
#Define data splitting function
def split_data_with_annotations(source_dir, annotation_dir, train_dir, val_dir, test_dir,
                                train_anno_dir, val_anno_dir, test_anno_dir,
                                train_split=0.7, val_split=0.15, test_split=0.15):
    """
    Splits the image data and their corresponding annotations into training, validation, and
testing sets.

    Parameters:
    source_dir (str): The directory containing the original images.
    annotation_dir (str): The directory containing the annotations for the images.
    train_dir (str): The directory where the training images will be saved.
    val_dir (str): The directory where the validation images will be saved.
```

```
    test_dir (str): The directory where the test images will be saved.
    train_anno_dir (str): The directory where the training annotations will be saved.
    val_anno_dir (str): The directory where the validation annotations will be saved.
    test_anno_dir (str): The directory where the test annotations will be saved.
    train_split (float): The proportion of images to include in the training set.
    val_split (float): The proportion of images to include in the validation set.
    test_split (float): The proportion of images to include in the test set.
    """

    # Ensure the destination directories exist
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)
    os.makedirs(train_anno_dir, exist_ok=True)
    os.makedirs(val_anno_dir, exist_ok=True)
    os.makedirs(test_anno_dir, exist_ok=True)

    # Get all image file names from the source directory
    image_files = [f for f in os.listdir(source_dir) if os.path.isfile(os.path.join(source_dir, f))
and f.endswith('.jpg')]

    # Shuffle the image files
    random.shuffle(image_files)

    # Calculate the split indices
    train_idx = int(len(image_files) * train_split)
    val_idx = train_idx + int(len(image_files) * val_split)

    # Split the images
    train_files = image_files[:train_idx]
    val_files = image_files[train_idx:val_idx]
    test_files = image_files[val_idx:]

    # Copy the files and their corresponding annotations to their respective directories
    for file_name in train_files:
        shutil.copy(os.path.join(source_dir, file_name), os.path.join(train_dir, file_name))
        shutil.copy(os.path.join(annotation_dir, file_name.replace('.jpg', '.txt')),
                    os.path.join(train_anno_dir, file_name.replace('.jpg', '.txt')))

    for file_name in val_files:
        shutil.copy(os.path.join(source_dir, file_name), os.path.join(val_dir, file_name))
        shutil.copy(os.path.join(annotation_dir, file_name.replace('.jpg', '.txt')),
                    os.path.join(val_anno_dir, file_name.replace('.jpg', '.txt')))

    for file_name in test_files:
        shutil.copy(os.path.join(source_dir, file_name), os.path.join(test_dir, file_name))
        shutil.copy(os.path.join(annotation_dir, file_name.replace('.jpg', '.txt')),
                    os.path.join(test_anno_dir, file_name.replace('.jpg', '.txt')))

    print(f"Training set: {len(train_files)} images")
    print(f"Validation set: {len(val_files)} images")
    print(f"Test set: {len(test_files)} images")
```

```
#Define folder path
source_directory = '/content/drive/MyDrive/NEU-DET/images'
annotation_directory = '/content/drive/MyDrive/NEU-DET/YOLO_annotation'
train_directory = '/content/drive/MyDrive/NEU-DET/train/images'
train_annotation_directory = '/content/drive/MyDrive/NEU-DET/train/labels'
validation_directory = '/content/drive/MyDrive/NEU-DET/val/images'
validation_annotation_directory = '/content/drive/MyDrive/NEU-DET/val/labels'
test_directory = '/content/drive/MyDrive/NEU-DET/test/images'
test_annotation_directory = '/content/drive/MyDrive/NEU-DET/test/labels'

# Execute data splitting function
split_data_with_annotations(source_directory, annotation_directory,
                            train_directory, validation_directory, test_directory,
                            train_annotation_directory, validation_annotation_directory,
test_annotation_directory)
```

#Perform data augmentation on training set

```
# Define the augmentation pipeline
augmentation_pipeline = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    A.GaussianBlur(p=0.2),
    ToTensorV2()
], bbox_params=A.BboxParams(format='yolo', label_fields=['labels'], min_visibility=0.2))

# Function to augment image and bounding boxes in YOLO format
def augment_image_and_bboxes_yolo(image_path, bboxes, labels, images_output_dir,
annotations_output_dir, aug_pipeline, num_augmentations=2):
    image = cv2.imread(image_path)
    image_name = os.path.basename(image_path).split('.')[0]

    for i in range(num_augmentations):
        augmented = aug_pipeline(image=image, bboxes=bboxes, labels=labels)
        augmented_image = augmented["image"]
        augmented_bboxes = augmented["bboxes"]
        augmented_labels = augmented["labels"]

        # Convert the image to the correct format and save it
        augmented_image= augmented_image.permute(1, 2, 0).cpu().numpy()
        augmented_image_path = os.path.join(images_output_dir,
f"{image_name}_aug_{i}.jpg")
        cv2.imwrite(augmented_image_path, augmented_image)

        # Save augmented bounding boxes in YOLO format
        augmented_bboxes_path = os.path.join(annotations_output_dir,
f"{image_name}_aug_{i}.txt")
        with open(augmented_bboxes_path, 'w') as f:
```

```
            for bbox, label in zip(augmented_bboxes, augmented_labels):
                x_center, y_center, bbox_width, bbox_height = bbox
                f.write(f"{label} {x_center} {y_center} {bbox_width} {bbox_height}\n")

        print(f"Saved augmented image and bboxes: {augmented_image_path},
{augmented_bboxes_path}")

# Path where images and annotations are stored
images_dir = '/content/drive/MyDrive/NEU-DET/train/images'
annotations_dir = '/content/drive/MyDrive/NEU-DET/train/labels'
images_output_dir = '/content/drive/MyDrive/NEU-DET/train/images'
annotations_output_dir = '/content/drive/MyDrive/NEU-DET/train/labels'

# Create output directories if they don't exist
os.makedirs(images_output_dir, exist_ok=True)
os.makedirs(annotations_output_dir, exist_ok=True)

# List all images
image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]

# Augment each image
for image_file in image_files:
    image_path = os.path.join(images_dir, image_file)
    annotation_path = os.path.join(annotations_dir, image_file.replace('.jpg', '.txt'))

    # Read annotations in YOLO format
    bboxes = []
    labels = []
    with open(annotation_path, 'r') as f:
        for line in f.readlines():
            label, x_center, y_center, width, height = map(float, line.strip().split())
            labels.append(int(label))
            bboxes.append([x_center, y_center, width, height])  # Keep in YOLO format
  #Execute data augmentation function
    augment_image_and_bboxes_yolo(image_path, bboxes, labels,images_output_dir,
annotations_output_dir, augmentation_pipeline)
```

#Model training

```
#Download the YOLOv10 pre-trained weight
!mkdir -p {HOME}/weights
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10n.pt
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10s.pt
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10m.pt
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10b.pt
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10x.pt
```

```
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10l.pt
!ls -lh {HOME}/weights

# Define the data.yaml file for training
data = {
    'names': [
        'crazing',
        'inclusion',
        'patches',
        'pitted_surface',
        'rolled-in_scale',
        'scratches'
    ],
    'nc': 6, #Number of defect class
    'path': '/content/drive/MyDrive/NEU-DET', #Dataset path
    'test': 'test/images', #Tesing data path
    'train': 'train/images', #Training data path
    'val': 'val/images' #Validation data path
}

# Path to the output data.yaml file
yaml_file_path = '/content/drive/MyDrive/NEU-DET/data.yaml'

# Write the content to the YAML file
with open(yaml_file_path, 'w') as file:
    yaml.safe_dump(data, file)

print(f"data.yaml file has been successfully written to {yaml_file_path}")

#Train the YOLOv10-X model for defect detection
%cd {HOME}
!yolo task=detect mode=train epochs=300 plots=True verbose=True batch=8
optimizer='SGD' lr0=0.001 imgsz=640 patience=100 \
model={HOME}/weights/yolov10x.pt \
data={HOME}/drive/MyDrive/NEU-DET/data.yaml

#Visualise the training process
Image(filename=f'{HOME}/runs/detect/train/results.png', width=600)
```

#Model Testing

```
#Evaluate trained yolov10 on testing set
test_results = model.val(split='test',plots=True)

#Visualise testing result
# F1 curve
Image(filename=f'{HOME}/runs/detect/val/F1_curve.png', width=600)

#Precison-recall curve
Image(filename=f'{HOME}/runs/detect/val/PR_curve.png', width=600)
```

```
#precision-confidence curve
Image(filename=f'{HOME}/runs/detect/val/P_curve.png', width=600)


#recall-confidence curve
Image(filename=f'{HOME}/runs/detect/val/R_curve.png', width=600)

#Confusion matrix
Image(filename=f'{HOME}/runs/detect/val/confusion_matrix.png', width=600)

#Normalised confusion matrix
Image(filename=f'{HOME}/runs/detect/val/confusion_matrix_normalized.png', width=600)

#Compare actual label with model prediction
#Defect labels on testing batch 0
Image(filename=f'{HOME}/runs/detect/val/val_batch0_labels.jpg', width=600)

#Model prediction on testing batch 0
Image(filename=f'{HOME}/runs/detect/val/val_batch0_pred.jpg', width=600)

#Defect labels on testing batch 1
Image(filename=f'{HOME}/runs/detect/val/val_batch1_labels.jpg', width=600)

#Model prediction on testing batch 1
Image(filename=f'{HOME}/runs/detect/val/val_batch1_pred.jpg', width=600)

#Defect labels on testing batch 2
Image(filename=f'{HOME}/runs/detect/val/val_batch2_labels.jpg', width=600)

#Model prediction on testing batch 2
Image(filename=f'{HOME}/runs/detect/val/val_batch2_pred.jpg', width=600)
```

#Save the experimental data

```
#Export the model
model.export(format='onnx')

#Save all the model file into my google drive
!cp -r /content/runs /content/drive/MyDrive/yolov10_NEUDET_result/
```

#Appendix B (2) Code for GC10-DET

```
#Mount to google drive
from google.colab import drive
drive.mount('/content/drive')

#Install required package and libraries
!pip install -q git+https://github.com/THU-MIG/yolov10.git
import os
import numpy as np
import cv2
import xml.etree.ElementTree as ET
import shutil
import random
import albumentations as A
import matplotlib.pyplot as plt
from albumentations.pytorch import ToTensorV2
from ultralytics import YOLOv10
import glob
import yaml
from IPython.display import Image, display
HOME = os.getcwd()
print(HOME)
```

#Convert dataset from COCO annotation format to YOLO annotation format

```
#Define format conversion function
def convert_voc_to_yolo(xml_file, txt_file, image_width, image_height):
    try:
      tree = ET.parse(xml_file)
      root = tree.getroot()

      with open(txt_file, 'w') as f:
        for obj in root.findall('object'):
            class_name = obj.find('name').text
            class_id = class_name_to_id[class_name]  # Convert class name to class ID

            bndbox = obj.find('bndbox')
            xmin = int(bndbox.find('xmin').text)
            ymin = int(bndbox.find('ymin').text)
            xmax = int(bndbox.find('xmax').text)
            ymax = int(bndbox.find('ymax').text)

            # Convert from Pascal VOC format to YOLO format
            x_center = (xmin + xmax) / 2.0 / image_width
            y_center = (ymin + ymax) / 2.0 / image_height
            width = (xmax - xmin) / image_width
            height = (ymax - ymin) / image_height

            f.write(f"{class_id} {x_center:.6f} {y_center:.6f} {width:.6f} {height:.6f}\n")
    #Print keyerror if exist
```

```
    except KeyError as e:
        print(f"KeyError: {e} in file: {xml_file}")
    except Exception as e:
        print(f"An error occurred: {e} in file: {xml_file}")

def create_yolo_annotations(xml_folder, output_folder, image_folder):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    for xml_file in os.listdir(xml_folder):
        if xml_file.endswith('.xml'):
            xml_path = os.path.join(xml_folder, xml_file)
            image_name = os.path.splitext(xml_file)[0] + '.jpg'
            image_path = os.path.join(image_folder, image_name)


            image_width, image_height = get_image_size(image_path)  # Implement this
function to get image size

            txt_file = os.path.join(output_folder, os.path.splitext(xml_file)[0] + '.txt')
            convert_voc_to_yolo(xml_path, txt_file, image_width, image_height)

def get_image_size(image_path):

    image = cv2.imread(image_path)
    return image.shape[1], image.shape[0]

# target folder paths
xml_folder = '/content/drive/MyDrive/GC-10-DET/annotation'
output_folder = '/content/drive/MyDrive/GC-10-DET/YOLO_annotation'
image_folder = '/content/drive/MyDrive/GC-10-DET/images'

# Assign class names and corresponding IDs
class_name_to_id = {
    '1_chongkong': 1,
    '2_hanfeng': 2,
    '3_yueyawan': 3,
    '4_shuiban': 4,
    '5_youban': 5,
    '6_siban': 6,
    '7_yiwu': 7,
    '8_yahen': 8,
    '9_zhehen': 9,
    #Both '10_yaozhe' and '10_yaozhed' belongs to class 10, probably due to typo error.
    '10_yaozhe': 10,
    '10_yaozhed': 10
}

create_yolo_annotations(xml_folder, output_folder, image_folder)
```

```
#Adjusting class index to begin from 0.
annotation_path = '/content/drive/MyDrive/GC-10-DET/YOLO_annotation'

# Function to adjust class indices to start from 0.
def adjust_annotations(annotation_path):
    for filename in os.listdir(annotation_path):
        if filename.endswith('.txt'):
            file_path = os.path.join(annotation_path, filename)
            with open(file_path, 'r') as file:
                lines = file.readlines()
            with open(file_path, 'w') as file:
                for line in lines:
                    parts = line.strip().split()
                    parts[0] = str(int(parts[0]) - 1)  # Adjust class index
                    file.write(' '.join(parts) + '\n')

# Adjust annotations
adjust_annotations(annotation_path)
```

Split dataset into Train, test, and validation set.

```
#Define data splitting function
def split_data_with_annotations(source_dir, annotation_dir, train_dir, val_dir, test_dir,
                                train_anno_dir, val_anno_dir, test_anno_dir,
                                train_split=0.7, val_split=0.15, test_split=0.15):

    # Ensure the destination directories exist
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)
    os.makedirs(train_anno_dir, exist_ok=True)
    os.makedirs(val_anno_dir, exist_ok=True)
    os.makedirs(test_anno_dir, exist_ok=True)

    # Get all image file names from the source directory
    image_files = [f for f in os.listdir(source_dir) if os.path.isfile(os.path.join(source_dir, f))
and f.endswith('.jpg')]

    # Shuffle the image files
    random.shuffle(image_files)

    # Calculate the split indices
    train_idx = int(len(image_files) * train_split)
    val_idx = train_idx + int(len(image_files) * val_split)

    # Split the images
    train_files = image_files[:train_idx]
    val_files = image_files[train_idx:val_idx]
    test_files = image_files[val_idx:]

    # Copy the files and their corresponding annotations to their respective directories
    for file_name in train_files:
```

```
        shutil.copy(os.path.join(source_dir, file_name), os.path.join(train_dir, file_name))
        annotation_file = file_name.replace('.jpg', '.txt')
        annotation_path = os.path.join(annotation_dir, annotation_file)
        if os.path.exists(annotation_path):
            shutil.copy(annotation_path, os.path.join(train_anno_dir, annotation_file))

    for file_name in val_files:
        shutil.copy(os.path.join(source_dir, file_name), os.path.join(val_dir, file_name))
        annotation_file = file_name.replace('.jpg', '.txt')
        annotation_path = os.path.join(annotation_dir, annotation_file)
        if os.path.exists(annotation_path):
            shutil.copy(annotation_path, os.path.join(val_anno_dir, annotation_file))

    for file_name in test_files:
        shutil.copy(os.path.join(source_dir, file_name), os.path.join(test_dir, file_name))
        annotation_file = file_name.replace('.jpg', '.txt')
        annotation_path = os.path.join(annotation_dir, annotation_file)
        if os.path.exists(annotation_path):
            shutil.copy(annotation_path, os.path.join(test_anno_dir, annotation_file))

    print(f"Training set: {len(train_files)} images")
    print(f"Validation set: {len(val_files)} images")
    print(f"Test set: {len(test_files)} images")

#Define folder path
source_directory = '/content/drive/MyDrive/GC-10-DET/images'
annotation_directory = '/content/drive/MyDrive/GC-10-DET/YOLO_annotation'
train_directory = '/content/drive/MyDrive/GC-10-DET/train/images'
train_annotation_directory = '/content/drive/MyDrive/GC-10-DET/train/labels'
validation_directory = '/content/drive/MyDrive/GC-10-DET/val/images'
validation_annotation_directory = '/content/drive/MyDrive/GC-10-DET/val/labels'
test_directory = '/content/drive/MyDrive/GC-10-DET/test/images'
test_annotation_directory = '/content/drive/MyDrive/GC-10-DET/test/labels'

#Execute data splitting function
split_data_with_annotations(source_directory, annotation_directory,
                            train_directory, validation_directory, test_directory,
                            train_annotation_directory, validation_annotation_directory,
test_annotation_directory)
```

#Perform data augmentation on training set

```
# Define the augmentation pipeline
augmentation_pipeline = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    A.GaussianBlur(p=0.2),
    ToTensorV2()
], bbox_params=A.BboxParams(format='yolo', label_fields=['labels'], min_visibility=0.2))
```

```
# Function to augment image and bounding boxes in YOLO format
def augment_image_and_bboxes_yolo(image_path, bboxes, labels,images_output_dir,
annotations_output_dir, aug_pipeline, num_augmentations=2):
    image = cv2.imread(image_path)
    image_name = os.path.basename(image_path).split('.')[0]

    for i in range(num_augmentations):
        augmented = aug_pipeline(image=image, bboxes=bboxes, labels=labels)
        augmented_image = augmented["image"]
        augmented_bboxes = augmented["bboxes"]
        augmented_labels = augmented["labels"]

        # Convert the image to the correct format and save it
        augmented_image= augmented_image.permute(1, 2, 0).cpu().numpy()
        augmented_image_path = os.path.join(images_output_dir,
f"{image_name}_aug_{i}.jpg")
        cv2.imwrite(augmented_image_path, augmented_image)

        # Save augmented bounding boxes in YOLO format
        augmented_bboxes_path = os.path.join(annotations_output_dir,
f"{image_name}_aug_{i}.txt")
        with open(augmented_bboxes_path, 'w') as f:
            for bbox, label in zip(augmented_bboxes, augmented_labels):
                x_center, y_center, bbox_width, bbox_height = bbox
                f.write(f"{label} {x_center} {y_center} {bbox_width} {bbox_height}\n")

        print(f"Saved augmented image and bboxes: {augmented_image_path},
{augmented_bboxes_path}")

# Path where images and annotations are stored
images_dir = '/content/drive/MyDrive/GC-10-DET/train/images'
annotations_dir = '/content/drive/MyDrive/GC-10-DET/train/labels'
images_output_dir = '/content/drive/MyDrive/GC-10-DET/train/images'
annotations_output_dir = '/content/drive/MyDrive/GC-10-DET/train/labels'

# List all images
image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]

# Augment each image
for image_file in image_files:
    image_path = os.path.join(images_dir, image_file)
    annotation_path = os.path.join(annotations_dir, image_file.replace('.jpg', '.txt'))
    # Check if the annotation file exists
    if not os.path.exists(annotation_path):
        print(f"Annotation file not found, skipping: {annotation_path}")
        continue

    # Read annotations in YOLO format
    bboxes = []
    labels = []
    with open(annotation_path, 'r') as f:
```

```
        for line in f.readlines():
            label, x_center, y_center, width, height = map(float, line.strip().split())
            labels.append(int(label))
            bboxes.append([x_center, y_center, width, height])  # Keep in YOLO format
 #Execute augmentation function
    augment_image_and_bboxes_yolo(image_path, bboxes, labels,images_output_dir,
annotations_output_dir, augmentation_pipeline)
```

#Model training

```
 #Define data.yaml file for training
data = {
    'path': '/content/drive/MyDrive/GC-10-DET', #Dataset path
    'train': 'train/images',#Training data path
    'val': 'val/images',#Validation data path
    'test': 'test/images',# testing data path
    'nc': 10,# number of defect class
    'names': ['punching_hole', 'welding_line', 'crescent_gap', 'water_spot', 'oil_spot',
              'silk_spot', 'inclusion', 'rolled_pit', 'crease', 'waist_folding'
              ]
}

# Path to the output data.yaml file
yaml_file_path = '/content/drive/MyDrive/GC-10-DET/data.yaml'

# Write the content to the YAML file
with open(yaml_file_path, 'w') as file:
    yaml.safe_dump(data, file)

print(f"data.yaml file has been successfully written to {yaml_file_path}")

#Download the YOLOv10 pre-trained weight
!mkdir -p {HOME}/weights
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10n.pt
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10s.pt
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10m.pt
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10b.pt
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10x.pt
!wget -P {HOME}/weights -q
https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10l.pt
!ls -lh {HOME}/weights

%cd {HOME}
#Training YOLOv10-X based defect detection model
!yolo task=detect mode=train epochs=300 plots=True verbose=True batch=8
optimizer='SGD' lr0=0.001 imgsz=640 patience=100\
```

```
model={HOME}/weights/yolov10x.pt \
data={HOME}/drive/MyDrive/GC-10-DET/data.yaml
#Visualise the training process
Image(filename=f'{HOME}/runs/detect/train/results.png', width=600)
```

#Model testing

```
#Perform prediction on testing set
model=YOLOv10(f'{HOME}/runs/detect/train/weights/best.pt')
model.predict(source='/content/drive/MyDrive/GC-10-DET/test/images', save=True)

#Visualzie some prediction result
for image_path in glob.glob(f'{HOME}/runs/detect/predict/*.jpg')[:10]:
  display(Image(filename=image_path, width=600))
  print("\n")

#Evaluate trained yolov10 on testing set
test_results = model.val(imgsz=640,split='test',plots=True)
```

#Visualise testing result

```
#F1 curve
Image(filename=f'{HOME}/runs/detect/val/F1_curve.png', width=600)

#Precision-recall curve
Image(filename=f'{HOME}/runs/detect/val/PR_curve.png', width=600)

#Precision-confidence curve
Image(filename=f'{HOME}/runs/detect/val/P_curve.png', width=600)

#Recall-confidence curve
Image(filename=f'{HOME}/runs/detect/val/R_curve.png', width=600)

#Confusion matrix
Image(filename=f'{HOME}/runs/detect/val/confusion_matrix.png', width=600)

#Normalised confusion matrix
Image(filename=f'{HOME}/runs/detect/val/confusion_matrix_normalized.png', width=600)
```

#Compare actual defect labels with model prediction

```
#Defect labels for testing batch 0
Image(filename=f'{HOME}/runs/detect/val/val_batch0_labels.jpg', width=600)

#Model prediction on testing batch 0
Image(filename=f'{HOME}/runs/detect/val/val_batch0_pred.jpg', width=600)

#Defect labels on testing batch 1
Image(filename=f'{HOME}/runs/detect/val/val_batch1_labels.jpg', width=600)

#Model prediction on testing batch 1
Image(filename=f'{HOME}/runs/detect/val/val_batch1_pred.jpg', width=600)
```

```
#Defect labels on testing batch 2
Image(filename=f'{HOME}/runs/detect/val/val_batch2_labels.jpg', width=600)
#Model prediction on testing batch 2
Image(filename=f'{HOME}/runs/detect/val/val_batch2_pred.jpg', width=600)
```

#Save the experimental data

```
#Export the model
model.export(format='onnx')

#Save all the model file into  google drive
!cp -r /content/runs /content/drive/MyDrive/YOLOv10_result_GC10-DET/
```