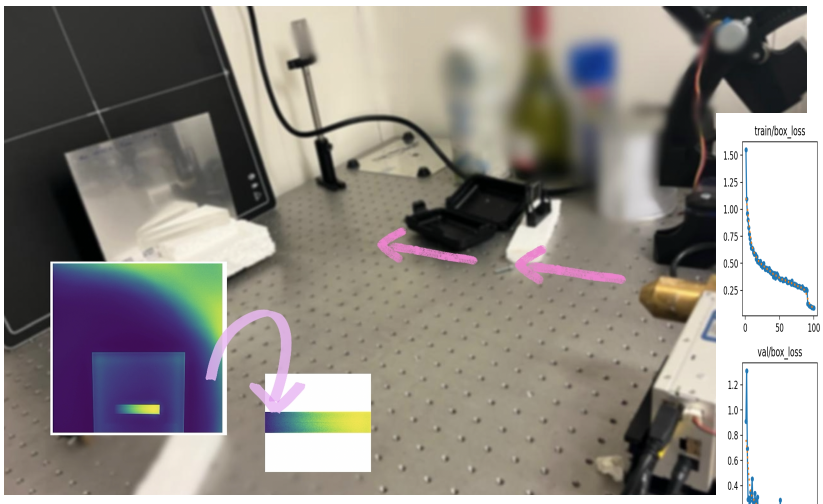
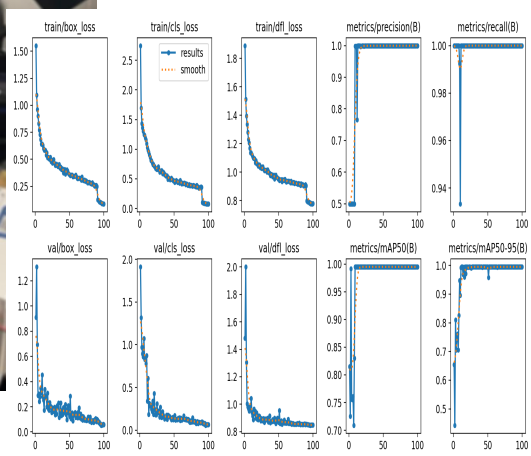
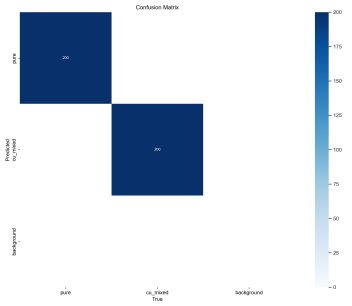
Scientific Project I - EL060A- "Advanced Materials Characterization Using X-ray Imaging and YOLOv8: A Scientific Exploration"

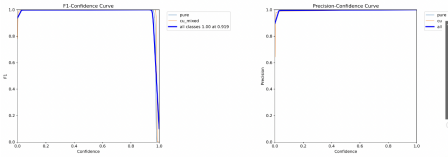


(a) The diagram of "The experimental setup in the laboratory with x-ray tube and flat panel detector.".



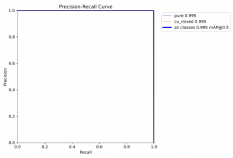
(b) Loss curves



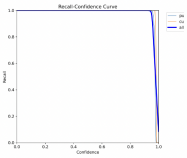


(a) The diagram of "F1 confidence curve".

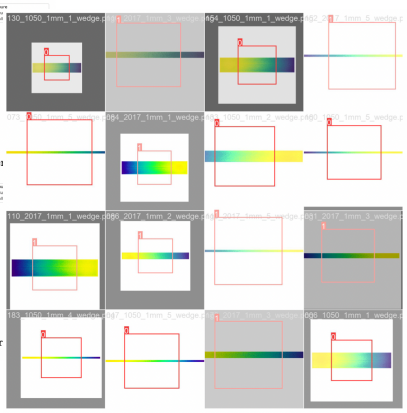
(b) The diagram of "Precision confide



(c) The diagram of "Precision-Recall curve".



(d) The diagram of "Recall cur



(c) Precision-Recall curve

Figure 1: Performance of YOLOv8n

Chethana Johans
(noch2300@student.miun.se)
Mid Sweden University, Sundsvall

Contents

List of Figures

List of Tables

1 Abstract

X-ray-based materials characterization techniques play a critical role in both scientific research and industrial applications. By analyzing how materials absorb X-rays, these techniques provide insight into their electronic and atomic structures, informing assessments of composition, oxidation states, and coordination environments. Although X-ray absorption spectroscopy (XAS)[1] is a powerful method in this domain, it is not applicable in our laboratory setting due to its specialized requirements. Instead, this study employed a more accessible X-ray imaging technique using a laboratory X-ray tube setup to investigate material absorption characteristics.

We examined wedge and samples made of various material compositions, focusing on three thickness levels: 1mm, 2mm, and 5mm. A 5cm long wedge was used in Project 01. The X-ray tube operated at 50keV with a current of 240μA, and each exposure was standardized to 25 seconds to ensure consistent imaging conditions. The goal was to observe how X-ray absorption varies with material type and thickness.

For image classification and interpretation, we implemented deep learning techniques, specifically the YOLO[2] model, due to its high accuracy and real-time processing capability. Training the model on Dataset 5, which included a significantly larger number of images, resulted in markedly improved classification performance.

Although this study did not use XAS directly, we highlight how AI[3] methods particularly deep learning can be leveraged in materials characterization tasks, including automated interpretation of X-ray imaging data. AI techniques contribute to improved efficiency, reduced manual analysis, and improved consistency, offering scalable solutions for research, quality control, and industrial inspection[13, 14].

Keywords: XAS, deep learning, image classification, Material Characterization, Alloys, Wedge Absorption Study, AI, YOLO.

[1] X-ray Absorption Spectroscopy
[2] You Only Look Once
[3] Artificial Intelligence

2 Introduction

In today's technological landscape, real-time image classification is vital across various fields, including healthcare diagnostics, industrial automation, and scientific research. These applications demand systems that can accurately and efficiently process and analyze images in real time, facilitating intelligent decision-making and automation. In materials science, for instance, precise material classification is crucial for quality control, defect detection, and structural analysis. Automated image classification in industrial environments helps identify material inconsistencies, thereby ensuring product reliability. In medical imaging, advanced classification techniques play a key role in detecting anomalies, which enhances early diagnosis and treatment outcomes.

This research utilizes X-ray Absorption Spectroscopy (XAS) as a non-destructive imaging technique to analyze material properties. XAS offers valuable insights into the structural and compositional characteristics of materials, making it an effective tool in both scientific and industrial contexts. When paired with deep learning, XAS-based imaging can significantly improve material classification accuracy through automated image analysis, as recent studies have shown promising results using neural networks and AI to enhance the interpretation of spectroscopic data [28, 9].

To achieve efficient and accurate image classification of X-ray images, this study usesv8, a cutting-edge deep learning model known for its high speed and precision. Although primarily designed for object detection, YOLOv8's flexible architecture also supports image classification, making it well-suited for analyzing X-ray images of wedge-shaped material samples with varying compositions. The model was chosen for its ability to process entire images efficiently, enabling real-time classification based on absorption characteristics, material structure, and composition.

The training data set was systematically prepared by categorizing X-ray images according to material type and wedge thickness, and then divided into training, validation, and testing subsets to evaluate model performance under diverse imaging conditions. Techniques such as data augmentation, hyperparameter tuning, and transfer learning were employed to enhance generalization and robustness. For details on data set preparation and annotation, see Section [4.2] (Dataset preparation), while Chapter 3 provides a comprehensive explanation of the YOLOv8 architecture, implementation, and training methodology[10].

2.1 Contributions

This study presents an AI-driven approach to X-ray-based material classification using YOLOv8, enhancing accuracy and automation in material characterization. The research systematically evaluates the impact of the size of the dataset on the performance of the model by structuring the dataset into training, validation, and testing subsets and progressively smaller divisions. Advanced deep learning techniques, including data enhancement, hyperparameter tuning, and transfer learning, were applied to improve classification accuracy. The model achieved an AP of 95%, demonstrating its effectiveness in distinguishing between pure alloys and Cu composites. By addressing key challenges such as impact on data volume, noise reduction, and real-time classification, this study contributes to the broader application of AI in material science, in-

dustrial automation, and quality control, providing a scalable and efficient solution for X-ray image analysis[32].

3 Background literature

3.1 Architecture of YOLOv8

YOLOv8, like its predecessors, uses a single-stage object detection approach that processes images in one pass through the network. The architecture consists of several key components. There are input, backbone, neck, head and detection layers.

The backbone is responsible for extracting essential features from the input images. YOLOv8 typically uses advanced convolutional neural network (CNN) architectures such as CSPDarknet, which is an improved version of Darknet used in previous YOLO versions. CSPDarknet is designed to improve feature learning while reducing computational cost[24].

The neck further processes the features extracted by the backbone and prepares them for detection. YOLOv8 employs a feature pyramid network (FPN) and path aggregate network (PAN) structure to improve feature representation on different scales. This helps in detecting objects of various sizes. The head is where the actual object detection happens. YOLOv8 uses multiple detection heads to predict bounding boxes and class probabilities on different scales. Each head operates on different feature maps produced by the neck, allowing the model to detect objects of varying sizes more effectively[8].

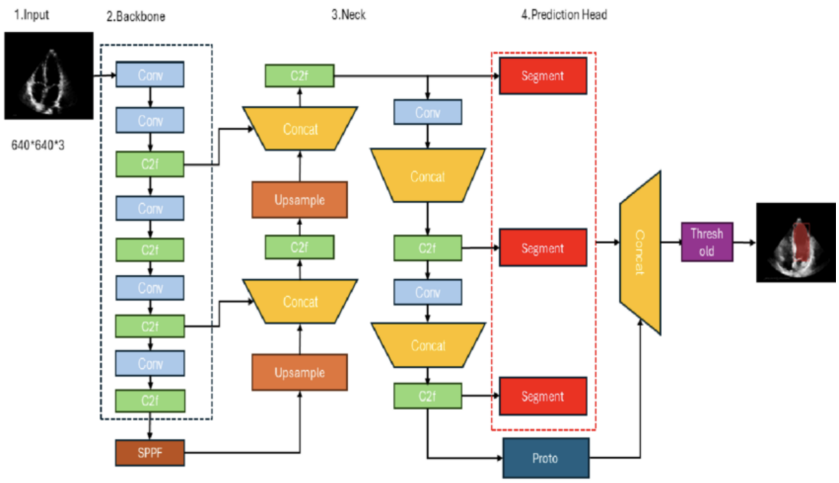


Figure 2: The diagram of "Architecture of Yolo.[2]"

Detection layers generate the final output, which includes bounding box coordinates, objectness scores, and class probabilities. YOLOv8 applies anchor boxes to different grid cells of the feature maps, predicting offsets and confidences for each anchor box[26]. Finally, the results are post-processed with a thresholding step to produce the final segmented and detected objects.

This unified architecture ensures efficient, high-accuracy real-time detection and segmentation suitable for dynamic and complex environments.

3.2 YOLOv8 for Image Classification

YOLOv8, traditionally designed for object detection, has been adapted for image classification due to its efficiency, speed, and versatility. Unlike object detection, which identifies multiple objects and their locations within an image, classification focuses solely on determining the presence of a specific object. In this study, the goal is to classify materials based on X-ray absorption characteristics rather than detect multiple objects. YOLOv8's classification mode is well-suited for this task as it processes images efficiently without the computational overhead of localization.

One of the key advantages of YOLOv8 is its pre-trained classification models, which enable effective training even with smaller datasets. These models utilize deep feature extraction and transfer learning, reducing the need for extensive labeled data while maintaining high accuracy. This makes YOLOv8 particularly useful for X-ray image classification, where collecting large datasets can be challenging. Furthermore, its streamlined architecture ensures fast inference, making it an ideal choice for real-time material classification and spectral analysis. Given that the X-ray images used in this research contain single-sample compositions (pure alloys or Cu mixtures), object detection is unnecessary. By leveraging YOLOv8's classification capabilities, this study ensures high-speed processing and reliable material categorization, optimizing the analysis of X-ray absorption patterns[7].

3.3 YOLO Model Optimization and Evaluation

3.3.1 Hyper parameters in YOLO

In training YOLOv8 models, hyperparameters play a crucial role in optimizing the training of deep learning algorithms. The careful selection of these parameters significantly influences the model's performance, particularly once an appropriate dataset and architecture have been established. Fine-tuning these hyperparameters can lead to substantial improvements in the model's accuracy and efficiency. Hyperparameters can be categorized into several groups based on their functions, such as learning rate parameters, optimizer parameters, augmentation parameters, and model-specific parameters. The learning rate (lr) determines the step size for updating model weights during training. A small value results in slow learning, while a large value can lead to unstable training. Typical values range between 0.001 and 0.01, and the learning rate is often adjusted dynamically to ensure efficient convergence. Common learning rate schedulers include StepLR (which reduces the learning rate at fixed intervals), ExponentialLR (which decays the learning rate exponentially over time), and Cosine AnnealingLR (which adjusts the learning rate following a cosine curve, gradually reducing it towards zero). Optimizer parameters are used to minimize the loss function by adjusting the model's weights. Common optimizers for YOLOv8 include Adam, SGD (Stochastic Gradient Descent), and RMSprop. Key optimizer-related hyperparameters include momentum, which is used with SGD to accelerate gradient vectors in the right direction, typically set to 0.9, and weight decay, which helps

prevent overfitting by penalizing large weights, typically set between 1e-4 and 1e-5. Image augmentation techniques, such as flipping, rotation, scaling, color jitter, and cutout, are used to artificially expand the dataset and improve model generalization. Flipping images horizontally or vertically introduces variability and helps the model handle orientation changes, while rotating images within a specific range (e.g., -30° to +30°) helps the model become invariant to changes in object alignment. Scaling involves resizing images while maintaining their aspect ratio, which helps the model identify objects at varying sizes. Color jitter, which randomly adjusts brightness, contrast, saturation, and hue, helps the model handle changes in lighting and camera settings. Cutout randomly masks out parts of an image during training to force the model to focus on other regions and improve generalization. Model-specific parameters include batch size, the number of samples processed before updating model weights, with typical values ranging from 16 to 64 depending on GPU memory; the number of epochs, or the number of complete passes through the training dataset, typically ranging from 50 to 200; and image size, where higher resolutions like 640x640 or 800x800 improve accuracy but increase computational requirements. Anchor boxes, which are predefined bounding boxes used to predict object locations, are optimized for the dataset to improve detection performance. Regularization parameters such as dropout (which randomly sets a fraction of input units to zero during training to prevent overfitting) and label smoothing (which reduces the model's confidence by adjusting one-hot encoded labels) help reduce overfitting. Training parameters like early stopping, which halts training when a monitored metric stops improving, prevent unnecessary computations and overfitting. YOLOv8 employs a population-based genetic algorithm (GA) for hyperparameter optimization, which can enhance model performance but requires significantly more time and computational resources compared to standard training methods, often necessitating the use of multiple GPUs. While effective, this approach may not be the most cost-effective or time-efficient option, especially for large-scale deployments[24].

3.3.2 Loss function

In YOLOv8, the loss function is designed to optimize three key components: localization (bounding box regression), confidence (objectness), and classification (object class prediction). The overall loss combines these three losses to ensure the model not only detects objects accurately but also classifies them correctly and estimates their bounding boxes precisely[30].

The localization loss measures how well the predicted bounding boxes match the ground truth boxes, typically using the Intersection over Union (IoU) between the predicted and ground truth boxes. Variations of IoU-based loss include Mean Squared Error (MSE), which measures the squared differences between predicted and ground truth coordinates. IoU Loss, which directly optimizes the IoU between the predicted and ground truth boxes. GIoU (Generalized IoU) Loss, which improves IoU by considering the shape and size of the predicted box. DIoU (Distance IoU) Loss, which factors in the distance between the centers of the predicted and ground truth boxes and CIoU (Complete IoU) Loss, which combines aspects of IoU, GIoU, and DIoU, accounting for distance, overlap area, and aspect ratio. The confidence loss evaluates the model's certainty about the presence of an object in the predicted bounding box, typically using Binary Cross-Entropy Loss, which measures the difference between the predicted

confidence score and the ground truth, or Focal Loss, which helps handle class imbalance by down-weighting well-classified examples and focusing on hard-to-classify ones. The classification loss assesses how well the predicted class probabilities match the ground truth labels, generally calculated using Cross-Entropy Loss, which measures the difference between the predicted class probabilities and one-hot encoded ground truth labels, or Focal Loss, which, similar to its use in confidence loss, addresses class imbalance by focusing more on hard to classify examples[29].

The overall loss Function for YOLOv8 is a weighted sum of the three components mentioned above. The formula for the combined loss can be represented as follows.

Total Loss=λloc(Localization Loss) + λconf (Confidence Loss) + λcls(Classification Loss)

3.3.3 Overfitting and underfitting

Overfitting and underfitting are two common problems in machine learning and data analysis that affect the performance of a model.

Overfitting occurs when a model learns the training data too well, capturing noise and details that do not generalize to new, unseen data. This results in high accuracy on the training set but poor performance on validation or test data, with a noticeable gap between training and validation performance. Overfitting is typically caused by overly complex models with too many parameters, training for too many epochs, or having insufficient training data. Solutions include simplifying the model, using regularization techniques like L1 or L2 to penalize large coefficients, and applying early stopping to halt training when validation performance declines. Additionally, cross-validation can help ensure generalization, while data augmentation and ensemble methods can increase data diversity and reduce overfitting[11].

Underfitting, on the other hand, occurs when a model is too simple to capture the underlying patterns in the data, leading to poor performance on both the training and validation/test sets. This issue is characterized by low accuracy and a small gap between training and validation performance. Underfitting often arises from using overly simplistic models, insufficient training time, or choosing an incorrect model architecture. To address underfitting, increasing model complexity by adding more parameters or layers, performing feature engineering to add relevant information, and training the model for more epochs can help. Hyperparameter tuning, such as adjusting learning rate or batch size, and improving data quality to better represent the problem domain can also enhance the model's ability to learn effectively[1]

3.3.4 Metrics Used in YOLOv8

The performance of YOLOv8 is evaluated using key metrics commonly used in both image classification and object detection tasks. Precision measures the accuracy of positive predictions and is calculated as the ratio of true positives (TP) to the sum of true positives and false positives (FP). Recall measures the model's ability to identify all relevant objects, calculated as the ratio of true positives to the sum of true positives and false negatives (FN). These metrics rely on the confusion matrix, which consists of four key components. There are,

True Positives (TP) are instances where the model correctly predicts the positive class.

False Positives (FP) are instances where the model incorrectly predicts the positive class.

False Negatives (FN) are instances where the model incorrectly predicts the negative class.

True Negatives (TN) are instances where the model correctly predicts the negative class. From these values, several performance metrics can be derived as shown below.

- Precision- It measures the accuracy of positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

- Recall (Sensitivity or True Positive Rate)- It measures the ability of the model to identify all positive instances.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

- Accuracy- It measures the overall correctness of the model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

- F1 Score- It is the harmonic mean of precision and recall, providing a single metric that balances the two.

$$F1score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

- Specificity (True Negative Rate)- It measures the ability of the model to identify all negative instances.

$$Specificity = \frac{TN}{TN + FN} \tag{5}$$

In addition to these classification-based metrics, YOLOv8 utilizes Mean Average Precision (mAP) as the primary evaluation metric for object detection. mAP is the mean of Average Precision (AP) scores across all classes, where AP represents the area under the precision-recall curve at different confidence thresholds. Another crucial metric is Intersection over Union (IoU), which measures the overlap between the predicted and ground truth bounding boxes. An IoU threshold (e.g., 0.5) is commonly used to determine whether a detection is classified as a true positive or false positive[3]. In YOLO models, the confidence score can be calculated using the equation below,

$$Confidence = P * (Object) * IOU \tag{6}$$

Here, P(object) is the probability of an object being present, and IOU (Intersection Over Union).

In image classification, the model predicts class labels for an input image, whereas in object detection, the model also predicts bounding box coordinates. However, both tasks rely on similar evaluation metrics for assessing prediction quality and model performance.

4 Materials and Methods- Training YOLOv8 for Image Classification

4.1 Dataset Collection Setup

X-ray Absorption Spectroscopy (XAS) is a powerful technique for investigating the local structure and electronic properties of materials by measuring how X-ray absorption varies with energy [15, 21]. It is especially effective in analyzing metallic alloys at the atomic level. However, XAS typically requires access to synchrotron radiation sources and highly sensitive, energy-resolving detectors, which are beyond the capabilities of our current laboratory setup.

In this study, instead of performing XAS, we utilize a more accessible X-ray transmission imaging approach [17]. Our setup captures the intensity of X-rays transmitted through alloy plates and wedge-shaped filters of varying thicknesses. This allows us to study material-dependent X-ray attenuation properties, which are influenced by composition and thickness. By analyzing these transmitted intensities, we infer material characteristics indirectly, focusing on absorption behavior rather than atomic-scale structural data.

Analysis of Material Attenuation Using X-ray Imaging and Deep Learning:

- An X-ray tube with fixed energy settings (50 keV, 240 μA).
- A flat panel detector used to capture transmitted X-ray intensity through alloy samples and wedge-shaped filters.
- Image processing and analysis performed using Python-based tools and deep learning frameworks such as PyTorch and YOLOv8 for material classification.

In the experimental setup, the X-ray tube operates at a peak energy of 50keV, generating X-rays that excite core electrons in the alloy sample. The sample is placed between the X-ray tube and the flat panel detector. The detector captures transmitted X-rays, providing spatially resolved measurements of absorption behavior.

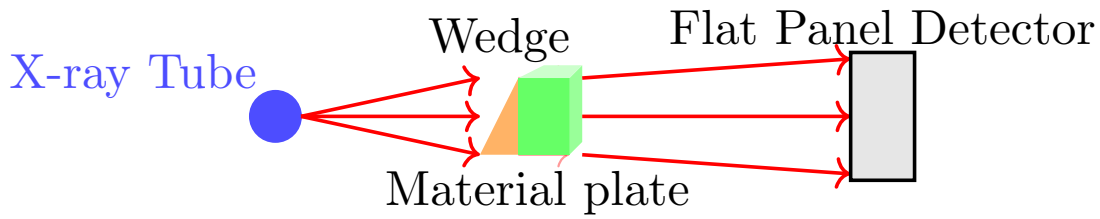


Figure 3: Experimental setup: X-rays pass through a triangle wedge and directly into a material plate before reaching the flat panel detector.

Experimental Procedure:

The X-ray tube and flat panel detector were first calibrated to ensure accurate spatial alignment and consistent intensity measurement. Calibration involved setting the tube voltage

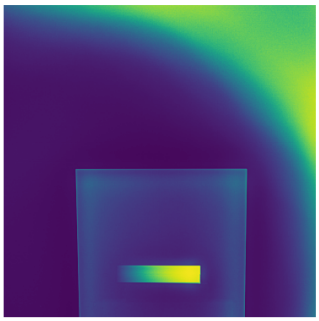
and current to stable, reproducible values and confirming that the detector output corresponded accurately to known reference intensities using a standard sample.

To investigate the absorption behaviour near elemental absorption edges, the X-ray tube voltage was incrementally varied over a range of values that adjusted the spectral distribution of emitted X-rays. Although the flat panel detector does not record energy-resolved spectra, it captures transmitted X-ray intensity, and by scanning across different tube voltages, we inferred relative absorption changes as a function of energy. This method enables qualitative analysis of the behaviour of the absorption edge for elements within the sample.

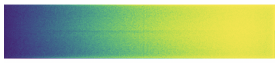
To assess spatial variations in absorption, transmitted intensity measurements were acquired at multiple sample positions. The sample was scanned laterally so that different microstructural regions were placed in the beam path. These included areas previously identified (via microscopy or contrast analysis) as representing different phases (chemically or structurally distinct regions), grain boundaries (interfaces between crystallographic grains), and defect sites (such as voids, dislocations, or inclusions). In the X-ray transmission images, these features appear as regions of varying intensity, phases exhibit contrast due to differences in elemental composition or density; grain boundaries may appear as fine lines or subtle gradients; and defect sites manifest as local intensity irregularities or discontinuities. The spatial resolution of the detector, along with energy-dependent absorption contrast, enabled us to distinguish these features in the recorded intensity profiles.

4.2 Dataset Preparation

The experimental datasets have been organized into training (80%), validation (10%), and testing (10%) sets. Python code for the process has been mentioned in B.2. Each dataset was categorized into two classes, pure and cu_mix. To improve model accuracy, a multi-level dataset generation strategy was adopted. From each original image, the central wedge region was cropped to a fixed size of 800×800 pixels and saved as an image in the Dataset 1 folder. This cropped wedge was then further divided into smaller regions by splitting it horizontally, while maintaining the same image size (800×800), and saved accordingly in separate folders are Dataset 2 (half of the middle wedge), Dataset 3 (quarter), Dataset 4 (eighth) and Dataset 5 (tenth). For image annotation, tools such as LabelImg, Labelbox, and Roboflow were considered. Ultimately, LabelImg was chosen due to its ease of use and compatibility with YOLO annotation format. A data.yaml configuration file was then created to specify the paths for the training, validation, and testing datasets, along with the two defined class labels[22].



(a) Original Image for pure 5mm.



(b) Cropped middle wedge with a fixed size of 800×800 pixels, saved as an image in the Dataset 1 folder.

Figure 4: Experimental results for a selected data sample.

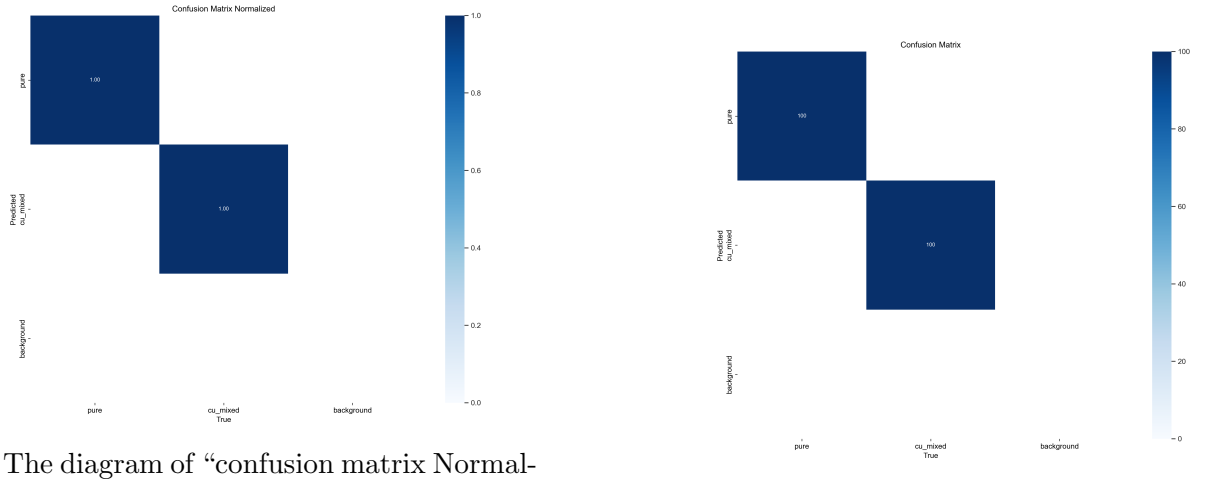
Key training parameters included 150 epochs, an image size of 800×800, a batch size of 8, and one worker for data loading. The trained model was validated using the val method and saved as yolov8n-trained.pt, with Python code referenced in B.6 Additionally, a YOLOv8 configuration file (yolov8.yaml) was created, specifying a Darknet backbone with 53 layers, a YOLO head for single-class detection with 9 predefined anchor boxes, and training settings such as a learning rate of 0.001, momentum of 0.9, and weight decay of 0.0005. The dataset configuration was referenced externally via data.yaml for flexibility with the python code B.5. Experimental images and corresponding annotation files (.txt format) were organized into directories, with any non-YOLO format annotations converted accordingly (Python code referenced in B.1)[16].

5 Results analysis

This section presents the performance evaluation of the YOLOv8 model on classified X-ray absorption images across different material thicknesses (1mm, 2mm, and 5mm), using evaluation metrics include confusion matrices, precision-recall (PR) curves, and training-validation loss curves. To assess the model's robustness and generalization, different thickness levels were tested individually and in combination. The 1mm dataset led to excessive overfitting, as evident from the unstable loss curves. The 5mm thickness showed the best overall results, with high performance metrics, but still exhibited signs of overfitting. To mitigate these issues, datasets were combined specifically, 1mm with 2mm, and 2mm with 5mm. However, the combined 1mm and 2mm dataset did not yield the expected improvement and still suffered from instability. In contrast, the 2mm and 5mm combination performed better, though some overfitting persisted. Among all, the 5mm dataset alone provided the best performance despite the risk of overfitting.

5.1 The confusion matrices

The confusion matrix is a visual representation of the model's performance after training, specifically showing how well the model classifies instances of each class. The confusion matrices (Figure 5) delineate the performance of YOLOv8 models on an object classification task across two classes: "Pure" and "Cu_mixed". The true labels are presented in rows, whereas the predicted labels are illustrated in columns. Overall, the YOLOv8n (nano) models demonstrate analogous performance, each displaying an average precision of around 1.000. The values in the matrix are normalized, meaning they are expressed as proportions of the total predictions for each class, ranging from 0 to 1. A value of 1 indicates perfect prediction accuracy for that class[6].



(a) The diagram of "confusion matrix Normalized".

(b) The diagram of "confusion matrix".

Figure 5: Performance of YOLOv8n- Confusion matrices for Dataset_1.

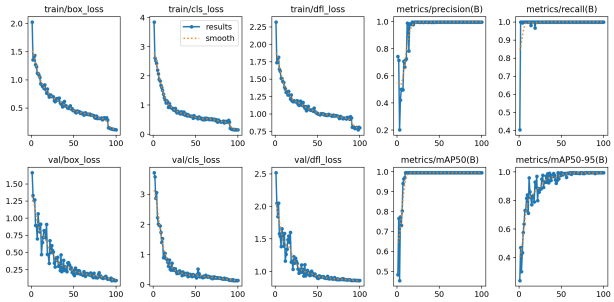
The second confusion matrix gave us additional information on the model's performance. Unlike the normalized confusion matrix, this one shows the actual counts of instances for each

class. For pure aluminium 1050, the model predicted all 100 instances of the pure class correctly. This is indicated by the cell at (pure) having a count of 100, for Cu_mixed, the model predicted all 100 instances of the cu-mixed class correctly. Since the background count is zero, all input images are consequently assigned to one of the two classes, "Pure" or"CU". Mis-classifications are reflected as false positives or false negatives between these two classes. The absence of background detections simplifies the interpretation of the confusion matrix, focusing solely on inter-class prediction errors.

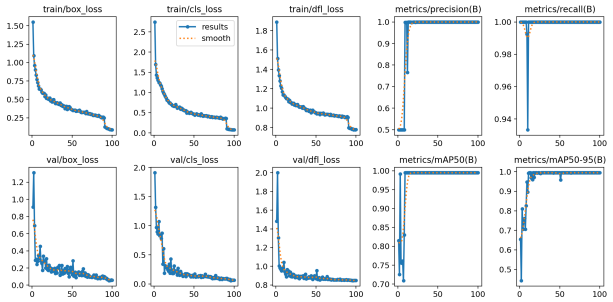
5.2 Results for 5mm thickness

5.2.1 Training and Validation Loss Curves

The YOLOv8 model addresses detection and classification tasks by optimizing three primary loss components, Box Loss, Class Loss (Cls_loss), and Distribution Focal Loss (Dfl_loss). Box Loss minimizes errors in the positioning and sizing of predicted bounding boxes, Class Loss improves the correct categorization of detected objects, and Distribution Focal Loss enhances the model's focus on difficult or ambiguous predictions. Together, these components contribute to more accurate and robust model performance. These images show the training and validation



(a) Loss curve for Dataset_1 (Less images=200).



(b) Loss curve for Dataset_5 (2000 images).

Figure 6: Selected 'Training and Validation Loss Curves vs no of Epochs', Datasets 1 and 5.

loss curves for YOLOv8n across two datasets, Dataset 1 (200 images from the original cropped middle wedge) and Dataset 5 (2000 images, created by expanding Dataset 1 by a factor of 10). A larger dataset results in more stable model performance, with reduced fluctuation in loss curves. The horizontal axis represents the number of epochs (100), while the vertical axis shows the loss value, where lower values indicate better model predictions.

In Dataset 1, training losses such as box loss and classification loss decrease smoothly box loss, for instance, drops from around 2,0 to below 0.3 indicating effective learning. However, large fluctuations in validation losses and metrics (e.g. mAP50-95 varying between 0.5 and 0.95) suggest mild overfitting and instability [27]. In contrast, Dataset 5 demonstrates improved stability, with training losses steadily declining and precision and recall rising consistently toward 1.0. Despite this, occasional spikes in validation losses, such as sudden increases in classification loss, indicate that overfitting risks persist, although to a lesser extent [4].

5.2.2 Comparison of Evaluation Metrics for Different Datasets at 5_mm Thickness

Dataset	Class	Precision (P)	Recall (R)	mAP@50	mAP@50:95
Dataset 1-200 images	All	0.498	1.000	0.985	0.919
	Pure	0.498	1.000	0.975	0.881
	cu_mixed	0.497	1.000	0.995	0.957
Dataset 2-400 images	All	0.499	1.000	0.995	0.995
	Pure	0.499	1.000	0.995	0.995
	cu_mixed	0.498	1.000	0.995	0.995
Dataset 3-800 images	All	0.999	1.000	0.995	0.995
	Pure	0.999	1.000	0.995	0.995
	cu_mixed	0.999	1.000	0.995	0.995
Dataset 4-1600 images	All	1.000	1.000	0.995	0.995
	Pure	0.999	1.000	0.995	0.995
	cu_mixed	1.000	1.000	0.995	0.995
Dataset 5- 2000 images	All	1.000	1.000	0.995	0.995
	Pure	1.000	1.000	0.995	0.995
	cu_mixed	1.000	1.000	0.995	0.995

Table 1: Evaluation Metrics Comparison for 5mm Thickness Across Different Datasets.

Table 1 presents a comparative evaluation of five datasets, each containing both pure and cu mixed classes. The performance metrics, Precision (P), Recall (R), mAP@50, and mAP@50:95 are remain consistently high across datasets 3,4 and 5, indicating robust model perfor- mance. Notably, Dataset 5 achieves better scores for mAP@50, highlighting optimal detection capabilities. The uniformity of the results across class types demonstrates that the model generalizes well regardless of the composition of the dataset.

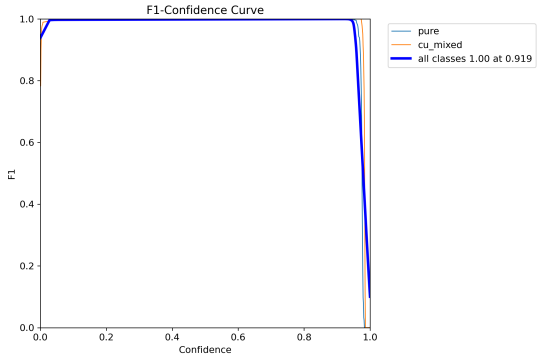
5.2.3 Precision–Recall Curves for Dataset_5 at 5mm Thickness

The model demonstrates exceptional performance across all evaluated metrics. For both the "Pure" and "Cu-mixed" classes, precision values of 0.998 and 0.997, respectively, indicate that nearly all predicted instances were correct, with very few false positives. Recall scores of 1.000 for both classes confirm that all actual instances were successfully identified, with no false negatives.

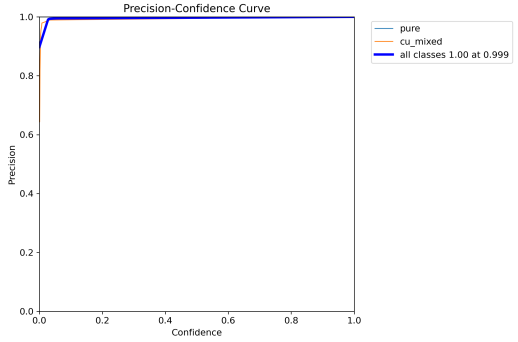
The mean Average Precision (mAP) at an IoU threshold of 0.5 (mAP@50) is 0.995, reflecting highly accurate detections for both classes. Additionally, the mAP@50-95, which averages precision over multiple IoU thresholds (0.5 to 0.95), also scores 0.995, suggesting the model maintains strong performance across varying levels of localization strictness[23].

The F1-Confidence Curve further supports this robustness. The combined F1 score peaks at 1.00 when the confidence threshold is set to approximately 0.919. This balance of precision and recall across thresholds confirms the model's stability and effectiveness.

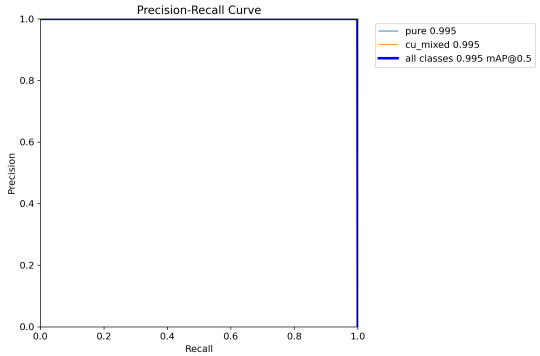
Overall, the consistently high metrics across all evaluations along with insights from the confusion matrix and F1 analysis indicate that the model is both well-fitted to the training



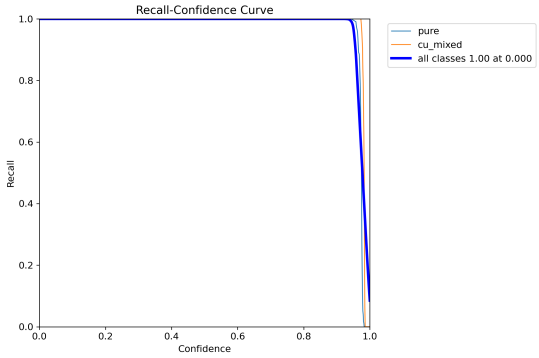
(a) The diagram of "F1 confidence curve".



(b) The diagram of "Precision confidence curve".



(c) The diagram of "Precision-Recall curve".



(d) The diagram of "Recall curve".

Figure 7: The PR curve of YOLOv8n- 5mm thickness for Dataset_5

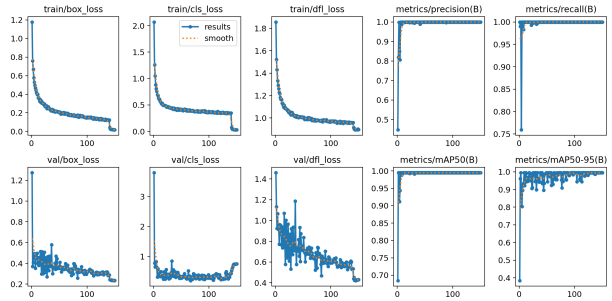
data and generalizes well[32].

5.3 Results for 1mm and 2mm thickness

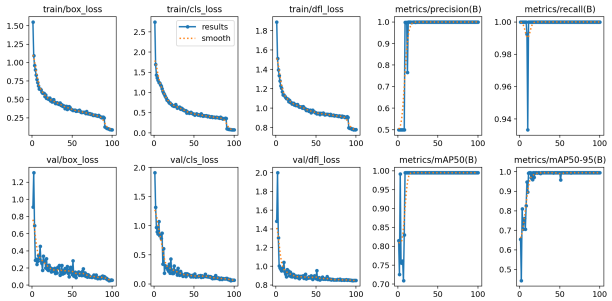
For the 1mm thickness, the YOLOv8n model was trained on all datasets, with Dataset_5 (which has more images than the others) showing better performance, as shown in the below diagram8. The training losses (box_loss, cls_loss, dfl_loss) showed a steady decrease, indicating effective learning. However, Dataset_5 exhibited signs of overfitting, with fluctuations in the validation loss curves. While the training loss converged more smoothly, the validation loss fluctuated, suggesting that the model was memorizing specific patterns rather than generalizing well to unseen data. Box_loss remained the most stable among the loss components, but overfitting was still apparent, highlighting the need for further adjustments, such as regularization or additional data augmentation, to improve generalization.

For the 2mm thickness, similar trends were observed. Dataset_5 showed a steady decrease in loss values, but overfitting persisted, as evidenced by fluctuations in the validation loss curves. The performance metrics, including precision and recall, remained high, but mAP50-95 was slightly noisier, indicating some instability in generalization. While Dataset_5 performed

better than smaller datasets, the overfitting issue persisted, suggesting that more data or further model improvements would be beneficial for more stable and reliable results.



(a) Loss curve for 1mm- Dataset_5(2000 Images).

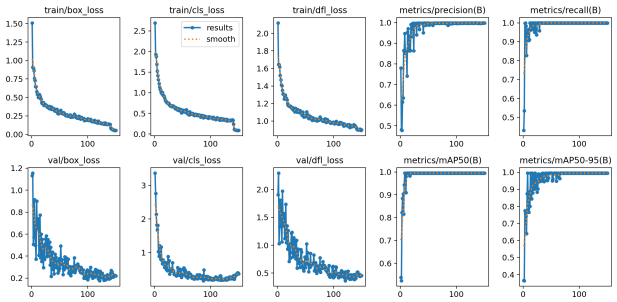


(b) Loss curve for 2mm Dataset_5(2000 images).

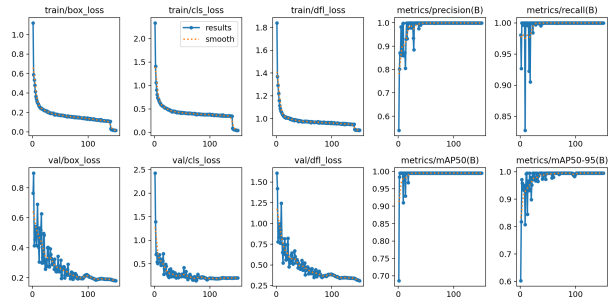
Figure 8: Best performance of YOLOv8n - Training and Validation Loss Curves for 1mm and 2mm thickness

5.4 Results for combining 2mm and 5mm thickness

5.4.1 Training and Validation Loss Curves



(a) Loss curve for Dataset_1(Less images).



(b) Loss curve for Dataset_5(More images).

Figure 9: Performance of YOLOv8n - Training and Validation Loss Curves for Dataset_1 and Dataset_5.

These images represent the training and validation loss curves for YOLOv8n on Dataset 1 and Dataset 5, using a combined dataset with both 2mm and 5mm wedge sizes. The loss curves (box_loss, cls_loss, dfl_loss) show a downward trend as the number of epochs increases. This indicates that the model is learning effectively and the loss is decreasing, which is expected in a well-trained model. The training losses (top row) and validation losses (bottom row) decrease steadily and stabilize towards the later epochs.

Comparison Between Dataset_1 and Dataset_5- Both datasets exhibit similar trends in terms of loss reduction, but the initial loss values and the stabilization points differ slightly. Dataset_1 shows slightly higher variation in some of the validation loss metrics compared to Dataset_5,

which suggests a potential difference in data complexity or distribution. The precision and recall graphs show rapid improvement at the start and then stabilize close to 1.0, which indicates high detection accuracy. The mAP50 and mAP50-95 curves also show a rising trend, meaning that the model's ability to detect objects improves over epochs.

5.4.2 Comparison of Evaluation Metrics Across Datasets for Combined 2mm and 5mm Thicknesses

Dataset	Class	P	R	mAP@50	mAP@50-95
Dataset_1	All	0.999	1	0.995	0.995
	Pure	0.998	1	0.995	0.995
	cu_mixed	1	1	0.995	0.995
Dataset_2	All	0.999	1	0.995	0.995
	Pure	0.999	1	0.995	0.995
	cu_mixed	0.999	1	0.995	0.995
Dataset_3	All	1	1	0.995	0.995
	Pure	1	1	0.995	0.995
	cu_mixed	1	1	0.995	0.995
Dataset_4	All	1	1	0.995	0.995
	Pure	1	1	0.995	0.995
	cu_mixed	1	1	0.995	0.995

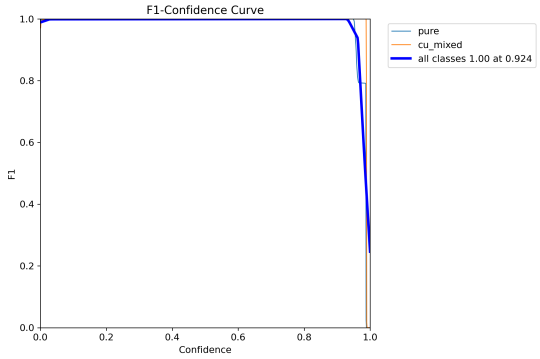
Table 2: Performance metrics comparison across datasets for combining 2mm and 5mm thickness.

The performance evaluation across four datasets demonstrates consistently high values for Precision (P), Recall (R), mAP@50, and mAP@50-95. Precision ranges from 0.998 to 1.000, while Recall is consistently 1.0 across all datasets and class types. Similarly, both mAP@50 and mAP@50-95 maintain a uniform value of 0.995 throughout. These results indicate that the model performs with exceptional accuracy and generalizes well across different dataset compositions without any degradation in performance.

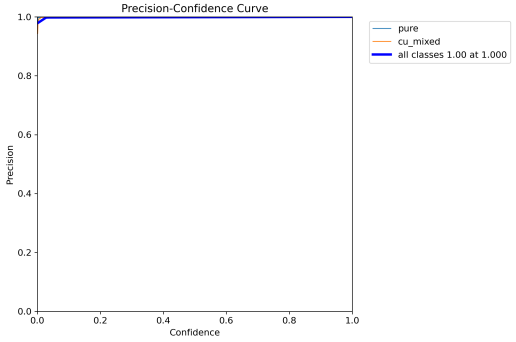
5.4.3 Precision–Recall Curves for Dataset 5 combining 2mm and 5mm thickness

The model demonstrates exceptional performance across all evaluated metrics. For both the "Pure" and "Cu-mixed" classes, precision values of 0.995 and 0.995, respectively, indicate that nearly all predicted instances were correct, with very few false positives. Recall scores of 1.000 for both classes confirm that all actual instances were successfully identified, with no false negatives.

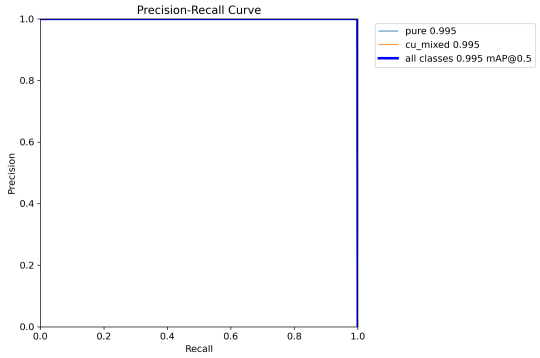
The mean Average Precision (mAP) at an IoU threshold of 0.5 (mAP@50) is 0.995, reflecting highly accurate detections for both classes. Additionally, the mAP@50-95, which averages precision over multiple IoU thresholds (0.5 to 0.95), also scores 0.995, suggesting the model maintains strong performance across varying levels of localization strictness.



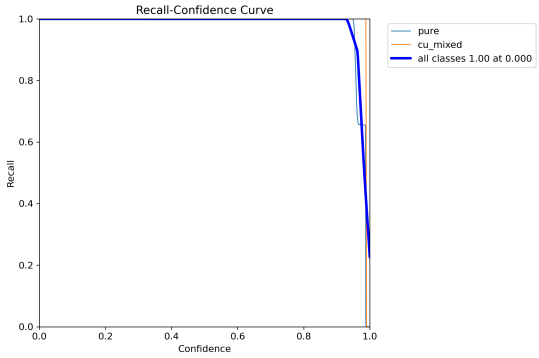
(a) The diagram of "F1 confidence curve".



(b) The diagram of "Precision confidence curve".



(c) The diagram of "Precision-Recall curve".



(d) The diagram of "Recall curve".

Figure 10: The PR curve of YOLOv8n- 2mm and 5mm

The F1-Confidence Curve further supports this robustness. The combined F1 score peaks at 1.00 when the confidence threshold is set to approximately 0.924. This balance of precision and recall across thresholds confirms the model's stability and effectiveness. Overall, high metrics across all evaluations along with insights from the confusion matrix and F1 analysis indicate that the model is both well-fitted to the training data and generalizes well[32].

5.5 Summary of "Training and Validation Loss Curves for YOLOv8n"

The training and validation loss curves clearly indicate that the best performance was achieved using the 5mm thickness dataset. Initially, the model was trained on the 1mm dataset, which resulted in the poorest performance across all experiments. Training with the 2mm thickness yielded better results than the 1mm dataset, but still fell short of expectations. In contrast, the 5mm dataset produced significantly improved outcomes in terms of training stability and validation accuracy[20].

To explore the impact of data combination, the 1mm and 2mm datasets were merged and used for training. However, the resulting performance was not satisfactory. On the other hand, combining the 2mm and 5mm datasets led to noticeable improvements, though the loss curves

still did not surpass those from training exclusively on the 5mm dataset.

These results suggest that increased material thickness and larger wedge sizes contribute to more accurate and stable training outcomes. Among all dataset configurations, the 5mm thickness dataset and the combined 2mm and 5mm dataset performed best, exhibiting high precision, recall, and mAP metrics. While combining datasets appeared to stabilize the training process, some fluctuations remained, indicating that further tuning particularly focused on validation consistency could lead to even better model performance[33].

Best Performing Dataset for Each Thickness Level

Table 3: Performance Metrics for Each Thickness Using Dataset 5 (More Image Samples via Finer Splitting)

Thickness (mm)	**Precision (P)**	**Recall (R)**	**mAP@50**	**mAP@50-95**
1 mm	0.675	0.890	0.884	0.884
2 mm	0.870	1.000	0.920	0.995
5 mm	1.000	1.000	1.000	0.996
1 mm + 2 mm	0.799	1.000	0.984	0.984
2 mm + 5 mm	1.000	1.000	0.995	0.995

As seen in Table 3 the dataset for 5mm thickness achieved the highest overall performance. Although combining data sets (e.g., 2mm + 5mm) improved generalization, the data set of only 5mm consistently produced the most stable and accurate results.

6 Final Conclusion and Improvements

6.1 Conclusion

The training and validation loss curves, along with the metrics comparison tables, provide meaningful insights into the performance and generalizability of the YOLOv8n model across datasets consisting of pure and Cu-mixed materials. The results demonstrate that the model performs strongly in accurately classifying both material types across varying dataset conditions and thickness levels (1mm, 2mm, and 5mm).

Among all datasets, those corresponding to 5mm thickness, as well as the combined 2mm and 5mm datasets, yielded the best performance, showing high precision, recall and mean average precision (mAP). The inclusion of additional data in the combined dataset may have contributed to more stable training; however, some fluctuations in performance still remain. Further hyperparameter tuning and regularization could help improve validation stability.

Overall, the near-perfect precision, recall, and mAP scores across all datasets indicate that the model is highly reliable in detecting and distinguishing between the two material classes. The consistent performance across datasets suggests that the model is robust to minor variations in input data, such as those caused by different material thicknesses. This implies that the model effectively learns generalizable features rather than relying on specific, dataset-dependent characteristics [12].

The smooth and steadily declining loss curves further reinforce the conclusion that the training process was well-executed. The absence of erratic fluctuations or divergence in these curves indicates that the model converged efficiently. Additionally, the minimal gap between training and validation losses reflects the model's strong generalization capabilities, ensuring its performance extends beyond the training data to unseen test samples.

The uniformity of results across datasets and metrics highlights the reliability of the data preparation and augmentation strategies, as well as the YOLOv8n architecture's suitability for this classification task. Collectively, these findings confirm that the approach taken in this study successfully addresses the classification problem for pure and Cu-mixed materials and provides a solid foundation for deploying such models in practical applications.

This robust performance across different datasets underscores the potential of the model to generalize further in real-world conditions, making it a promising candidate for tasks requiring precise and consistent material classification[25].

6.2 Future Work and Improvements

Future research can explore several key areas to enhance the performance, generalizability, and real-world applicability of the YOLOv8n model. One important direction is to expand dataset diversity by deliberately introducing controlled variations beyond existing noise — such as synthetic occlusions, changes in lighting conditions, motion blur, and different object orientations or positions. These enhancements would more accurately simulate real-world variability and help evaluate the model's robustness under less controlled conditions. Additionally, incorporating data from different imaging devices or acquisition settings could further test the

model's adaptability. Finally, deploying the model in operational environments will provide valuable feedback on its performance outside of the laboratory context and highlight practical constraints or needed optimizations [5].

Optimizing model complexity presents another valuable direction. Evaluating lighter or more efficient versions of YOLO may reduce inference time and computational costs, making the system more suitable for resource constrained applications. Complementary to this, incorporating advanced data augmentation techniques like geometric transformations, adversarial examples, and synthetic data generation could improve generalization on unseen data[10].

To improve interpretability, techniques such as Grad-CAM or SHAP should be integrated. These tools offer visual explanations of the model's predictions, increasing trust and transparency especially critical in industrial and scientific settings. Hyperparameter tuning, including exploration of learning rates, batch sizes, and regularization methods, remains vital for maximizing performance. The use of cross-validation will further ensure stable results across varying data splits[19].

Building on the foundation of this project, future work may transition from binary to multi-class classification. In the second phase of the project, X-ray imaging of samples may be performed at 50 keV and 300μA. To improve spectral resolution, the wedge size may be increased from 5cm to 7cm, with continued focus on 1mm, 2mm, and 5mm thick samples. These adjustments enable a more detailed investigation of absorption characteristics and material differentiation. This study expands the material analysis to five aluminum-based alloys. There are pure aluminum (Al 1050), aluminum-copper (Al 2017), aluminum-magnesium (Al 5083), aluminum-silicon-magnesium (Al 6082), and aluminum-zinc-magnesium (Al 7075). The inclusion of multiple aluminum-based alloys and the use of CNNs combined with a Tailed Method provided effective solutions for long-tailed data distributions. Adopting techniques such as class rebalancing, data enhancement, or weighted loss functions could help improve the model's ability to accurately classify material types represented. This would enhance overall classification performance and allow the model to better distinguish between materials with subtle differences. Furthermore, integrating confidence threshold calibration using F1 confidence curves could help balance precision and recall dynamically.

In general, these enhancements, ranging from architectural improvements to deeper understanding of the model, will strengthen the ability of the model to function as a reliable, scalable, and explainable tool for automated material classification in diverse domains.

References

[1] Anil Kumar Prajapati Anil and Umesh Kumar Singh. An optimal solution to the overfitting and underfitting problem of healthcare machine learning models. *Journal of Systems Engineering and Information Technology (JOSEIT)*, 2(2):77–84, 2023.

[2] Madankumar Balasubramani, Chih-Wei Sung, Mu-Yang Hsieh, Edward Huang, Jiann-Shing Shieh, and Maysam Abbod. Automated left ventricle segmentation in echocardiography using yolo: A deep learning approach for enhanced cardiac function assessment, 05 2024.

[3] Alibek Barlybayev, Nurzada Amangeldy, Bekbolat Kurmetbek, Iurii Krak, Bibigul Razakhova, Nazira Tursynova, and Rakhila Turebayeva. Personal protective equipment detection using yolov8 architecture on object detection benchmark datasets: a comparative study. *Cogent Engineering*, 11(1):2333209, 2024.

[4] Jonathan T Barron. A general and adaptive robust loss function. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4331–4339, 2019.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[6] Olivier Caelen. A bayesian interpretation of the confusion matrix. *Annals of Mathematics and Artificial Intelligence*, 81(3):429–450, 2017.

[7] Siyu Chen, Yixuan Li, Yidong Zhang, Yifan Yang, and Xiangxue Zhang. Soft x-ray image recognition and classification of maize seed cracks based on image enhancement and optimized yolov8 model. *Computers and Electronics in Agriculture*, 216:108475, 2024.

[8] Tausif Diwan, G Anirudh, and Jitendra V Tembhurne. Object detection using yolo: Challenges, architectural successors, datasets and applications. *multimedia Tools and Applications*, 82(6):9243–9275, 2023.

[9] Mehrdad Haghighatlari, Nikolai Hüsing, and et al. Learning to learn from data: Information extraction from x-ray absorption spectra using unsupervised and supervised machine learning. *npj Computational Materials*, 6:24, 2020.

[10] Muhammad Hussain. Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, 11(7):677, 2023.

[11] David D Jensen and Paul R Cohen. Multiple comparisons in induction algorithms. *Machine Learning*, 38:309–338, 2000.

[12] Ping Jiang, Aolin Qi, Jiao Zhong, Yahui Luo, Wenwu Hu, Yixin Shi, and Tianyu Liu. Field cabbage detection and positioning system based on improved yolov8n. *Plant Methods*, 20(1):96, 2024.

[13] Ahmet Karaman, Ishak Pacal, Alper Basturk, Bahriye Akay, Ufuk Nalbantoglu, Seymanur Coskun, Omur Sahin, and Dervis Karaboga. Robust real-time polyp detection system design based on yolo algorithms by optimizing activation functions and hyper-parameters with artificial bee colony (abc). *Expert systems with applications*, 221:119741, 2023.

[14] Didem Ketenoglu. A general overview and comparative interpretation on element-specific x-ray spectroscopy techniques: Xps, xas, and xrs. *X-Ray Spectrometry*, 51(5-6):422–443, 2022.

[15] D.C. Koningsberger and R. Prins. *X-ray Absorption: Principles, Applications, Techniques of EXAFS, SEXAFS and XANES*. Wiley-Interscience, 1988.

[16] Varsha Kshirsagar, Raghavendra Hemant Bhalerao, and Manish Chaturvedi. Modified yolo module for efficient object tracking in a video. *IEEE Latin America Transactions*, 21(3):389–398, 2023.

[17] S. Mayo, F. Chen, and D. Paganin. X-ray imaging for materials science. *Annual Review of Materials Research*, 51:91–114, 2021.

[18] Lillian C McDermott, Mark L Rosenquist, and Emily H Van Zee. Student difficulties in connecting graphs and physics: Examples. *Am. J. Phys*, 55(6):6, 1987.

[19] Shunqi Mei, Yishan Shi, Heng Gao, and Li Tang. Research on fabric defect detection algorithm based on improved yolov8n algorithm. *Electronics*, 13(11):2009, 2024.

[20] Anastasiia Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. *Advances in neural information processing systems*, 30, 2017.

[21] M. Newville. Fundamentals of xafs. *Reviews in Mineralogy and Geochemistry*, 78(1):33–74, 2014.

[22] Chibuzo Joseph Nnonyelu, Meng Jiang, Marianthi Adamopoulou, and Jan Lundgren. A machine- learning -based approach to direction-of-arrival sectorization using spherical microphone array. In *2024 IEEE 13rd Sensor Array and Multichannel Signal Processing Workshop (SAM)*, pages 1–5, 2024.

[23] Mohamad Azfar Mohamad Rastari, Rosniza Roslan, Raseeda Hamzah, Noor Hasimah Ibrahim Teo, Fadilah Ezlina Shahbudin, and Khyrina Airin Fariza Abu Samah. Recycle waste detection and classification model using yolo-v8 for real-time waste management. In *2024 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, pages 372–377. IEEE, 2024.

[24] Mukaram Safaldin, Nizar Zaghden, and Mahmoud Mejdoub. An improved yolov8 to detect moving objects. *IEEE Access*, 12:59782–59806, 2024.

[25] Carlos Santos, Marilton Aguiar, Daniel Welfer, and Bruno Belloni. A new approach for detecting fundus lesions using image processing and deep neural network architecture based on yolo model. *Sensors*, 22(17):6441, 2022.

[26] Mupparaju Sohan, Thotakura Sai Ram, Rami Reddy, and Ch Venkata. A review on yolov8 and its advancements. In *International Conference on Data Intelligence and Cognitive Informatics*, pages 529–545. Springer, 2024.

[27] Guoguang Tan, Yongsheng Ye, Jiawei Chu, Qiang Liu, Li Xu, Bin Wen, and Lili Li. Real-time detection method of intelligent classification and defect of transmission line insulator based on lightweight-yolov8n network. *Journal of Real-Time Image Processing*, 22(2):53, 2025.

[28] Janis Timoshenko, Deyu Lu, Yuewei Lin, and Anatoly I Frenkel. Supervised machine-learning-based determination of three-dimensional structure of metallic nanoparticles. *The journal of physical chemistry letters*, 8(20):5091–5098, 2017.

[29] Remco van der Meer, Cornelis W Oosterlee, and Anastasia Borovykh. Optimally weighted loss functions for solving pdes with neural networks. *Journal of Computational and Applied Mathematics*, 405:113887, 2022.

[30] Tom Viering and Marco Loog. The shape of learning curves: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(6):7799–7819, 2022.

[31] Mingjie Wang and Fuzhong Li. Real-time accurate apple detection based on improved yolov8n in complex natural environments. *Plants*, 14(3):365, 2025.

[32] Fan Zhang, Fangtao Ren, Jieping Li, and Xinhong Zhang. Automatic stomata recognition and measurement based on improved yolo deep learning model and entropy rate superpixel algorithm. *Ecological Informatics*, 68:101521, 2022.

[33] Tian Zhang, Pengfei Pan, Jie Zhang, and Xiaochen Zhang. Steel surface defect detection algorithm based on improved yolov8n. *Applied Sciences*, 14(12):5325, 2024.

Appendices

A YOLO-Model

A.1 Image Classification:

The labels “pure” and “cu_mixed” are correctly assigned to the different samples. The image below suggests that the dataset contains two classes, “pure” and “cu_mixed”. The alternating labels indicate that the model is distinguishing between these two classes. All images in the grid appear to have bounding boxes that encompass the objects. The bounding boxes are clearly visible, with the labels in red, indicating the class to which the object within the box belongs. The labeling seems consistent with the visual data. The model or labling process has correctly assigned the “pure” and “cu_mixed” labels according to the visible features (color strips). The model (or the ground truth) is correctly classifying the objects as either “pure” or “cu_mixed”,” as evidenced by the labels on the bounding boxes. This suggests that the model has learned to differentiate between the two classes fairly well. The appearance of both “pure” and “cu_mixed” labels in the grid suggests that the dataset contains a balanced number of samples from each class, which is beneficial for training.

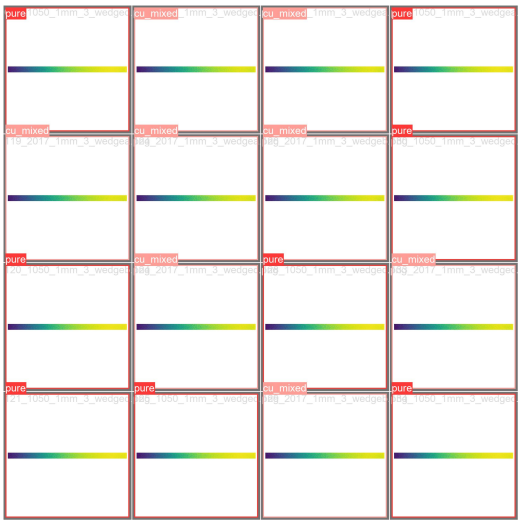
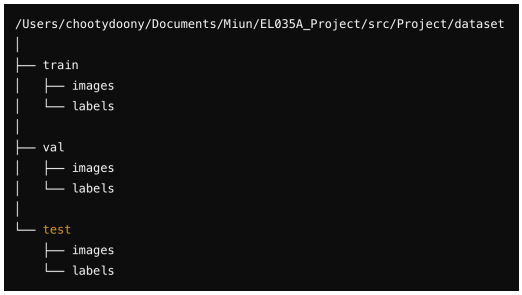


Figure 11: The diagram of “classifying images labelled as “pure” and “cu_mixed”

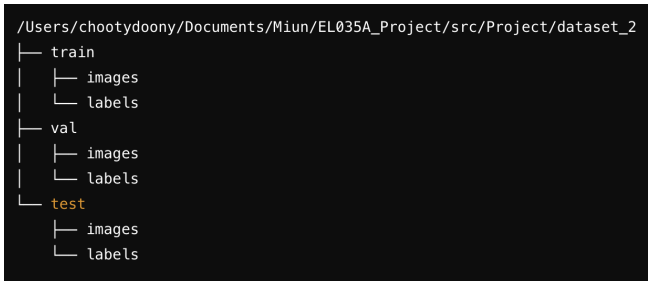
A.2 Organising Data- Images:

Experimental images and their corresponding annotation files have been placed in directories. Each image has been paired with a corresponding .txt file containing the annotations.

When annotations have been in a different format, they have been converted to the YOLO format. The experimental image datasets have been converted to their own labels (.txt files). Python code for this process has been mentioned in B.1.



(a) Organized my experimental dataset for original dataset



(b) Organized my experimental dataset for dataset-2

Figure 12: Organized my experimental dataset

A.2.1 Data Configuration (data.yaml):

The data.yaml file has been created to specify the paths to the training and validation images and annotations. This file has also defined the number of classes and their names.

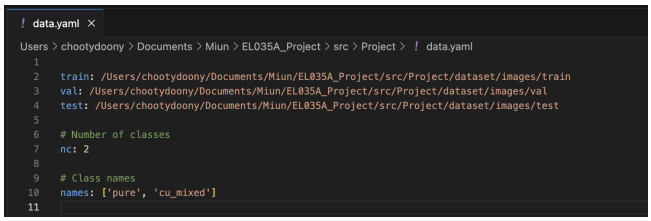


Figure 13: The diagram of "data.yaml structure"

A.2.2 Model Configuration (yolov8.yaml):

A configuration file specific to YOLOv8 has been prepared (if required by the implementation), detailing parameters such as model architecture, training settings, and augmentation strategies. Python code for the yolov8n.yaml file has been mentioned in B.5.

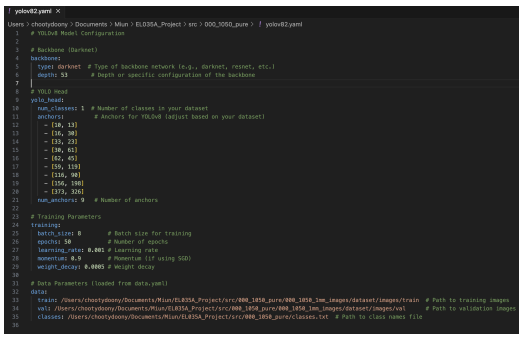


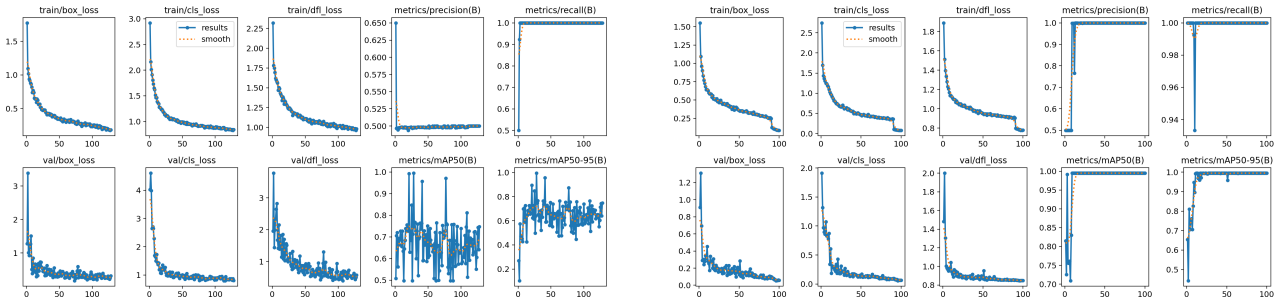
Figure 14: The diagram of "yolov8.yaml structure"

Preprocess Data

Data Augmentation: Data augmentation techniques have been implemented to increase the diversity of the training dataset. Common techniques such as random cropping, flipping, rotation, and color adjustments have been applied. Normalization: Images have been normalized (typically by dividing pixel values by 255) to facilitate model training.

A.3 Results for 1mm thickness

A.3.1 Training and Validation Loss Curves



(a) Loss curve for Dataset_1(Less Images). (b) Loss curve for Dataset_5(More Images).

Figure 15: Performance of YOLOv8n - Training and Validation Loss Curves for Dataset_1 and Dataset_5..

These images show training and validation loss curves for YOLOv8n for two datasets, Dataset_1 (original dataset with 300 images) and Dataset_5 (the tenth dataset mean 300*10 images). From this can concluded that more images means less fluctuations. The horizontal axis of the graphs had represented the number of epochs(here 150), each epoch being a complete iteration through the training dataset. More epochs had generally led to better model performance, but excessive training could have resulted in overfitting. The vertical axis had indicated the loss value, which measured how well the model had performed during training. A lower loss value had signified better accuracy, while a higher loss had suggested poor predictions. The goal had been to minimize the loss over the course of training.

Both datasets exhibit a decreasing trend in loss curves (box_loss, cls_loss, dfl_loss) over epochs, indicating that the model is learning effectively. The validation loss follows a similar trend but with some fluctuations, suggesting generalization but potential overfitting in some cases.

The precision and recall graphs show high values, especially for Dataset_5, suggesting good performance. The mAP50 and mAP50-95 metrics exhibit different behaviours, Dataset_1 has more fluctuations in mAP50-95. Dataset_5 shows a more stable mAP curve, possibly indicating better consistency. Dataset_5 appears to have slightly lower initial losses and converges more smoothly. Dataset_1 shows more variance in its mAP50-95, which may indicate inconsistencies in performance across different classes or bounding box sizes. Overall, Dataset_5 might be yielding better stability and convergence, while Dataset_1 shows more fluctuations, possibly due to dataset complexity or class imbalance[31].

Dataset	Class	P	R	mAP@50	mAP@50-95
Dataset_1	All	0.488	1	0.752	0.752
	Pure	0.498	1	0.766	0.766
	cu_mixed	0.500	1	0.537	0.537
Dataset_2	All	0.499	1	0.835	0.823
	Pure	0.500	1	0.812	0.789
	cu_mixed	0.498	1	0.858	0.858
Dataset_3	All	0.500	1	0.709	0.709
	Pure	0.500	1	0.709	0.732
	cu_mixed	0.500	1	0.686	0.686
Dataset_4	All	0.799	1	0.884	0.884
	Pure	0.674	1	0.882	0.883
	cu_mixed	0.806	1	0.881	0.884

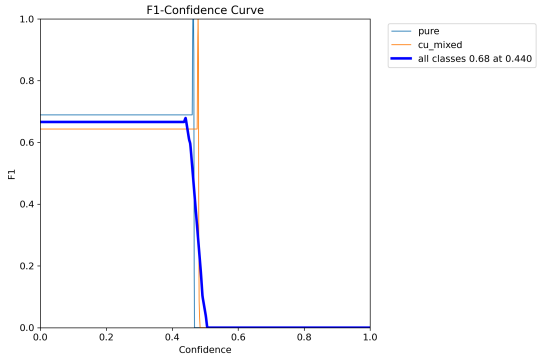
Table 4: Performance metrics comparison across datasets for 1mm thickness.

A.3.2 Comparison of Evaluation Metrics for 1_mm Thickness

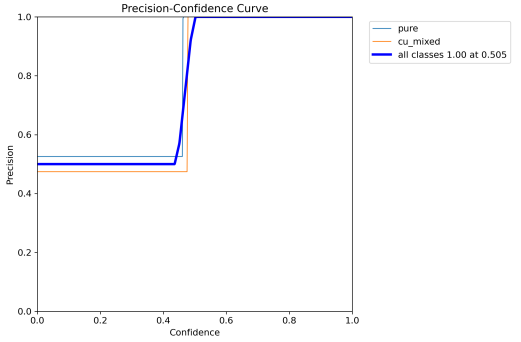
The table summarizes the model's performance across four datasets containing both Pure and cu mixed classes. Dataset 1 and Dataset 3 show slightly lower precision (around 0.488–0.5), Datasets 2 and 4 demonstrate significantly improved performance, with Dataset 4 achieving the highest scores reaching up to 0.806 precision and 0.881 mAP@50. This suggests the model performs better on Dataset 4 and maintains good generalization across different class types and datasets.

A.3.3 The PR curves for 1mm thickness

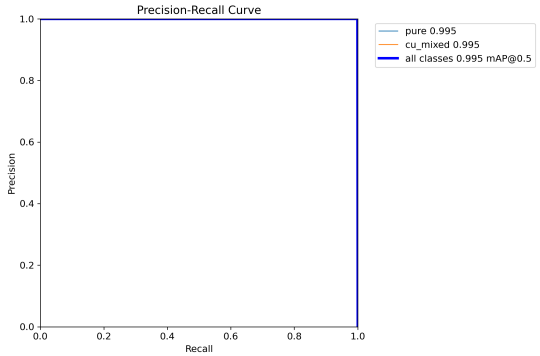
The model does not shows strong performance for all PR curves on this dataset, But with mAP@0.5 reaching 0.995 across all classes. Precision values for the "Pure" and "Cu-mixed" classes are 0.995 respectively, indicating good accuracy in predictions. Recall remains perfect at 1.000 for both classes, ensuring all relevant objects are detected. The Precision-Recall curve (Figure 7.c) reflects a more gradual decline, suggesting consistent but slightly varied performance across confidence thresholds. The F1-Confidence Curve peaks at 0.68 when the confidence threshold is around 0.404, highlighting an effective balance between precision and recall. The Precision-Confidence Curve shows perfect precision at a threshold of 0.05, while the Recall-Confidence Curve maintains high recall across lower confidence levels at 0.5. Overall, the model remains reliable with room for further optimization.



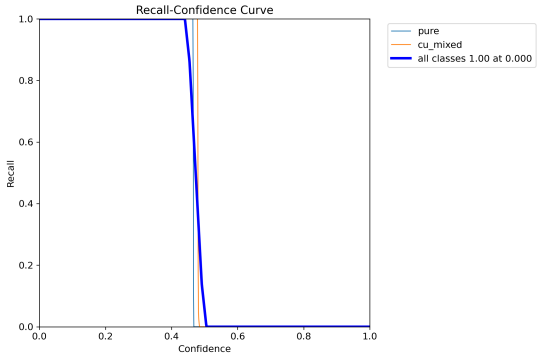
(a) The diagram of "F1 confidence curve".



(b) The diagram of "Precision confidence curve".



(c) The diagram of "Precision-Recall curve".



(d) The diagram of "Recall curve".

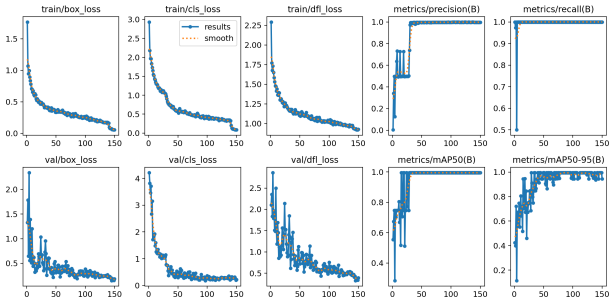
Figure 16: The PR curve of YOLOv8n- 1mm thickness

A.4 Results for 2mm thickness

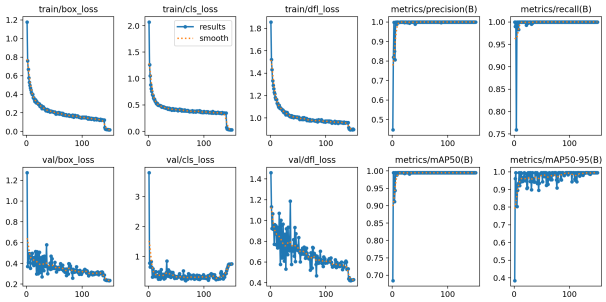
A.4.1 Training and Validation Loss Curves

These images show training and validation loss curves for YOLOv8n on two datasets (Dataset_1 and Dataset_5) for the 2mm-wedge size scenario. Training and Validation Losses: Both datasets show a steady decrease in loss values (box_loss, cls_loss, dfl_loss), meaning the model is learning effectively. The validation loss exhibits some fluctuations, but overall, it follows the decreasing trend of training loss. Dataset_5 has lower initial losses and appears to converge more smoothly, while Dataset_1 shows higher loss values and more fluctuations, especially in the validation curves. Performance Metrics (Precision, Recall, mAP50, mAP50-95): Precision and recall reach high values quickly, suggesting strong model performance. mAP50 is stable for both datasets, indicating good detection accuracy. mAP50_95 is noisier for Dataset_1, suggesting potential issues with generalization or class imbalance. Dataset_5 appears more stable and consistent in the evaluation metrics, while Dataset_1 shows more variance, especially in the later epochs.

Comparison to the 1mm Wedge Size Results (Previous Image Set): The loss values and metric trends are similar, but Dataset_5 consistently shows better stability and lower losses. Increasing wedge size from 1mm to 2mm does not significantly alter the trends, but it is important to check whether



(a) Loss curve for Dataset_1(Less images).



(b) Loss curve for Dataset_5(More images).

Figure 17: Performance of YOLOv8n - Training and Validation Loss Curves for Dataset_1 and Dataset_5.

the object size affects detection performance. That is why I have checked with 5mm wedge size as well. Dataset_5 performs more consistently with smoother convergence and stable evaluation metrics. Dataset_1 shows higher variability, possibly due to dataset differences, class imbalance, or increased complexity. The model learns effectively in both cases, but Dataset_5 might be preferable for more reliable results[18].

B Scripts made for this project

B.1 Python code for Convert data file to annotations(txt files):

```
import os
import xml.etree.ElementTree as ET

def convert(size, box):
    dw = 1. / size[0]
    dh = 1. / size[1]
    x = (box[0] + box[1]) / 2.0 - 1
    y = (box[2] + box[3]) / 2.0 - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh
    return (x, y, w, h)

def convert_annotation(image_id):
    in_file = open(f'Annotations/{image_id}.xml')
    out_file = open(f'labels/{image_id}.txt', 'w')
    tree = ET.parse(in_file)
```

```
    root = tree.getroot()
    size = root.find('size')
    w = int(size.find('width').text)
    h = int(size.find('height').text)

    for obj in root.iter('object'):
        difficult = obj.find('difficult').text
        cls = obj.find('name').text
        if cls not in classes or int(difficult) == 1:
            continue
        cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        b = (float(xmlbox.find('xmin').text), float(xmlbox.find('xmax').text), float
        (xmlbox.find('ymin').text), float(xmlbox.find('ymax').text))
        bb = convert((w, h), b)
        out_file.write(f"{cls_id} " + " ".join([str(a) for a in bb]) + '\n')

if __name__ == "__main__":
    if not os.path.exists('labels/'):
        os.makedirs('labels/')
    classes = ["class1", "class2", "class3"]  # Replace with your classes
    for image_id in os.listdir('Annotations'):
        image_id = image_id.split('.')[0]
        convert_annotation(image_id)
```

B.2 Python code for Organized data to train, validation and test files:

```
import os
import shutil
import random

# Define paths
dataset_path = '/Users/chootydoony/Documents/Miun/EL035A_Project/src/alloy_5mm/
Dataset_1050_pure/dataset_20p'
train_folder = os.path.join(dataset_path, 'train')
val_folder = os.path.join(dataset_path, 'val')
test_folder = os.path.join(dataset_path, 'test')

# Create subfolders if they don't exist
```

```
os.makedirs(train_folder, exist_ok=True)
os.makedirs(val_folder, exist_ok=True)
os.makedirs(test_folder, exist_ok=True)

# List all files in dataset_1 folder (ignoring subfolders)
all_files = [f for f in os.listdir(dataset_path) if os.path.isfile(os.path.join(dataset_path

# Shuffle the files for random splitting
random.shuffle(all_files)

# Calculate split sizes
total_files = len(all_files)
train_size = int(0.8 * total_files)
val_size = int(0.1 * total_files)
test_size = total_files - train_size - val_size

# Split the files
train_files = all_files[:train_size]
val_files = all_files[train_size:train_size + val_size]
test_files = all_files[train_size + val_size:]

# Function to move files
def move_files(file_list, destination_folder):
    for file_name in file_list:
        src_file = os.path.join(dataset_path, file_name)
        dst_file = os.path.join(destination_folder, file_name)
        shutil.move(src_file, dst_file)

# Move files to respective folders
move_files(train_files, train_folder)
move_files(val_files, val_folder)
move_files(test_files, test_folder)

print(f"Files have been split into train ({train_size}), val ({val_size}),
and test ({test_size}) folders.")
```

B.3 Python code for Organized data for two classes- Images:

```
import os
import shutil

# Define the base path where your dataset is located
base_path = '/Users/chootydoony/Documents/Miun/EL035A_Project/src/Project/
dataset/images'
```

```
# Define the subdirectories
subdirs = [‘train’, ‘val’, ‘test’]

# Define the class keywords and corresponding directories
class_keywords = {
    ‘pure’: ‘1050’,
    ‘cu_mixed’: ‘2017’
}

# Create the class-specific directories if they don’t exist
for subdir in subdirs:
    for class_name in class_keywords.keys():
        class_dir = os.path.join(base_path, subdir, class_name)
        os.makedirs(class_dir, exist_ok=True)

# Function to move files to appropriate class-specific directories
def move_files_to_class_dirs(subdir):
    subdir_path = os.path.join(base_path, subdir)
    for filename in os.listdir(subdir_path):
        file_path = os.path.join(subdir_path, filename)
        if os.path.isfile(file_path):
            moved = False
            for class_name, keyword in class_keywords.items():
                if keyword in filename:
                    dest_dir = os.path.join(subdir_path, class_name)
                    shutil.move(file_path, os.path.join(dest_dir, filename))
                    moved = True
                    break
            if not moved:
                print(f’Warning: Could not determine class for file {filename}’)

# Move files in each subdirectory
for subdir in subdirs:
    move_files_to_class_dirs(subdir)

print("Dataset has been split into class-specific directories.")
```

B.4 Python code for data.yaml:

```
import os

# Paths to your directories
```

```
train_images_dir = '/Users/chootydoony/Documents/Miun/EL035A_Project/src/alloy_5mm/
Dataset_7/train/images'
val_images_dir = '/Users/chootydoony/Documents/Miun/EL035A_Project/src/alloy_5mm/
Dataset_7/val/images'
test_images_dir = '/Users/chootydoony/Documents/Miun/EL035A_Project/src/alloy_5mm/
Dataset_7/test/images'
data_yaml_path = '/Users/chootydoony/Documents/Miun/EL035A_Project/src/alloy_5mm/
data7.yaml'

# Number of classes
num_classes = 2

# Class names
class_names = ['pure', 'cu_mixed']

# Create the data.yaml content
data_yaml_content = f"""
train: {train_images_dir}
val: {val_images_dir}
test: {test_images_dir}

# Number of classes
nc: {num_classes}

# Class names
names: {class_names}
"""

# Write the data.yaml content to the file
with open(data_yaml_path, 'w') as file:
    file.write(data_yaml_content)

print(f'data1.yaml file has been created at {data_yaml_path}')
```

B.5 Python code for yolov8.yaml:

```
    # YOLOv8 Model Configuration

# Backbone (Darknet)
backbone:
  type: darknet  # Type of backbone network (e.g., darknet, resnet, etc.)
  depth: 53       # Depth or specific configuration of the backbone

# YOLO Head
```

```
yolo_head:
  num_classes: 1  # Number of classes in your dataset
  anchors:         # Anchors for YOLOv8 (adjust based on your dataset)
    - [10, 13]
    - [16, 30]
    - [33, 23]
    - [30, 61]
    - [62, 45]
    - [59, 119]
    - [116, 90]
    - [156, 198]
    - [373, 326]
  num_anchors: 9   # Number of anchors

# Training Parameters
training:
  batch_size: 8        # Batch size for training
  epochs: 150           # Number of epochs
  learning_rate: 0.001 # Learning rate
  momentum: 0.9        # Momentum (if using SGD)
  weight_decay: 0.0005 # Weight decay

# Data Parameters (loaded from data.yaml)
data:
  train: /Users/chootydoony/Documents/Miun/EL035A_Project/src/000_1050_pure/
  000_1050_1mm_images/dataset/images/train  # Path to training images
  val: /Users/chootydoony/Documents/Miun/EL035A_Project/src/
000_1050_pure/000_1050_1mm_images/dataset/images/val  # Path to validation images
  classes: /Users/chootydoony/Documents/Miun/EL035A_Project/src/
  000_1050_pure/classes.txt  # Path to class names file
```

B.6 Python code for train_yolov8n.py:

```
from ultralytics import YOLO

def main():
    # Define the path to the data.yaml file
    data_yaml_path = '/Users/chootydoony/Documents/Miun/EL035A_Project/src/alloy_5mm/
    data1.yaml'

    # Load the YOLOv8 model
    model = YOLO("yolov8n.pt")  # using pre-trained model for better results
```

```
    # Train the model
    model.train(data=data_yaml_path, epochs=150, imgsz=800, batch=4)

    # Validate the model
    model.val()

    # Save the model
    model.save("yolov8n_trained1.pt")

if __name__ == "__main__":
    main()
```