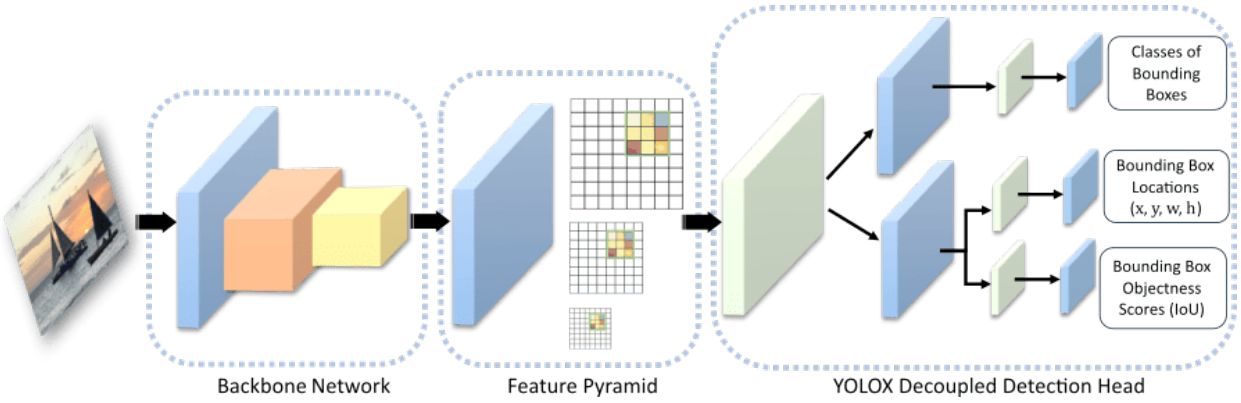
Rail Symbol Detect:
Automated BOM Generation of Railways
Schematics using YOLOX object detector

Introduction

The project aimed to simplify creating Bills of Material (BOM) from railway schematics by using machine learning and image processing. Focusing on making it easier to identify and catalog the electronic equipment and symbols used in railway routes, which are usually done manually. Applying technologies like MATLAB and YOLOX has taken a significant step towards making the process of documenting and planning railway construction more efficient and less time-consuming.

Background on the Algorithms

- **YOLOX**



The YOLOX algorithm enhances object detection with its single-stage, anchor-free approach, offering reduced model size and faster computation.

Unlike previous versions, it identifies objects by locating their centers without predefined anchor boxes. It employs a grid system at three scales for bounding box predictions, adaptable to varying image sizes.

YOLOX comprises three main components:

- CSP-DarkNet-53 backbone for feature extraction,
- Neck combining feature pyramid and path aggregation networks for multi-scale feature mapping.
- Decoupled detection head for classifying objects, predicting locations, and assessing object presence with confidence scores.

- **Image Segmentation**

  The segmentation algorithm divides an input image into smaller segments or tiles, considering overlap between them to ensure comprehensive coverage and analysis. It calculates the number of segments along the X and Y axes based on the image dimensions, desired segment size, and overlap percentage.

  The overlap size in pixels is derived from the segment size multiplied by the overlap ratio.

  Mathematically, the starting and ending coordinates for each segment are determined as follows:

$$\text{startX} = max\left(1, (x-1) \times \left(\text{SegmentSize}_{\text{width}} - \text{OverlapSize}_{\text{width}}\right) + 1\right)$$
$$\text{startY} = max\left(1, (y-1) \times \left(\text{SegmentSize}_{\text{height}} - \text{OverlapSize}_{\text{height}}\right) + 1\right)$$
$$\text{endX} = \min\left(\text{startX} + \text{SegmentSize}_{\text{width}} - 1, \text{imgWidth}\right)$$
$$\text{endY} = \min\left(\text{startY} + \text{SegmentSize}_{\text{height}} - 1, \text{imgHeight}\right)$$

  The number of segments along each dimension is computed as:

$$\text{numSegmentsX} = \left\lceil \frac{\text{imgWidth} - \text{overlapSizeX}}{\text{segmentSizeX} - \text{overlapSizeX}} \right\rceil$$
$$\text{numSegmentsY} = \left\lceil \frac{\text{imgHeight} - \text{overlapSizeY}}{\text{segmentSizeY} - \text{overlapSizeY}} \right\rceil$$

- **Data Augmentation**

  The data augmentation algorithm expands the image datasets through systematic transformations, crucial for improving object detection models' robustness. It applies flipping, rotation, scaling, and zooming to images and their corresponding bounding boxes.

  Mathematically, this involves constructing a transformation matrix for each augmentation.
  For rotation, the matrix is defined as:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  For scaling, the matrix is:

$$S(s) = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  The flip matrix for horizontal and vertical flips is:

$$F(x) = \begin{bmatrix} x & 0 & 0 \\ 0 & -x & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  To ensure a diverse set of augmented images we introduce randomness by selecting a transformation with equal probability, 1/3, and then applying parameters from specific distributions:
  rotation angles (θ) from a uniform distribution within a range (e.g., [−42,42]).
  scaling factors (s) from a uniform distribution (e.g., [0.9,1.1]).
  flip factor (x): $x = -1$ grants horizontal flip whereas $x = 1$ vertical flip.

  The number of images before and after the augmentation with 10 transformations:

Label	Count	ImageCount	Label	Count	ImageCount
AxleCounter	48	36	AxleCounter	548	396
DistanceSignal	29	19	DistanceSignal	308	198
ExitSignalShunting	18	13	ExitSignalShunting	285	198
HomeSignal	20	14	HomeSignal	308	198
PointElectrical	36	24	PointElectrical	429	286
ShuntingSignal	23	16	ShuntingSignal	319	209

- **Modified NMS**

  The classic Non-Maximum Suppression (NMS) algorithm involves selecting the bounding box with the highest score and removing all other boxes that have an Intersection over Union (IoU) value greater than a predefined threshold with the selected box.

  Mathematically, for a set of bounding boxes (B) with scores (S), and an IoU threshold (τ), the process can be summarized as follows:

  1. Select the bounding box (b_i) with the highest score (s_i) from (S).
  2. Compute IoU of (b_i) with all other boxes. Let $IoU(b_i, b_j)$ be the IoU of boxes (b_i) and (b_j).
  3. Remove all boxes (b_j) where $IoU(b_i, b_j) > \tau$.
  4. Repeat steps 1-3 for the next highest-scoring bounding box in the remaining set. This iterative process continues until all boxes are either selected or removed.

  The modified NMS algorithm with averaging adds a refinement step to the classic NMS process by averaging the coordinates of overlapping bounding boxes based on their detection scores.

  For each selected bounding box, B_{sel} $with\ a\ score\ S_{sel}$, and overlapping bounding boxes B_i with scores S_i where $IoU(B_{sel}, B_i) > \tau$, the new bounding box B_{new} is computed as:

  $$B_{new} = \frac{\sum_i (B_i \times S_i)}{\sum_i S_i}$$

  This equation calculates the score-weighted average position of all bounding boxes that overlap with B_{sel} more than the threshold τ, ensuring the final bounding box is optimally positioned based on the confidence scores of the detections.

- **Color Detection**

  The color detection algorithm identifies the predominant color of an image by comparing each pixel's color to predefined color ranges for each potential color name.

  For each color range, it calculates a mask indicating whether each pixel falls within that range. It then counts the number of pixels matching each color and selects the color with the highest pixel count as the predominant one.

  Mathematically, the mask for a given color is computed as:

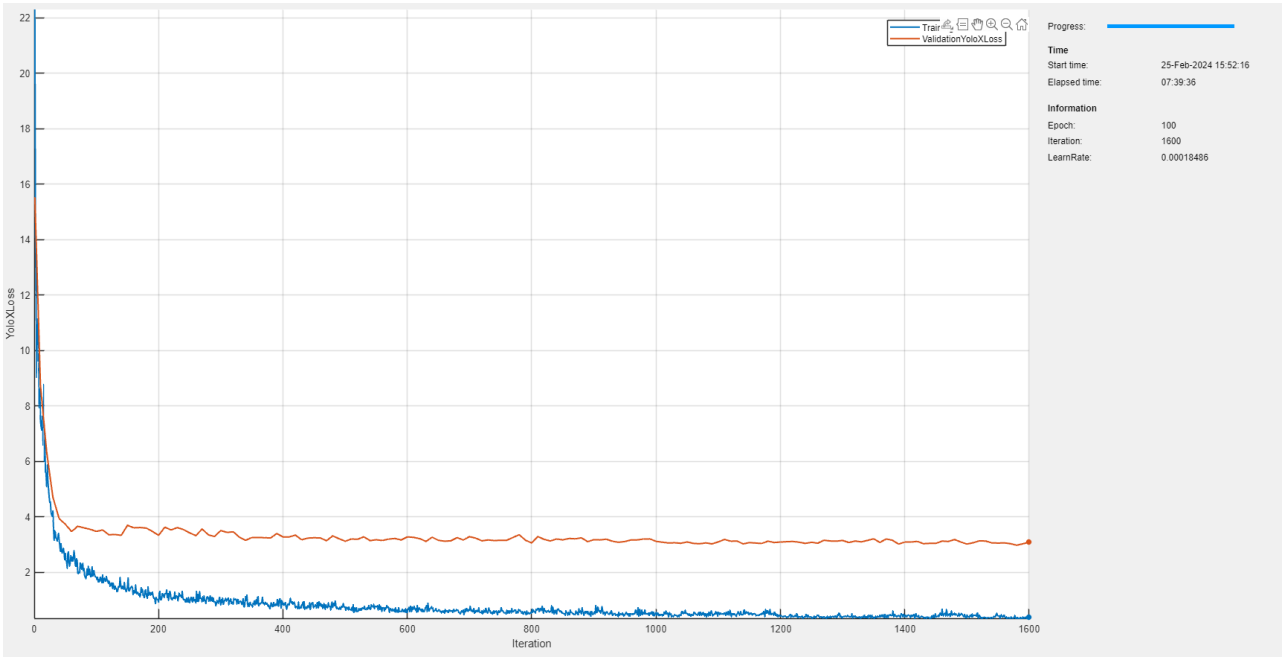
$$\begin{aligned}\text{mask} = &\left(R_{\text{low}} \leq R \leq R_{\text{high}}\right) \wedge \\ &\left(G_{\text{low}} \leq G \leq G_{\text{high}}\right) \wedge \\ &\left(B_{\text{low}} \leq B \leq B_{\text{high}}\right)\end{aligned}$$

  where $(R), (G), and\ (B)$ are the pixel values for red, green, and blue channels, and low and high are the bounds of the color range.

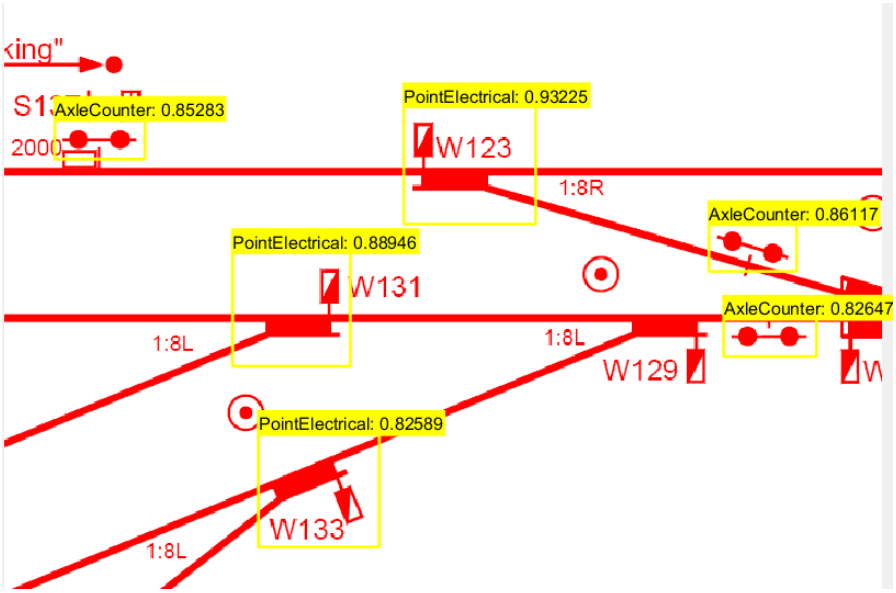
  The algorithm adjusts for background colors by potentially selecting the second most prevalent color if the most prevalent color is deemed to be background (typically white or yellow), based on its dominance in the image.

Examples of Successful and Less Successful Runs

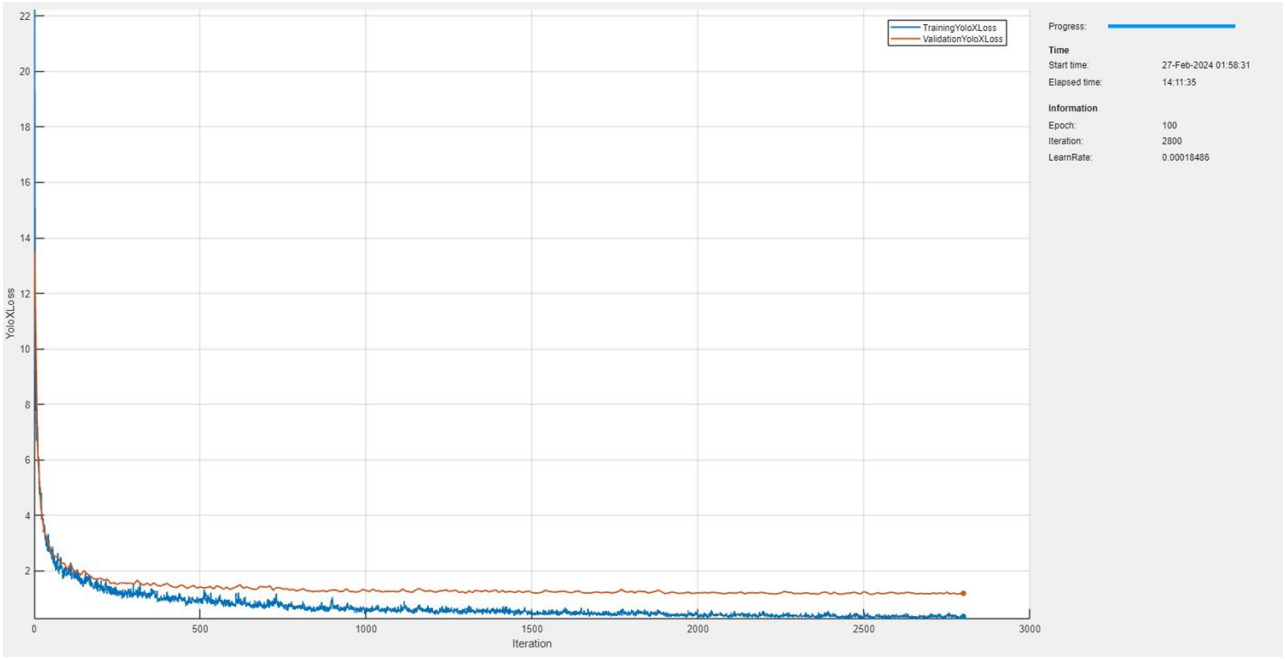
The first model, trained without augmented images, showed promising training loss convergence but struggled with higher validation loss, indicating overfitting. Its detection performance was suboptimal, with low precision scores across the test set, reflecting difficulty in accurately recognizing symbols or doing so with low confidence.



Label	Count	ImageCount	classNames	averagePrecision
AxleCounter	48	36	{'AxleCounter' }	0.85
DistanceSignal	29	19	{'DistanceSignal' }	0.23958
ExitSignalShunting	18	13	{'ExitSignalShunting'}	0
HomeSignal	20	14	{'HomeSignal' }	0.38
PointElectrical	36	24	{'PointElectrical' }	0.66667
ShuntingSignal	23	16	{'ShuntingSignal' }	0.57



The second model, incorporating augmented images, demonstrated improved loss convergence, with both training and validation losses indicating a more balanced fit. This model achieved high precision scores, suggesting effective symbol recognition with good confidence levels, although it wasn't immune to false positives



Label	Count	ImageCount	classNames	averagePrecision
			{'AxleCounter' }	1
AxleCounter	548	396	{'DistanceSignal' }	0.99841
DistanceSignal	308	198	{'ExitSignalShunting' }	0.95241
ExitSignalShunting	285	198	{'HomeSignal' }	1
HomeSignal	308	198	{'PointElectrical' }	0.99983
PointElectrical	429	286	{'ShuntingSignal' }	1
ShuntingSignal	319	209		

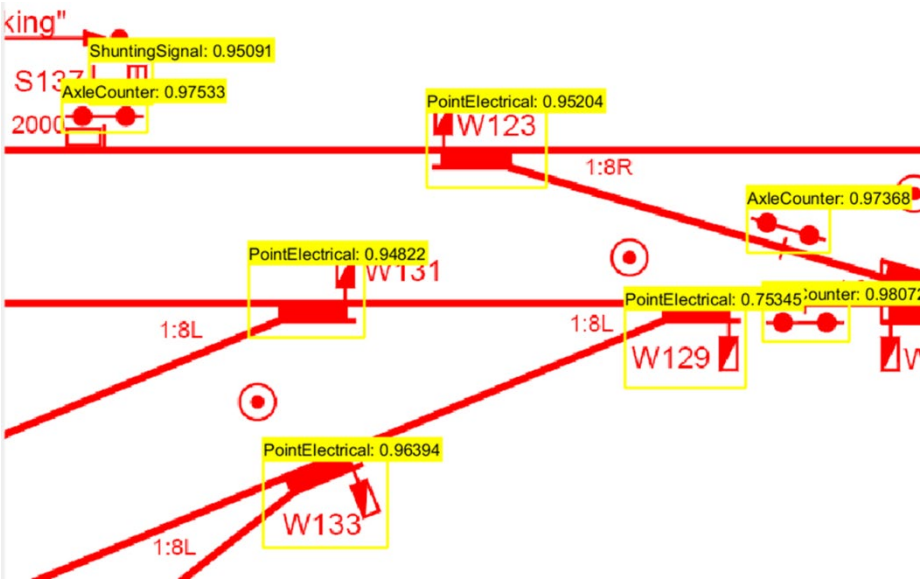


Table of Differences

Feature	Poor Model	Better Model
Data Augmentation	No	Yes
Training Loss	Converges close to zero	converges close to zero
Validation Loss	Stabilizes around 3.5	Stabilizes around 1.5
Average Precision (AP)	Ranges from 0 to 0.85	Between 0.95 to 1
Detection Accuracy	Struggles with recognition; low confidence	Good confidence; some false positives

This comparison underscores the role of addressing data scarcity in machine learning through synthetic data augmentation.

By expanding datasets synthetically, the enhanced model significantly improves in handling validation loss and precision in object detection.

This approach demonstrates the pivotal importance of creating comprehensive training datasets, enabling more effective symbol detection from schematics, and showcasing the power of synthetic data expansion in overcoming limitations posed by insufficient data.

Summary and Conclusion

This project showcased the use of advanced image processing and machine learning to automate generating BOM from railway schematics, highlighting both achievements and areas for improvement.

It highlighted challenges such as detection inconsistencies and limited training scope due to the focus on prevalent symbols, overlooking rare ones. It also pointed out issues with symbol variability, including composite symbols and those absent from the legend, affecting detection precision.

Despite these challenges, the project achieved significant automation in BOM generation, indicating the potential of machine learning in engineering. Future directions include expanding the model's training to encompass a greater variety of symbols and employing additional techniques to improve accuracy, aiming for a more versatile and precise tool for automated documentation.

Appendix: Program Files Summary

- **PDFImageConverter.m**: Converts PDF schematic to images for analysis.
- **ImageSegmenter.m**: Splits large image into smaller segments.
- **SymbolDetector.m**: Detects and identifies symbols within the images.
- **SymbolProcessor.m**: Processes and classifies detected symbols.
- **ExportToExcel.m**: Transfers processed data into an Excel spreadsheet for report.

- **Main.m**: The main script that coordinates the execution of the entire process.

- **YOLOXTrainingScript.m**: Contains the code for training the YOLOX object detection model with the dataset.
- **CreateAugmentedImageDataset.m**: Generates augmented images to expand the training dataset.