Welding Defect Detection with YOLO

Intelligent Systems

Fadi Aldeeb – 112537

Supervisor: prof. Joao Miguel da Costa Sousa

GitHub: [Fadialdeeb/Project---Welding-Defect-Detection: Instituto Superior Técnico - Intelligent Systems]

Master Mechanical Engineering

Academic year 2024-2025

27.01.2025

Table of Contents

Figures and Tables

1 Introduction

Welding defects can critically impact the structural integrity of welded materials, making their early detection vital. Many industries, such as automobile, aerospace, heavy industries, etc, have been interested in welding automation due to cost reduction and improving productivity. Most of previous studies have analysed the influence on the bead shape and weld defects by current, voltage, welding speed (Junsung Bae, 2024). As the demand for automation grows, integrating advanced detection techniques becomes crucial for maintaining quality and efficiency. One method of detecting these defects is recognition based on machine vision, which provides a reliable basis for robot arc welding. However, due to such factors as severe noise interference of welding images, slight morphological differences of same defect, and restricted computational capacity of on-site devices, it has been a challenge to learn distinguishable characteristics from various weld seam defects as well as to improve defects recognition accuracy and model generalizability (Yue Zhang, 2024). Several object detection models, particularly Ultralytics model "You Only Look Once" (YOLO) family (Ultralytics Inc. , 2024), have gained prominence in computer vision for their speed and accuracy.

As a mechanical engineer, the author chose this project as an example of integrating artificial intelligence with mechanics to address complex industrial challenges. Therefore, this report outlines the development of a YOLO-based model tailored for detecting welding defects, leveraging a dataset from Kaggle, and compares its results with those of prior contributors. It further identifies the advantages and disadvantages of the YOLO model, to see the possibility of using the model in actual companies.

2 Scope of the project

This project includes the application of object detection techniques to automate the identification of welding defects. Leveraging machine vision to improve the accuracy, and reliability of defect detection in industrial settings. Specifically, it involves:

- Developing a YOLO-based object detection model customized for welding defect detection.
- Using an open-source dataset, to train and evaluate the model.
- Comparing the YOLO model's performance against other contributors' models.
- Comparing the YOLO model's performance against an alternative model, RetinaNet, to understand their relative strengths and weaknesses.
- Evaluate models performances using standard metrics such as mAP (mean Average Precision), precision, and recall.
- Highlighting the potential for integrating AI with mechanical engineering to advance industrial automation.

The choice fell on YOLO because of its reputation as an acclaimed real-time object detection and image segmentation model, built on cutting-edge advancements in deep learning and

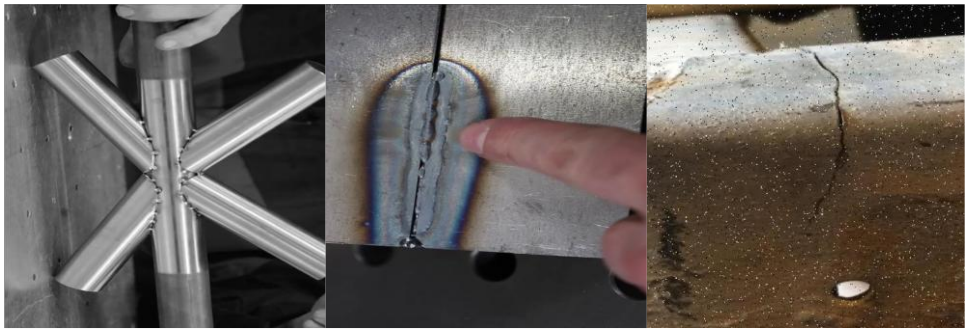
computer vision. YOLO divides a picture into a grid of NxM, for each cell, it tries to predict a bounding box for an object that would be cantered in that cell. The predicted bounding box can be larger than the cell from which it originates; the only constraint is that the centre of the box is somewhere inside the cell. Predicting a bounding box amounts to predicting six numbers: the four coordinates of the bounding box (in this case, the x and y coordinates of the centre, and the width and height), a confidence factor which tells us if an object has been detected or not, and finally, the class of the object (Lakshmanan, Görner, & Gillard, 2021).

3 Information on Data Set

For this project, the dataset used for training and evaluation was sourced from Kaggle titled: '*Welding Defect - Object Detection*', published by (Wijaya, 2024). A thorough analysis of the dataset was conducted, presented in this chapter the results of the analysis.

3.1 Class Distribution

The dataset contains 2028 welding images, all uniform size of 640x640 pixels, samples of these images are shown in Figure 1.



(a) (b) (c)

Figure 1: Images Samples (a) Good Weld, (b) Bad Weld, (c) Defect

The dataset contains three classes: Bad Weld, Good Weld, and Defect, Table 1 shows the corresponding index for each class.

Class Index	Class Name
0	Bad Weld
1	Good Weld
2	Defect

Table 1: Dataset Classes with the Corresponding Indices

Manual examination of the images revealed that Data Augmentation is pre-applied on the dataset by the source creator, an example is presented in Figure 2.

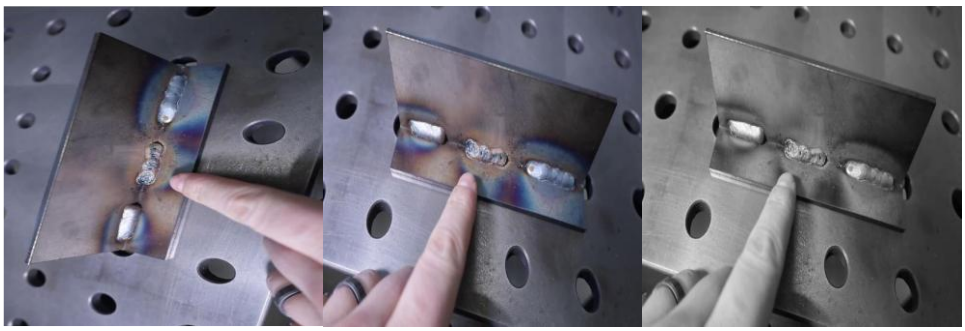


Figure 2: Data Augmentation Example

As we can see from Figure 3 and Figure 4, the distribution of the images is almost uniform among the three classes, with more emphasis on "Good Weld" pictures. But, if we look at from a perspective of two class classification problem, then we see that classes "Bad Weld" and "Defect" make 59% of the dataset compared to 41% of the "Good Weld" class. The emphasis in this case is shifted towards catching bad defects.

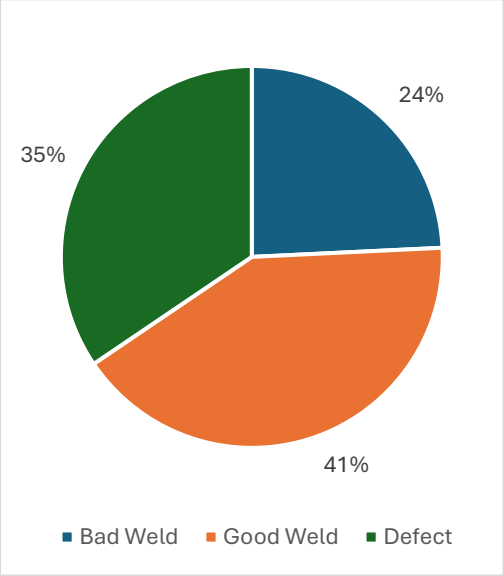


Figure 3: Percentage of Each Class

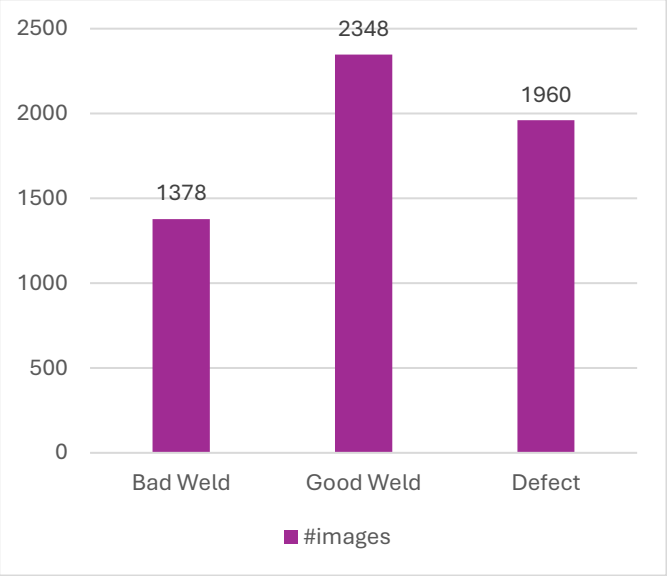


Figure 4: Number of Images in each Class

3.2 Subsets Split

Due to the small size of the dataset, it is ***not advisable*** to split the dataset into three subsets. Yet, the current split was used for two main reasons:

1. Ultralytics YOLO model uses a subset to assess the model while training, therefore there is a need to have a third dataset to assess the model.
2. The split was presented from the source, merging and re-splitting the dataset seemed unnecessarily complicated, due to the method used for splitting and labeling the images.

The dataset was split into three sets, as shown in Figure 5. The “Train Set” is used to train the model, while the “Validation Set” is used during training to monitor the model's performance. The “Test Set” is used after training to evaluate the model's final performance.

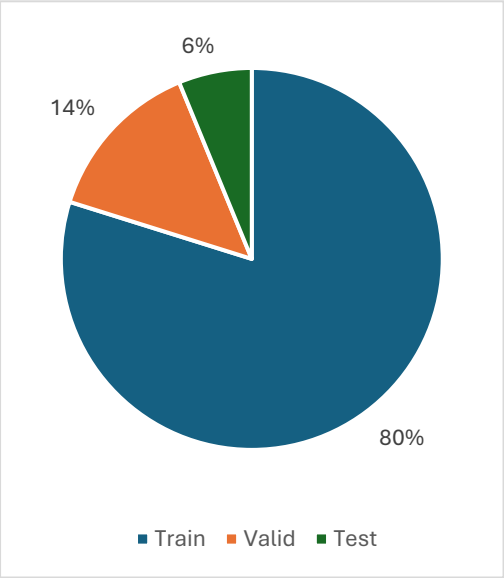


Figure 5: Percentage of Each Subset

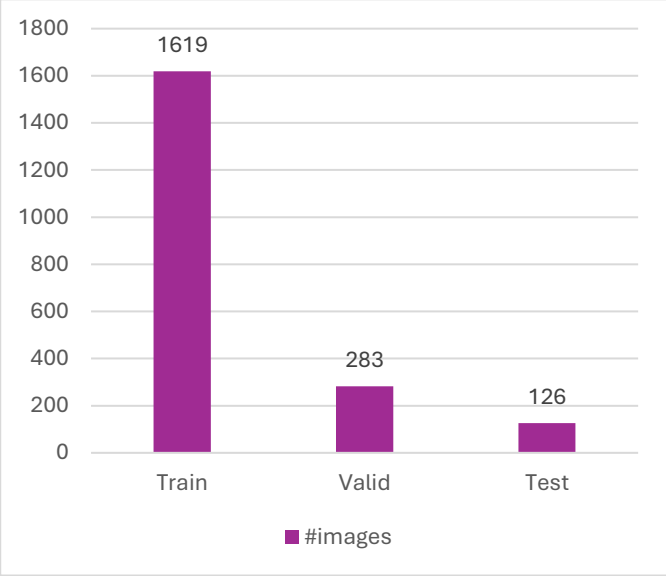


Figure 6: Number of Images in each Subset

Figure 7, shows the distribution of images of the three classes in each subset. We can see that the test set of 129 images, has a uniform distribution of classes, which is crucial to ensure an unbiased test.

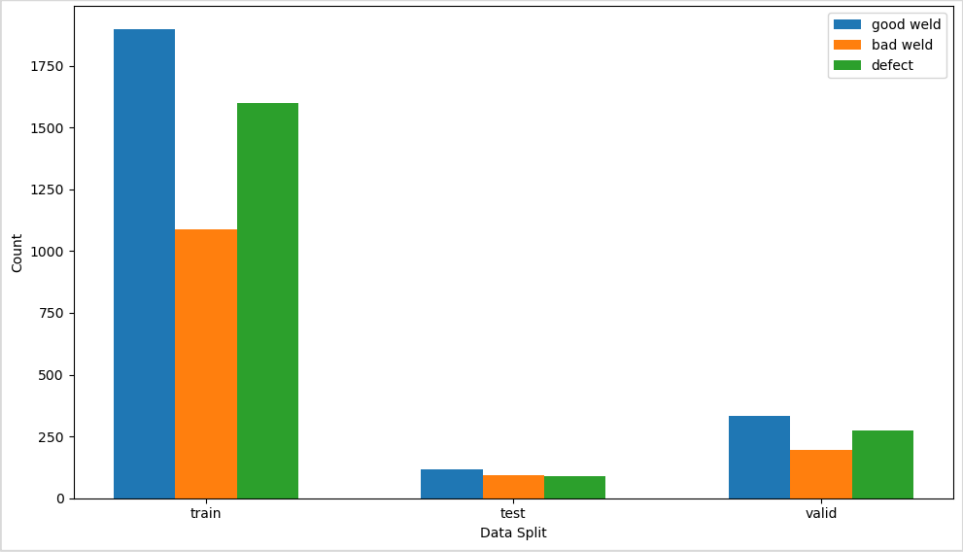


Figure 7: Class Distribution per Data Split

3.3 Bounding Boxes

Although not mention explicitly by the dataset creator, the bounding boxes of the images were created manually using a labelling tool, like Labellerr (Singh, 2024). These tools can extract the coordinates of the bounding boxes and the classes which we will later use for our model. Table 2 shows a sample of the output.

Class	x-center	y-center	width	height
0	0.4890625	0.41875	0.275	0.30625
2	0.56328125	0.515625	0.084375	0.075

Table 2: Label Sample

If we look at the distribution of the height and width of bounding boxes, presented in Figure 8, as a fraction of the image's dimensions, we can see that we have a large range of different dimensions, with the majority occupying 20% of the image. This means that the model may need to focus on learning features associated with smaller objects which requires a network with strong feature extraction capabilities.

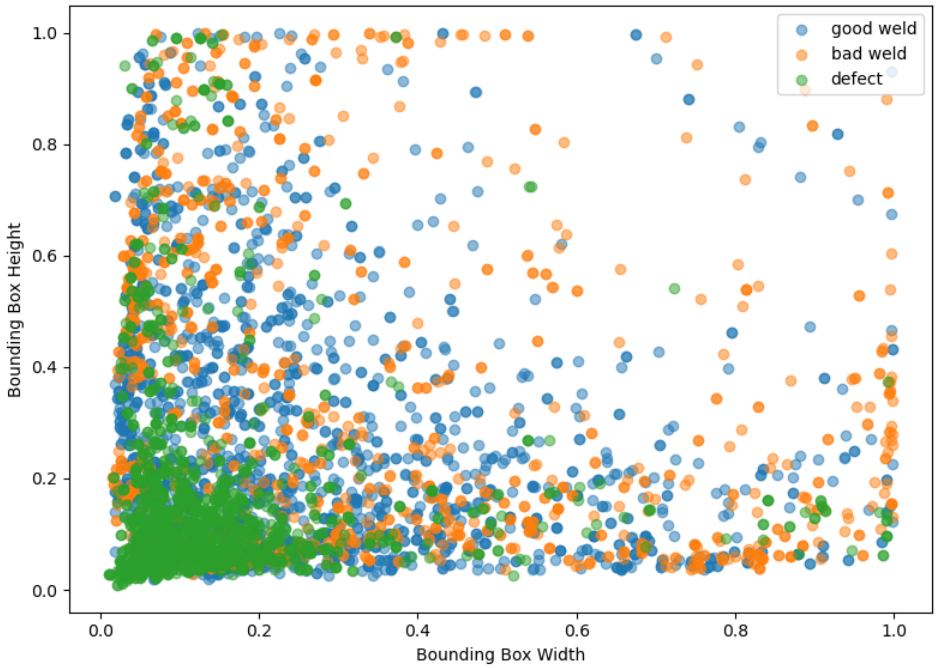


Figure 8: Distribution of Object Sizes

3.4 Community Contributors

The results of this project were benchmarked against outcomes achieved by other programmers on the same dataset, as documented on the Kaggle dataset website (Wijaya, 2024). This comparison includes nine models for a comprehensive evaluation as shown in Table 3:

#	Publisher	Date Published	Model Used	precision	recall	f1	mAP50[1]	mAP50-95[2]
1	*Mohammed Mohsen*	*Dec 18, 2024*	*yolov8m*	*0.65*	*0.69*	*0.66*	*0.67*	*0.41*
2	*Sinem Çelik*	*Oct 04, 2024*	*Yolov9t*	-	-	-	-	-
3	*Mike DeLong*	*Sep 01, 2024*	*resnext50*	*0.93*	*0.91*	*0.92*	-	-
4	*Quan Hoang Ngoc*	*Aug 30, 2024*	*yolov8n*	*0.57*	*0.51*	*0.53*	*0.55*	*0.35*
5	*Kero Ashraf*	*Aug 14, 2024*	*yolov9m*	*0.56*	*0.68*	*0.61*	*0.64*	*0.38*
6	*Yossef Mohammed*	*Aug 11, 2024*	*yolov9m*	*0.56*	*0.68*	*0.61*	*0.64*	*0.38*
7	*Muhammad Ellbendl Satria*	*Aug 06, 2024*	*yolov8m*	*0.69*	*0.49*	*0.57*	*0.56*	*0.3*
8	*Muhammad Faizan*	*Jul 14, 2024*	*yolov8m*	*0.78*	*0.71*	*0.74*	*0.76*	*0.52*
9	*Sukma Adhi Wijaya*	*Jun 30, 2024*	*yolov8m*	*0.79*	*0.70*	*0.74*	*0.77*	*0.54*

Table 3: Models Metrics Achieved by Contributors

Mike DeLong's ResNext50 model outperformed others with a precision of 0.93 and recall of 0.91, yielding an F1 score of 0.92, yet further inspection revealed that Mike DeLong is not performing object detection. Instead, it is doing multi-image classification and cannot localize defects within images, therefore it was not considered.

Precision ranged from 0.65 to 0.79, and recall ranged from 0.49 to 0.71 across contributors using YOLO models. Sukma Adhi Wijaya achieved the highest F1 score (0.74) among YOLOv8m users, therefore it will be taken as a benchmark.

[1] **mAP50:** Mean average precision calculated at an intersection over union (IoU) threshold of 0.50. It's a measure of the model's accuracy considering only the "easy" detections.

[2] **mAP50-95:** The average of the mean average precision calculated at varying IoU thresholds, ranging from 0.50 to 0.95. It gives a comprehensive view of the model's performance across different levels of detection difficulty.

4 The Models

For this project, four YOLO models were used: YOLOv8n, YOLOv8m, YOLOv11n, and YOLOv11m[3]. Version 8 was used to have a comprehensive comparison of other existing results, while version 11 was used regarding the it's the latest version. All four models were pretrained on the "COCO" dataset. The models were trained multiple times then tuned to find the optimal parameters, presented below in Table 4: Final Used Parameters are the best parameters values. The method of choosing the parameters was "trial and error", due to the large RAM requirements of the models, applying grid search was nearly impossible. Additionally, applying any dropout increased the loss significantly due to the small dataset.

Parameter	Value
Number of Epochs	100
Batch Size	8
Images Size	640
Device	Nvidia RTX 4060 GPU
Number of Workers	0 (Serial)
Optimizer	Auto
Enable Validation	True
Learning Rate	0.01
Drop Out	0

Table 4: Final Used Parameters of YOLO Models

4.1 YOLOv8n

Training the first model highlighted clearly the large computational requirements the model, this model which is considered simple among the YOLO family, would cause the device to crash, therefore the computation was moved to the **GPU** instead, and the batch size reduced gradually until reached **8 images per batch**. This allowed the model to run **100 epochs** in around **2.932 hours**, utilizing **3.89GB** of GPU memory.

Table 5 shows the metrics for the model. The model performs well on easy predictions for all class except "Defect" and performs somewhat badly on tough predictions.

Class	Instances	Precision	Recall	mAP50	mAP50-95
All	802	0.75	0.70	0.75	0.50
Bad Weld	194	0.72	0.78	0.81	0.59
Good Weld	335	0.81	0.83	0.87	0.63
Defect	273	0.72	0.50	0.56	0.56

Table 5: YOLOv8n Metrics

[3] The letter after the version refers to the size of the model, n: nano, t: tiny, m: medium, l:large, x: extra large

From the confusion matrix, Figure 9 we can see that the model is have a good diagonal dominance compared to the number or classes instances. Yet, the model is classifying some background objects as other classes and dismissing some defects as background.

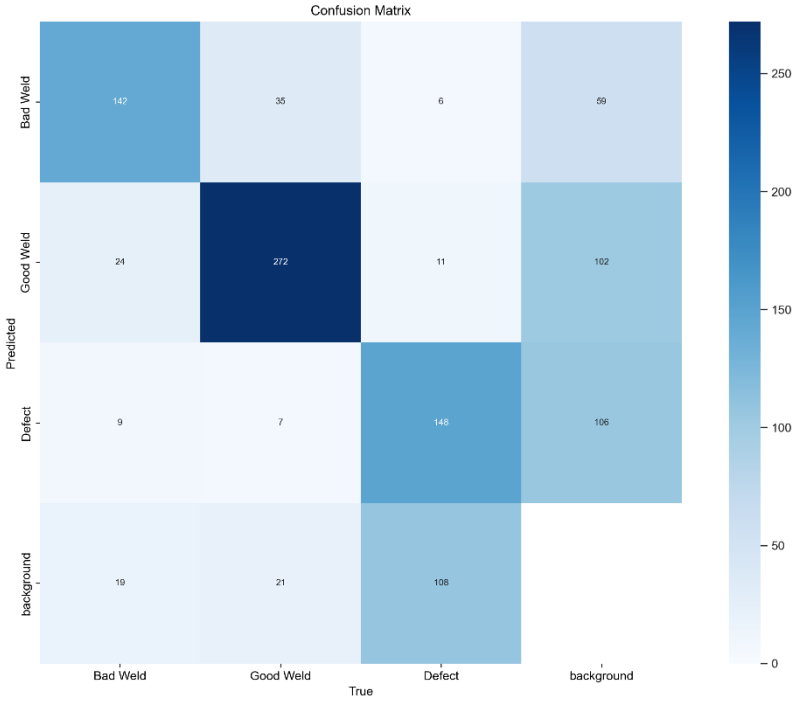


Figure 9: YOLOv8n Confusion Matrix[4]

The provided curves shown in Figure 10 show the model's performance, showing good overall recall and precision across different classes. The "Recall-Confidence" curve suggests high recall at lower confidence, while the "Precision-Confidence" curve indicates high precision at higher confidence. The "Precision-Recall" curve has an AUC of 0.743, indicating a decent balance between precision and recall. The "F1-Confidence" curve shows an optimal F1-score of 0.71 at a confidence threshold of 0.42. The model's performance could be further improved by addressing potential class imbalance and fine-tuning hyperparameters.

[4] **Background FP**: Refers to background objects that do not belong to either of the classes but detected as one of them. **Background FN:** Refers to objects missed by the detector and considered as some other background objects, (Aliy, 2022).

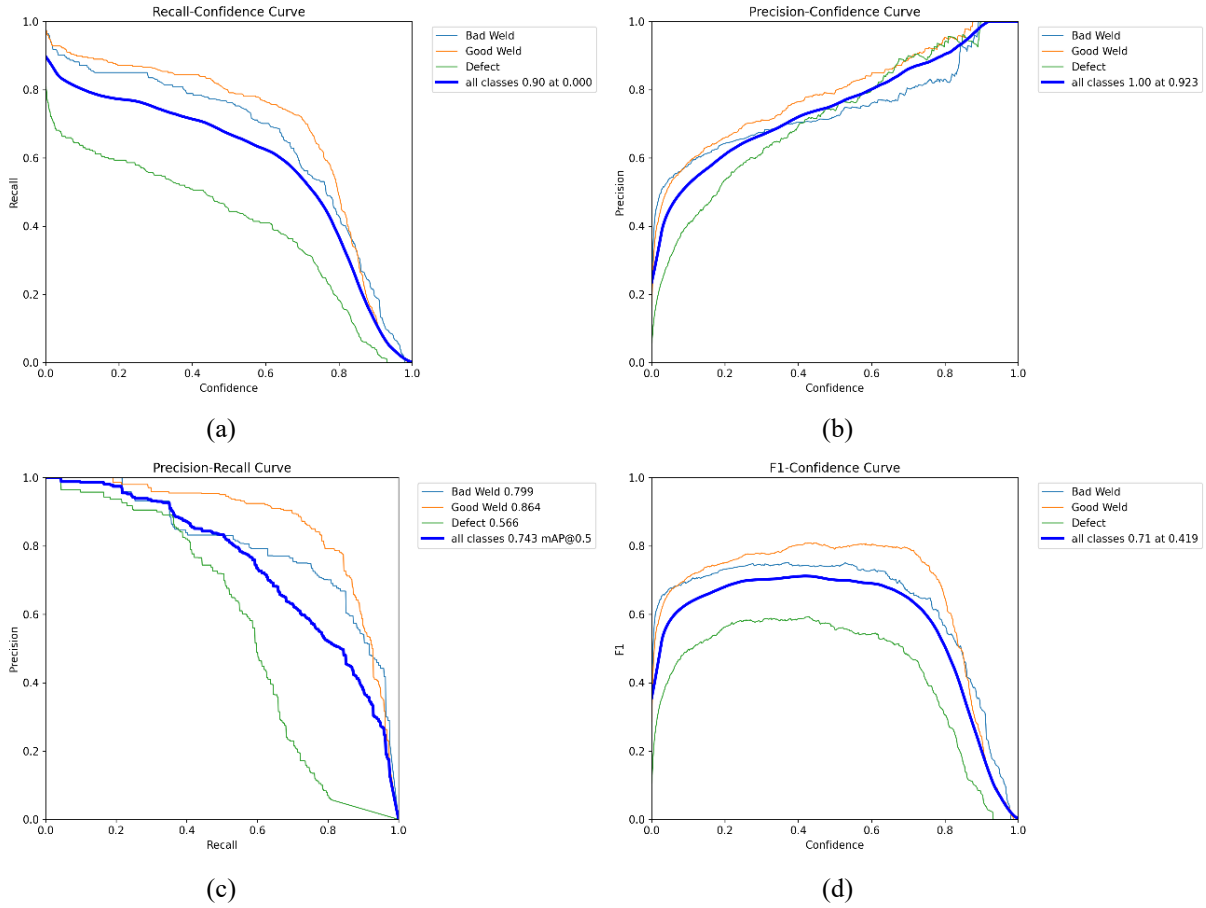


Figure 10: (a) Recall Confidence Curve (b) Precision Confidence Curve (c) Precision Recall Curve (d) F1 Confidence Curve

The plots provided in Figure 11 suggest that the YOLO model is training well. The decreasing losses and increasing metrics indicate that the model is learning to detect and classify objects more accurately. However, at around 90 Epochs, we can see signs of overfitting.

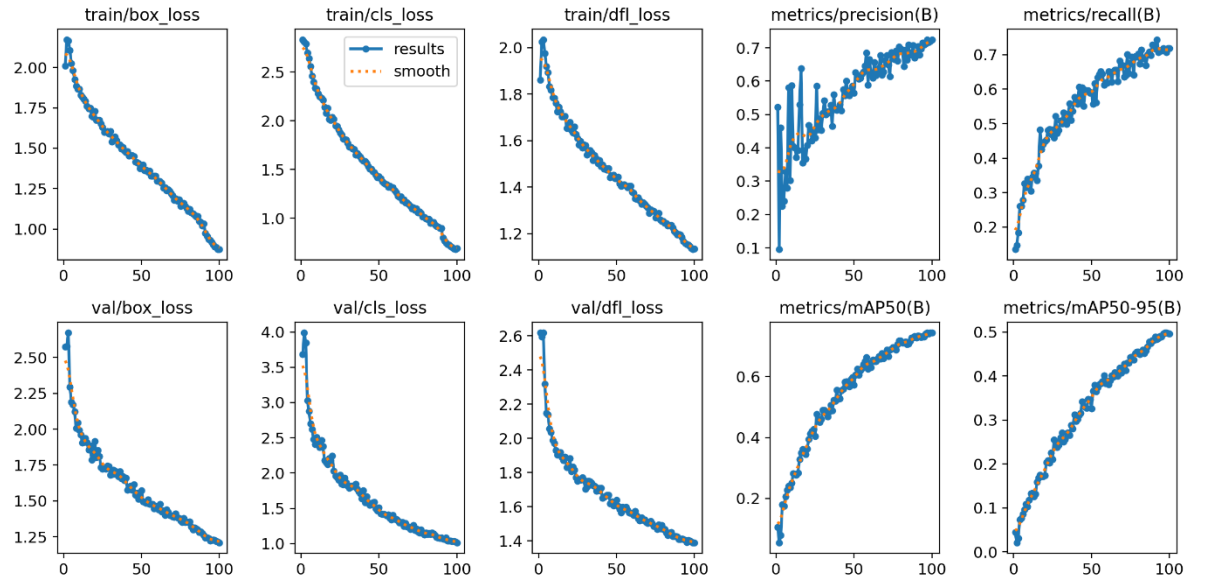
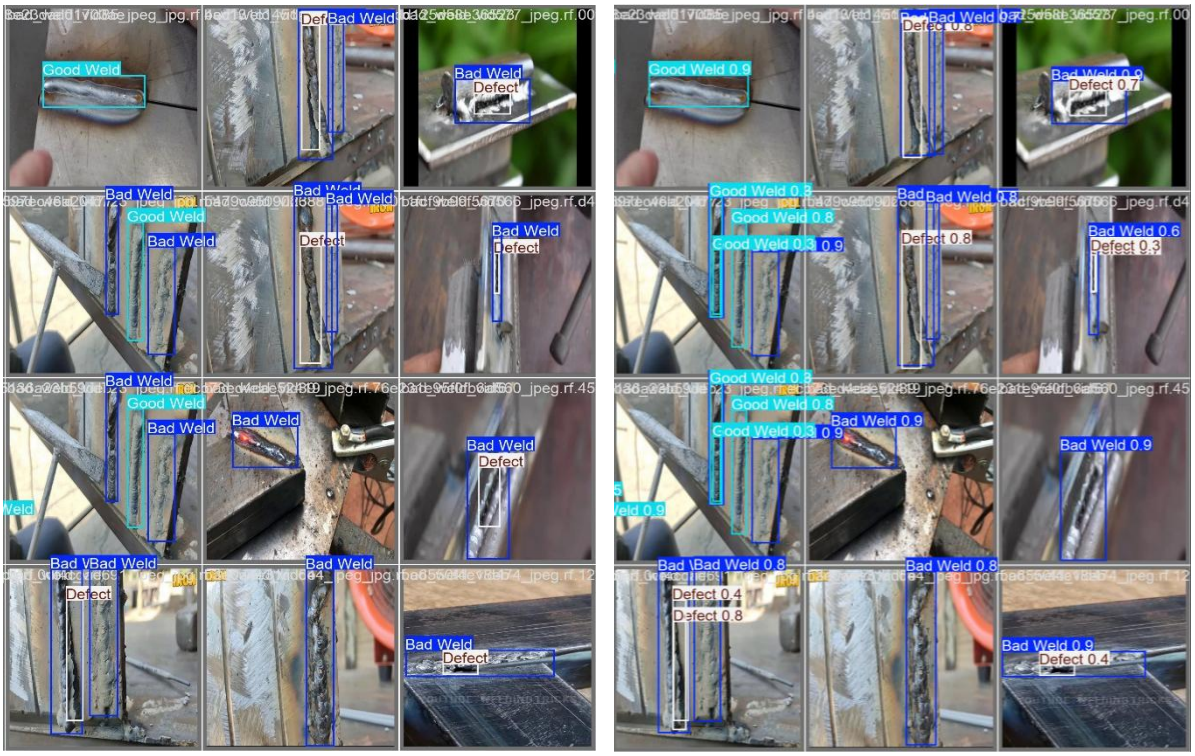


Figure 11: Training and Validation Curves for YOLOv8n Model



(a) (b)

Figure 12: Test Example (a) Labels (b) Predictions

4.2 YOLOv8m

This model ran **100 epochs** in **2.997 Hours**, utilizing the same amount of GPU memory: **3.89GB**.

Overall, the YOLOv8m model exhibits decent performance with an mAP50 of 0.74 and mAP50-95 of 0.49, as shown in Table 6. This indicates good object detection accuracy, particularly at a higher IoU threshold. Class-wise performance varies. "Good Weld" is detected most accurately, while "Defect" poses the greatest challenge with lower recall and mAP50.

Class	Instances	Precision	Recall	mAP50	mAP50-95
All	802	0.72	0.71	0.74	0.49
Bad Weld	194	0.70	0.78	0.79	0.57
Good Weld	335	0.76	0.84	0.86	0.62
Defect	273	0.70	0.50	0.56	0.29

Table 6: YOLOv8m Metrics

The confusion matrix, Figure 13 reveals that the model excels at classifying "Good Weld" instances, which can bias the model. The model struggles to detect "Defect" instances, leading to a high number of false negatives. Objects are incorrectly classified as background, at this point it is indicating potential issues with object classification.

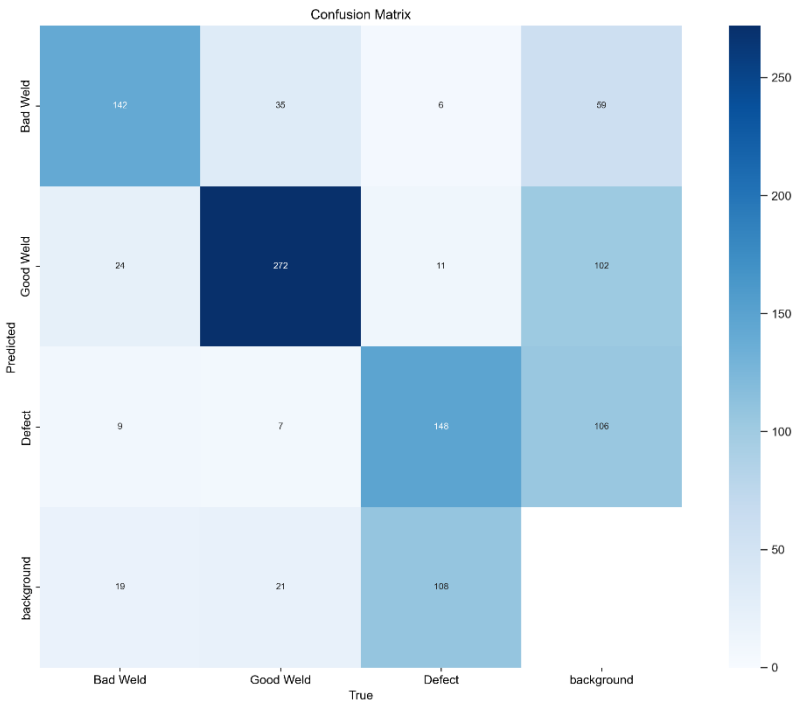


Figure 13: YOLOv8m Confusion Matrix

The "All Classes" curve has an AUC of 0.743, suggesting a decent balance between precision and recall, and achieves a good F1-score of 0.71 at a confidence threshold of around 0.42. See Figure 14.

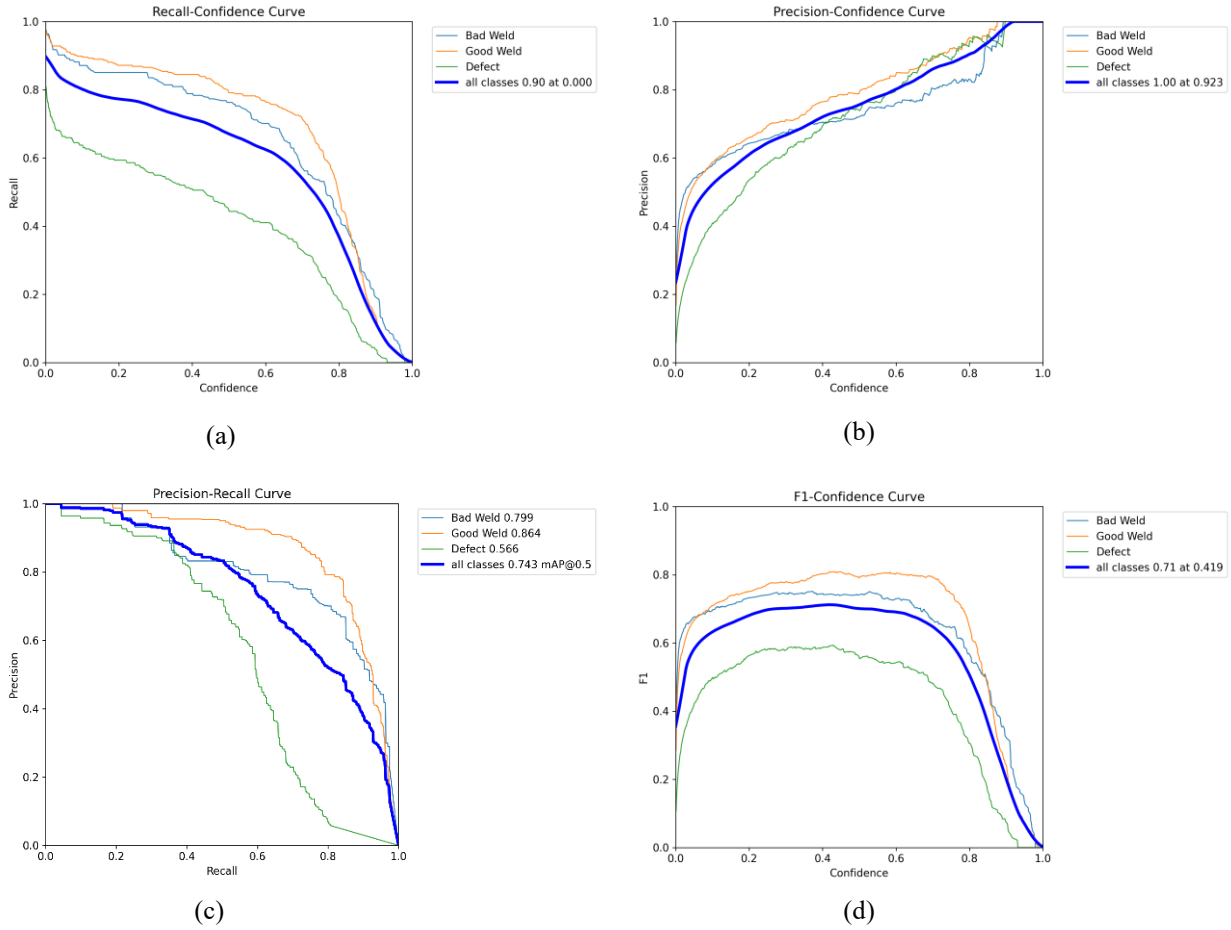


Figure 14: (a) Recall Confidence Curve (b) Precision Confidence Curve (c) Precision Recall Curve (d) F1 Confidence Curve

From Figure 15 we can see that the model is learning and generalizing well, like YOLOv8n model, with overfitting at the 90th epoch.

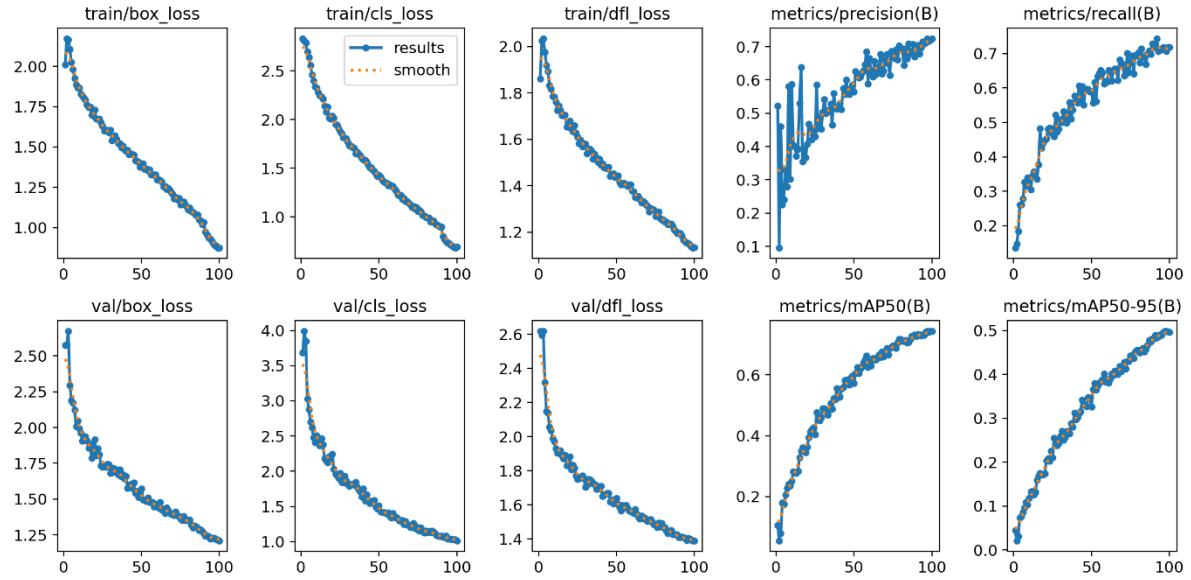
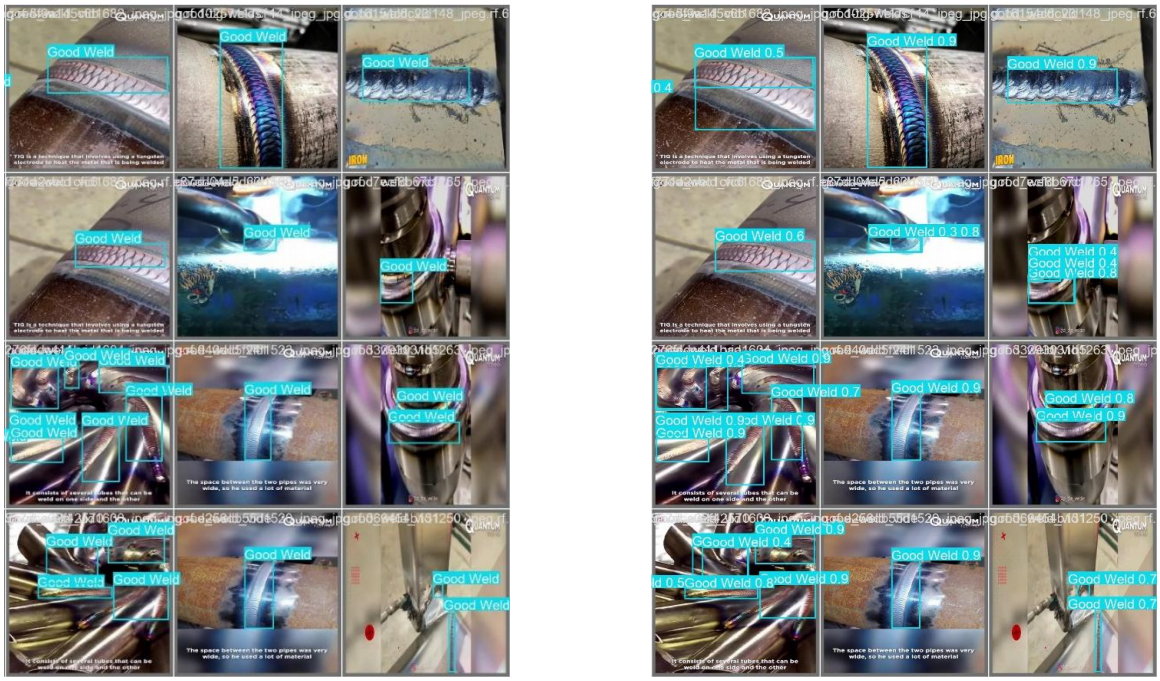


Figure 15: Training and Validation Curves for YOLOv8m Model



(a) (b)

Figure 16: Test Example (a) Labels (b) Predictions

4.3 YOLOv11n

This latest YOLO model finished the **100 epochs** in **2.651 hours**, using only **1.29GB** of GPU memory.

This model showed similar performance as its predecessors, with significantly worst mAP50-95 metric. Meaning it was hard for the model to detect hard-to-learn features, see Table 7.

Class	Instances	Precision	Recall	mAP50	mAP50-95
All	802	0.75	0.64	0.71	0.45
Bad Weld	194	0.76	0.76	0.80	0.56
Good Weld	335	0.78	0.78	0.83	0.56
Defect	273	0.70	0.36	0.49	0.23

Table 7: YOLOv11n Metrics

The confusion matrix in Figure 17 shows more defect objects are incorrectly classified as background, compared to the previous models.

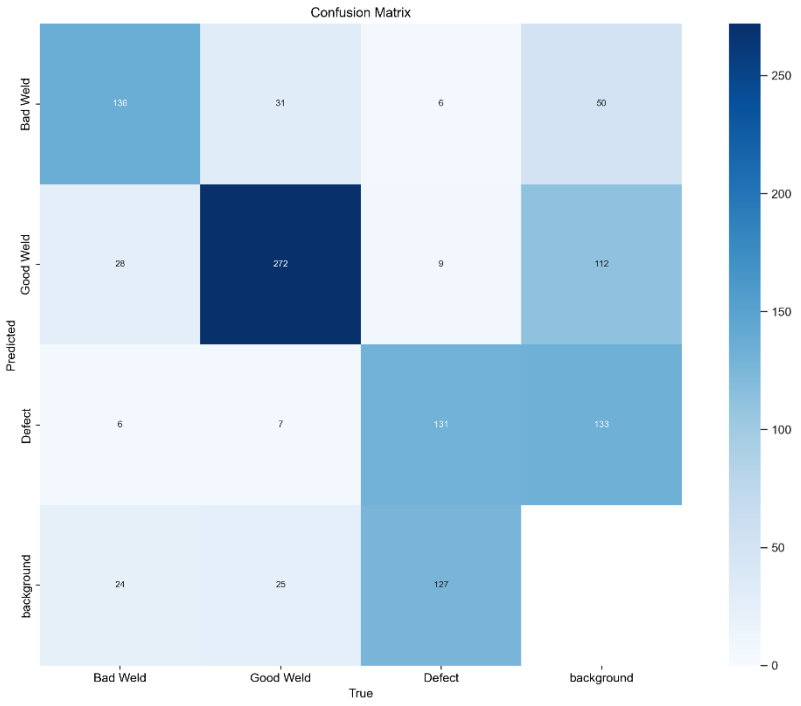


Figure 17: YOLOv11n Confusion Matrix

The Precision-Recall curve in Figure 18 shows a decent balance between precision and recall, with an AUC of 0.711. The F1-score reaches a peak of 0.68 at a confidence threshold of 0.437. These results indicate a model capable of detecting objects with reasonable accuracy and confidence.

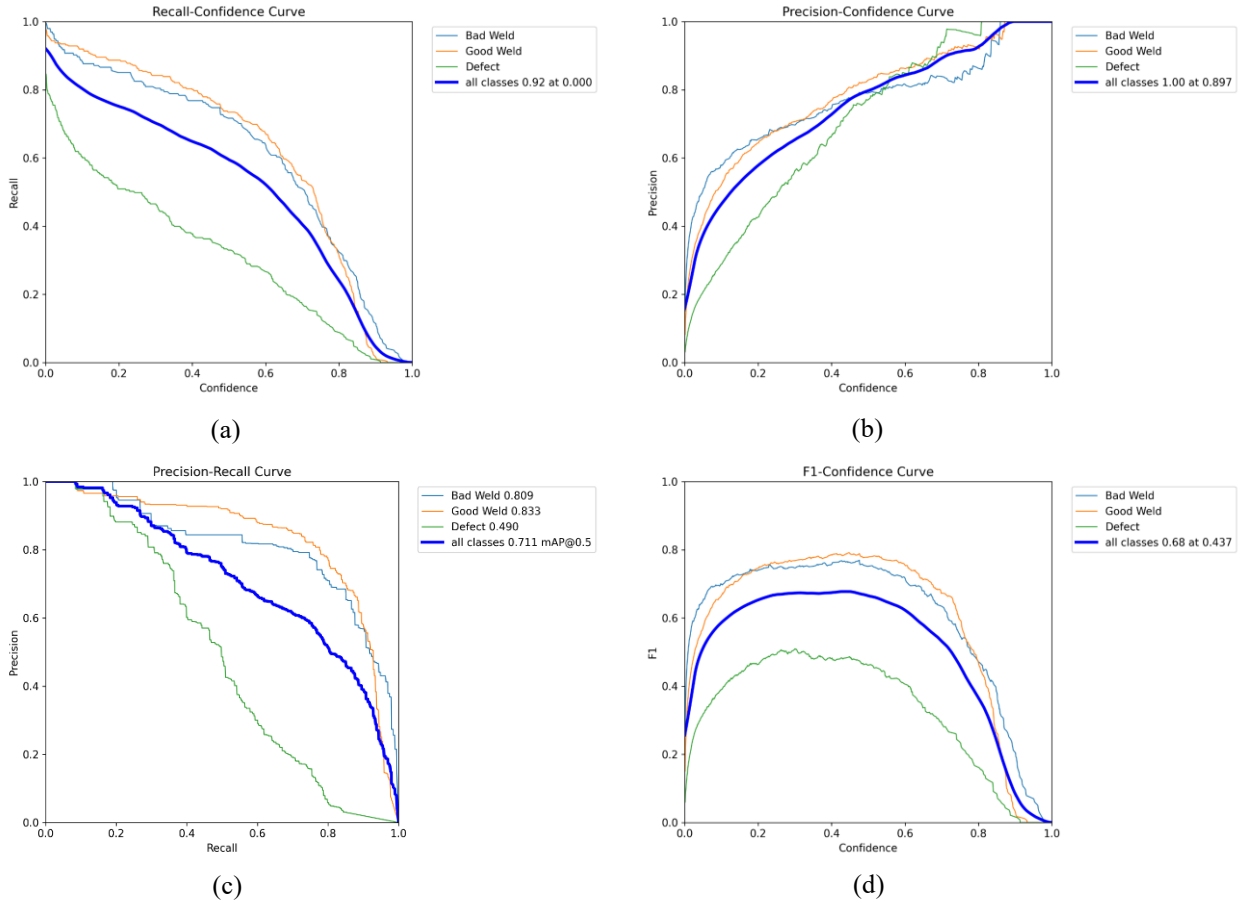


Figure 18: (a) Recall Confidence Curve (b) Precision Confidence Curve (c) Precision Recall Curve (d) F1 Confidence Curve

From Figure 19, we can see that the model is not overfitting significantly. The curves seem like previous models tested, which suggests a good network architecture, from here we can conclude that the low precision we have is due to the small dataset size, yet we will train a fourth model to be sure.

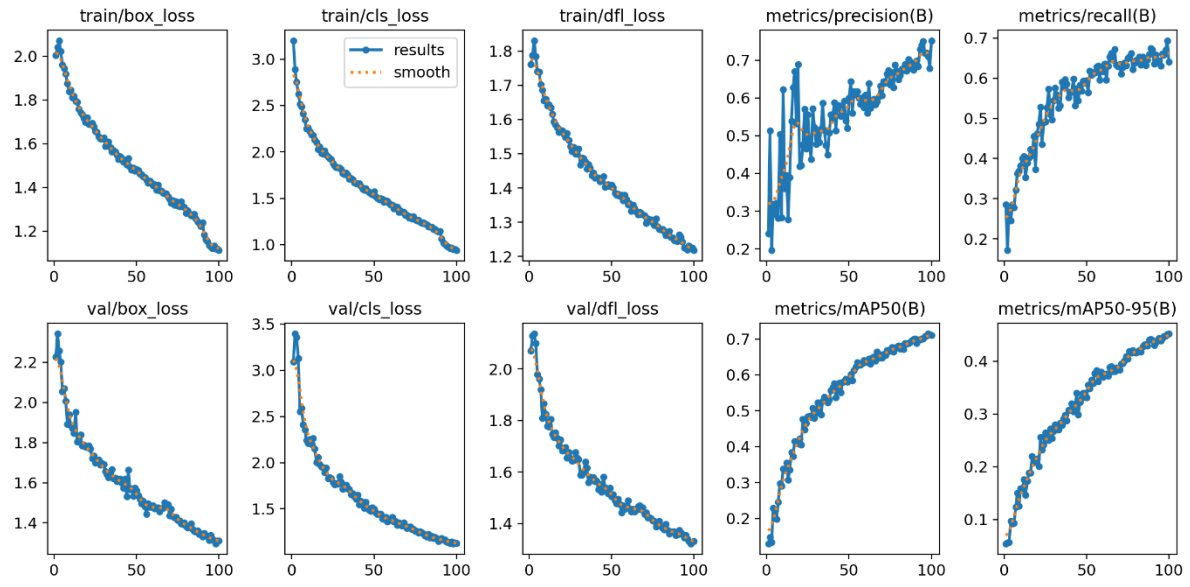
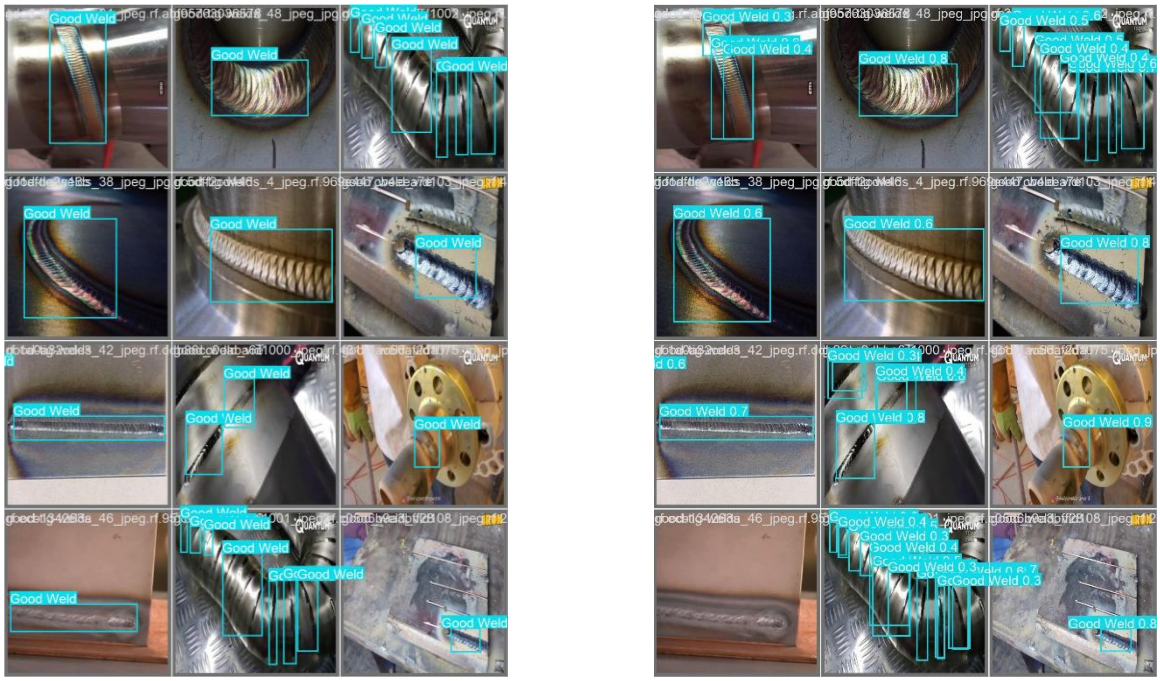


Figure 19: Training and Validation Curves for YOLOv11n Model



(a) (b)

Figure 20: Test Example (a) Labels (b) Predictions

4.4 YOLOv11m

This model completed **100 epochs** in **3.041 hours**, using **4.71GB** of GPU memory.

Overall, the model exhibits moderate performance with an mAP50 of 0.68 and mAP50-95 of 0.43, see Table 8. Yet, it struggles with the "Defect" class, showing the lowest metrics values among all models.

Class	Instances	Precision	Recall	mAP50	mAP50-95
All	802	0.67	0.66	0.68	0.43
Bad Weld	194	0.69	0.80	0.77	0.53
Good Weld	335	0.75	0.76	0.80	0.56
Defect	273	0.57	0.42	0.42	0.22

Table 8: YOLOv11m Metrics

The confusion matrix, Figure 21 is the same as the previous models, since this model is the "Medium" version, it is expected to behave worse on small datasets and cause underfitting.

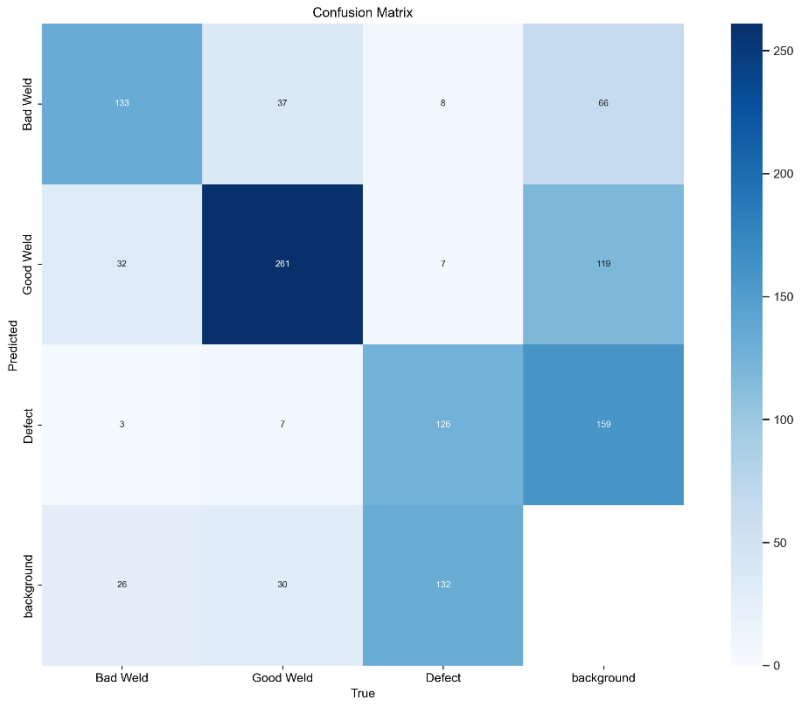


Figure 21: YOLOv11m Confusion Matrix

From Figure 22, we see that the F1-score reaches a peak of 0.66 at a confidence threshold of 0.340, which is not good. Yet like other models the rest of the curves show room for improvement.

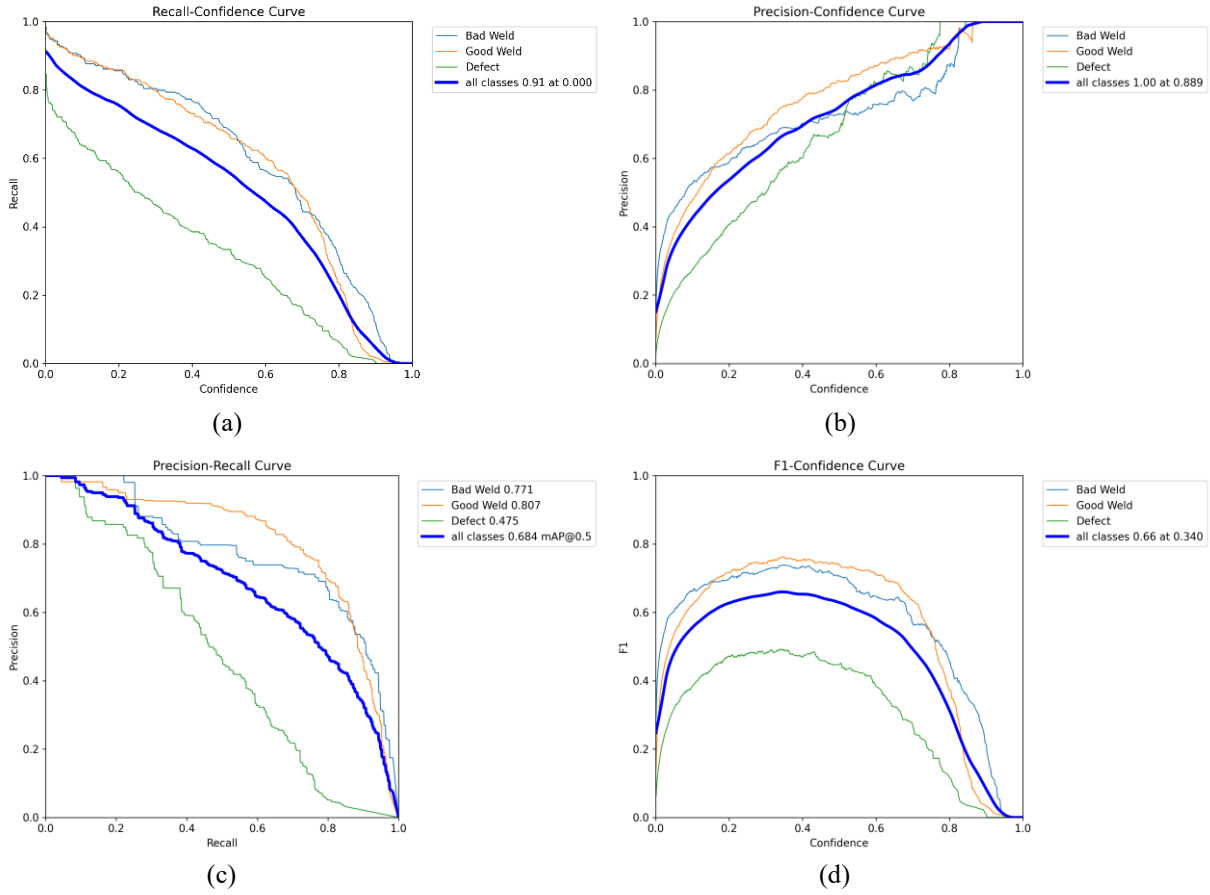


Figure 22: (a) Recall Confidence Curve (b) Precision Confidence Curve (c) Precision Recall Curve (d) F1 Confidence Curve

Figure 23 shows that the combination of decreasing losses and increasing metrics suggests a successful training process where the model is effectively learning from the data, yet the low values clearly indicate that the model is underfitting.

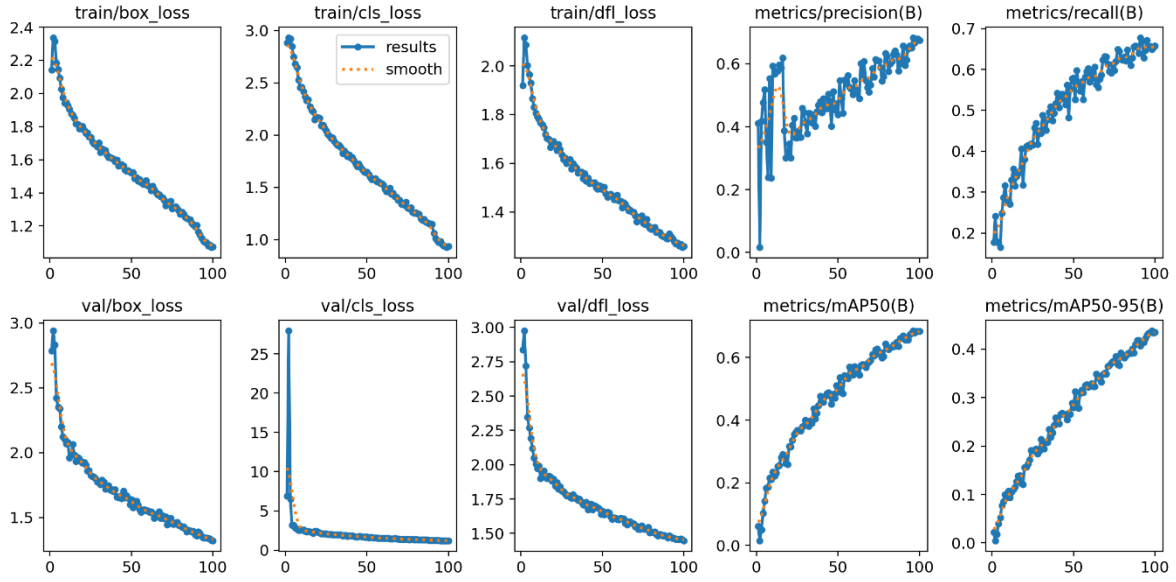


Figure 23: Training and Validation Curves for YOLOv11m Model

5 Comparing the Models

5.1 Comparison of Trained Models

The four YOLO models were evaluated using metrics such as precision, recall, F1 score, mean Average Precision (mAP), running time, and GPU usage, as presented Table 9.

Model	Running time [h]	GPU Usage [MB]	Precision	Recall	F1 Score	mAP50	mAP50-95
V8n	2.932	3.89	0.75	0.70	0.72	0.75	0.50
V8m	2.997	3.89	0.72	0.71	0.71	0.74	0.49
V11n	2.561	1.29	0.75	0.64	0.69	0.71	0.45
V11m	3.041	4.71	0.67	0.66	0.66	0.68	0.43

Table 9: Trained Models Comparison Metrics

Accuracy: V8n demonstrated the highest overall accuracy, with the highest metrics values. V11n achieved the same highest precision, while V8m shared the highest recall.

Efficiency: V11n exhibited the fastest running time, making it the most efficient model in terms of speed, and it had the lowest GPU usage, requiring the least amount of memory.

Overall: Based on these metrics, V8n appears to be the best-performing model overall. It achieves a good balance between precision, recall, and F1 score, has the highest mAP50 and mAP50-95, and has a reasonable running time respectively.

5.2 Comparison with other Contributors

Two more comparisons were made with the best model of the contributors (seen in Table 3). The first represented in Table 10, was made with corresponding model trained. While the second represented in Table 11, was made with our best model.

Model	Device Used	Usage	batch size	Epochs	Time [h]	Precision	Recall	F1	mAP50	mAP50-95
My V8m	Nvidia RTX 4060	3.89GB	8	100	2.997	0.72	0.71	0.71	0.74	0.49
Wijaya V8m	Tesla T4	4.17GB	16	120	1.150	0.79	0.70	0.74	0.77	0.54

Table 10: Comparison of Similar Models

Model	Device Used	Usage	batch size	Epochs	Time [h]	Precision	Recall	F1	mAP50	mAP50-95
My V8n	Nvidia RTX 4060	3.89GB	8	100	2.932	0.75	0.70	0.72	0.75	0.50
Wijaya V8m	Tesla T4	4.17GB	16	120	1.150	0.79	0.70	0.74	0.77	0.54

Table 11: Comparison of Best Models

Wijaya achieves a higher precision than our best model by 4%, and it shows improvement in both mAP50 and mAP50-95. We can trace this to the fact that the contributor has a much better devices and ran two GPUs in parallel, which was able to run 20 more epochs, with doubled

batch size while all other parameters are similar. Going back to Figure 11, we can see that we don't have a big overfitting at 100 epochs, and this suggest that there is still place for improvement.

6 Conclusions

Ultralytics YOLO family proved to be powerful models which can learn and adapt relatively quickly. For a small dataset like the one we use we were able to achieve a detection precision of 60% and increase it to 75% after fine tuning. The library is user friendly, equipped with many tools to calculate metrics and export plots simply by changing only one argument. For our small dataset, the nano version 8 suited us best, while for bigger datasets, larger models should be investigated.

On the other hand, the models have high computational requirements, with a calculated GPU RAM requirements around 4 GB, that is if we wanted a descent precision, not mentioning cache and environment requirements, making it is not accessible to everyone. Jupyter Notebook crashed multiple times before we were able to run it, due to the way the library is built where it does not clear the RAM cache while training (kajal1598, 2023). The batch size was reduced to 8 images per batch so that we reduce RAM usage as much as possible, otherwise the kernel would shut down after 10-20 epochs.

Future aspects of this project include attempting to secure a better device to train faster and make better parameters search, as well as increasing the size of the dataset. Another direction is to use the trained models at hand and attempt to make complimentary software that can calculate the actual dimensions of the defects and bad welding to make the very base of providing inputs for welding robotic arms.

In summary, YOLO is powerful tool, but not advisable for personal use. Companies can take advantage of these open-source models, to make major advancement. YOLO can be used in the intersection of AI and mechanical engineering, as it opens the doors for innovation.

7 References

Aliy, M. (2022 , Nov 29). *Stackoverflow*. Retrieved from What does "background" FN and FP in the confusion matrix of YOLOv7 stand for: https://stackoverflow.com/questions/74016101/what-does-background-fn-and-fp-in-the-confusion-matrix-of-yolov7-stand-for

Junsung Bae, S.-H. A.-H.-W. (2024). A Study on the Effects of Offset and Welding Speed on Bead Shape and Welding Defects in the FCA Fillet Welding Process. *Journal of Welding and Joining*. doi:10.5781/JWJ.2024.42.5.11

kajal1598. (2023, Dec 14). *Memory is not being properly released after making predictions in YOLOv8. #6981*. Retrieved from GitHub: https://github.com/ultralytics/ultralytics/issues/6981

Lakshmanan, V., Görner, M., & Gillard, R. (2021). *Practical Machine Learning for Computer Vision.* USA: O'Reilly Media, Inc.

Singh, S. (2024, Sep 20). *10 Best Image Labeling & Annotation Tools*. Retrieved from Labellerr: https://www.labellerr.com/blog/top-image-labeling-tools/

Ultralytics Inc. . (2024). *Ultralytics YOLO Docs*. Retrieved from Ultralytics: https://docs.ultralytics.com/

Wijaya, S. A. (2024, 06). *Welding Defect - Object Detection*. Retrieved 11 2024, from Kaggle: https://www.kaggle.com/datasets/sukmaadhiwijaya/welding-defect-object-detection/data

Yue Zhang, Q. Z. (2024). Welding defects recognition based on DCP-MobileViT network. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-024-02500-5