# The Use of Artificial Intelligence in the Recognition of Railway Assets Based on High-Resolution Drone Images

CRISTIAN ROBU
(Licenciado)

Relatório de Estágio para obtenção de grau de mestre em
Matemática Aplicada para a Indústria

Orientador(es):
Professor Doutor Engenheiro Carlos José Brás Geraldes
Professor Doutor Filipe Santiago Cal

Júri:
Presidente: Professor Doutor Luís Manuel Ferreira da Silva
Vogais:
Professor Doutor Engenheiro Carlos José Brás Geraldes
Professor Doutor Nuno David de Jesus Lopes

**Julho 2024**

# The Use of Artificial Intelligence in the Recognition of Railway Assets Based on High-Resolution Drone Images

CRISTIAN ROBU
(Licenciado)

Relatório de Estágio para obtenção de grau de mestre em
Matemática Aplicada para a Indústria

Orientador(es):
Professor Doutor Engenheiro Carlos José Brás Geraldes, ISEL
Professor Doutor Filipe Santiago Cal, ISEL

Júri:
Presidente: Professor Doutor Luís Manuel Ferreira da Silva, ISEL
Vogais:
Professor Doutor Engenheiro Carlos José Brás Geraldes, ISEL
Professor Doutor Nuno David de Jesus Lopes, ISEL

**Julho 2024**

# Acknowledgements

This internship was carried out, through a cooperation protocol with ISEL, at Infraestruturas de Portugal under the local supervision of Engineer João Gachet Alves.

First and foremost, I want to express my deepest gratitude to my parents, siblings, and sister-in-law for their unwavering support and encouragement throughout my academic journey. Their belief in me has been a constant source of motivation and strength. I could not have reached this point without their love and guidance.

I am profoundly thankful to Jewel Vicente and Rúben Mareco for their invaluable support and friendship. Their encouragement and companionship have been instrumental in my success, and I am incredibly grateful for their presence in my life.

I extend my heartfelt thanks to Instituto Superior de Engenharia de Lisboa (ISEL) for providing me with the opportunity to pursue this master's degree. The knowledge and experiences gained here have been pivotal in my academic and professional development, and I am sincerely grateful for this invaluable journey.

The Institut National des Sciences Appliquées (INSA) deserves special appreciation for the incredible Erasmus semester. The experience was invaluable, contributing significantly to my growth both personally and academically. The memories and lessons from INSA will always hold a special place in my heart.

Acknowledgement is given to all my classmates for their camaraderie and the collaborative spirit that made our time together so enriching. The bonds formed during this period have been a source of great joy and support.

I am deeply appreciative of every teacher from the master's program for their significant roles in shaping my academic journey. Special mention is given to:

- **Prof. Doctor Luís Manuel Ferreira da Silva**, the president of the master's degree program, for his unwavering assistance and guidance throughout the course. His leadership and support have been incredibly valuable.

- **Prof. Doctor Teresa Melo Quinteiro**, for her support with all matters related to Erasmus, ensuring a smooth and enriching experience. Her dedication and help have been greatly appreciated.
- **Prof. Doctor José Leonel Linhares da Rocha**, for his support and mentorship with all matters related to my master's, personal life, and Erasmus. His guidance has been a cornerstone of my success.
- **Prof. Doctor Engineer Carlos José Brás Geraldes** and **Prof. Doctor Filipe Santiago Cal**, for accepting the roles of supervisors from ISEL's side for this internship. Their mentorship and support have been invaluable during this crucial chapter of my life.

Heartfelt thanks are also conveyed to "Infraestruturas de Portugal" for offering me this remarkable opportunity. Special thanks are given to:

- **Engineer João Gachet Alves**, the supervisor from the company's side, for his expert guidance and support. His insights and assistance have been crucial.
- **Renato Vaz** and **Amadeu Silva**, for their warm welcome and constant readiness to guide and assist. Their support has made my journey smoother and more rewarding.

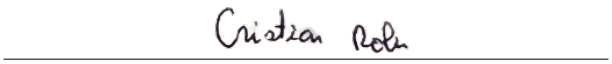
The collective support I have received has been instrumental in my achievements, and I am profoundly grateful for all the contributions to my success. Your encouragement and belief in me have made this journey possible.

Thank you all for being an integral part of this journey. Your support and kindness have touched me deeply, and I am forever grateful.

# Statement of integrity

I declare that this dissertation / project work / internship report is the result of my personal and independent research. Its content is original, and all sources listed in the bibliographic references were consulted and are duly mentioned in the text. I further declare that all scientific and technical references relevant to the development of the work are duly cited and included in the bibliographic references.

The author



Lisbon, ..07..,2024

# Resumo

Garantir a segurança e a continuidade operacional da infraestrutura ferroviária requer precisão, eficiência e fiabilidade nas práticas de manutenção e gestão. Os métodos tradicionais de gestão de ativos, caracterizados por processos manuais e altas taxas de erro, muitas vezes não cumprem estes requisitos devido à sua natureza laboriosa e custos significativos. Esta tese visa superar esses desafios integrando tecnologias avançadas de inteligência artificial (IA) no processo de gestão de ativos, com especial enfoque no *deep learning* para a deteção automática de objetos.

Esta investigação explora a aplicação do YOLOv9, a mais recente iteração do algoritmo de deteção de objetos *You Only Look Once*, para identificar e catalogar ativos ferroviários a partir de imagens de alta resolução capturadas por drones. O YOLOv9 é particularmente adequado para esta tarefa devido à sua eficiência no processamento de grandes volumes de imagens de alta resolução e à sua robustez em fornecer deteções de alta precisão em tempo real.

O estudo envolveu o treino do modelo YOLOv9 com um conjunto de dados de imagens capturadas por drones, que foram cuidadosamente anotadas com as localizações e identidades de vários ativos ferroviários. O desempenho do modelo foi avaliado medindo a sua *precision* e *recall* na deteção desses ativos sob uma ampla gama de diferentes condições ambientais.

Os resultados revelam que a IA pode melhorar a precisão e eficiência da gestão de ativos ferroviários. O modelo YOLOv9 reduziu significativamente o tempo necessário para inspeções de ativos, contribuindo assim para operações ferroviárias mais seguras. Esta abordagem destaca a escalabilidade e flexibilidade das tecnologias de IA na gestão de infraestruturas, oferecendo valiosos *insights* sobre o seu potencial para aplicações mais amplas em diversos setores.

Esta tese sublinha o potencial transformador da IA na melhoria da gestão de infraestruturas críticas e estabelece as bases para futuras investigações neste campo em rápida evolução.

## Key words

Inteligência Artificial, YOLOv9, Gestão de Ativos Ferroviários, Imagens de Drones, *Deep Learning*

# Abstract

Ensuring the safety and operational continuity of railway infrastructure requires precision, efficiency, and reliability in maintenance and management practices. Traditional methods of asset management, characterized by manual processes and high error rates, often fall short of these requirements due to their labor-intensive nature and significant costs. This thesis aims to overcome these challenges by integrating advanced artificial intelligence (AI) technologies into the asset management process, with a particular focus on deep learning for automated object detection.

This research explores the application of YOLOv9, the latest iteration of the You Only Look Once object detection algorithm, for identifying and cataloguing railway assets from high-resolution drone-captured images. Drones provide a unique aerial perspective, enabling comprehensive monitoring of extensive and often inaccessible railway areas. YOLOv9 is especially well-suited for this task due to its efficiency in processing large volumes of high-resolution images and its robustness in delivering real-time, high-accuracy detections.

The study involved training the YOLOv9 model on a dataset of drone-captured images, which were carefully annotated with the locations and identities of various railway assets. The performance of the model was assessed by measuring its precision and recall in detecting these assets under a wide range of different environmental conditions.

The findings reveal that AI can improve the precision and efficiency of railway asset management. The YOLOv9 model significantly reduced the time required for asset inspections, thereby contributing to safer railway operations. This approach highlights the scalability and flexibility of AI technologies in infrastructure management, offering valuable insights into their potential for broader applications across various sectors.

The findings underscore the transformative potential of AI in improving the management of critical infrastructure and lay the groundwork for future research in this rapidly evolving field.

## Key words

Artificial Intelligence, YOLOv9, Railway Asset Management, Drone Imaging, Deep Learning

# Contents

# List of Figures

# List of Tables

# Symbols & Abbreviations

## Symbology

### Latin

| | |
|---|---|
| $x'$ | New x-coordinate after rotation |
| $y'$ | New y-coordinate after rotation |
| $px$ | Original x-coordinate of the point |
| $py$ | Original y-coordinate of the point |
| $cx$ | x-coordinate of the centre of rotation |
| $cy$ | y-coordinate of the centre of rotation |
| $I$ | Input image |
| $K$ | Convolutional kernel |
| $S$ | Number of grid cells in YOLO |
| $B$ | Number of bounding boxes per grid cell |
| $C$ | Number of classes |

### Greek

| | |
|---|---|
| $\alpha$ | Learning rate in model training |
| $\beta$ | Coefficient for bounding box regression |
| $\theta$ | Angle of rotation |
| $\sigma$ | Sigmoid function |
| $\phi$ | Activation function |
| $\eta$ | Learning rate |
| $\epsilon$ | Small constant in optimization algorithms |
| $\mu$ | Mean in batch normalization |
| $\sigma^2$ | Variance in batch normalization |
| $\theta$ | Model parameters |

# Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| CVAT | Computer Vision Annotation Tool |
| GPU | Graphics Processing Unit |
| IP | Infraestruturas de Portugal |
| mAP | Mean Average Precision |
| mAP50 | Mean Average Precision at IoU threshold of 50% |
| mAP50-95 | Mean Average Precision averaged over IoU thresholds from 50% to 95% |
| SPPF | Spatial Pyramid Pooling-Fast |
| CNN | Convolutional Neural Network |
| SGD | Stochastic Gradient Descent |
| RMSprop | Root Mean Square Propagation |
| Adam | Adaptive Moment Estimation |
| COCO | Common Objects in Context |
| P | Precision |
| R | Recall |
| F1 | F1 Score |
| TP | True Positive |
| FP | False Positive |
| TN | True Negative |
| FN | False Negative |
| YOLO | You Only Look Once |

# Chapter 1

# Introduction

In recent years, the integration of artificial intelligence (AI) into asset management has revolutionized the way industries monitor and maintain their infrastructures. Traditional methods of railway asset management are often labor-intensive, costly, and prone to human error, highlighting the need for more accurate, efficient, and scalable solutions. The railway sector, a critical component of transportation and commerce, therefore, stands to benefit significantly from advanced AI applications.

This thesis investigates the application of AI, particularly deep learning techniques, to automate the recognition and cataloguing of railway assets using high-resolution drone-captured images. Drones offer a unique vantage point and the capability to capture detailed images across extensive and often inaccessible railway networks, making them ideal for this task. The use of drones for image acquisition provides the means to efficiently gather high-resolution data, which is essential for the precise identification and monitoring of railway assets.

The selection of YOLOv9, a cutting-edge iteration of the You Only Look Once (YOLO) object detection algorithm, is based on its efficiency and accuracy. YOLOv9 excels at processing the high-volume and high-resolution data typical of drone imagery, making it an optimal choice for this application. This project aims to demonstrate how YOLOv9 can be trained to accurately detect various railway assets, including tracks, and other critical components, thereby contributing to the digital transformation of railway infrastructure management.

The methodology of this research involved several stages, starting with data acquisition and preprocessing. High-resolution images captured by drones were processed and segmented to create a comprehensive dataset for training the YOLOv9 model. Annotations were meticulously applied to these images to mark the locations and identities of railway assets, providing the necessary ground truth for supervised learning.

To enhance the model's robustness and accuracy, various data augmentation techniques were employed, including rotation, background texturing, and additional background images.

These techniques ensured that the model could generalize well across different environmental conditions and asset appearances. The training process leveraged the computational power of GPUs to handle the intensive tasks of deep learning, with the model's performance being continuously monitored and evaluated through metrics such as precision, recall, and mean average precision (mAP).

One of the key achievements of this project was the development of a web-based application to make the AI model accessible and user-friendly. The application, built using Flask for the backend and React for the frontend, allows users to upload images or directories of images, select a detection model, and receive processed results with detected assets. This user interface significantly lowers the barrier to entry for non-technical users, enabling railway infrastructure managers to leverage advanced AI capabilities without needing in-depth technical knowledge.

By automating the asset detection process, the proposed solution seeks to reduce operational costs, enhance accuracy, and decrease the time required for inspection and maintenance. Additionally, this approach offers scalability and adaptability to various environments and conditions, showcasing the versatility of AI applications in real-world settings. The integration of AI into railway asset management not only streamlines the monitoring process but also improves the reliability and safety of railway operations.

This thesis presents the methodology and results of applying YOLOv9 to railway asset detection and discusses the broader implications of such technological advancements in smart infrastructure management. This research, aims to provide insights into the challenges and opportunities of deploying AI technologies for the maintenance and monitoring of critical transportation infrastructures, while underscore the transformative potential of AI in enhancing the efficiency and effectiveness of infrastructure management and paving the way for future innovations in this field.

## Thesis Structure

This thesis is organized into several chapters, each focusing on different aspects of the research:

- **Chapter 1: Introduction**

  Provides an overview of the research, outlining the objectives, significance, and scope of the study.

- **Chapter 2: Artificial Intelligence - Foundations and Applications**

  Explores the fundamental concepts of AI, including its theoretical frameworks, learning paradigms, neural networks, and CNN's. It looks into the core methods of deep learning, which form the basis for advanced AI applications.

- **Chapter 3: Company Description**

  An overview of Infraestruturas de Portugal, discussing its mission, operations, and strategic importance. It highlights the company's commitment to innovation and sustainability, including the internship experience and its impact.

- **Chapter 4: Data Acquisition and Preprocessing**

  Outlines the process of obtaining and preparing the data for model training. It covers image acquisition, initial data processing, image segmentation, the application of annotations, and discusses data augmentation techniques and the train-test split methodology.

- **Chapter 5: YOLOv9 and Model Training**

  Details the training process of the YOLOv9 model, including the optimization algorithms used, the evaluation metrics, the results obtained, and also covers the challenges encountered during training and the solutions implemented to overcome them.

- **Chapter 6: Discussion of Results and Model Performance Analysis**

  Provides an in-depth analysis of the results obtained from the model training and evaluation, and analyses the performance of the YOLOv9 model in detecting railway assets.

- **Chapter 7: Development of a User Interface and Technology Transfer**

  Describes the development of the web-based application, including its framework, design, implementation, and discusses user interaction and feedback mechanisms to ensure the interface is user-friendly and effective.

- **Chapter 8: Conclusion**

  Summarizes the key findings of the research, discusses the implications of the results, and outlines potential future work and the broader impact of AI in railway infrastructure management.

# Chapter 2

# Artificial Intelligence - Foundations and Applications

## 2.1 Artificial Intelligence

This section aims to explain the vast and complex domain of Artificial Intelligence (AI), a multifaceted field driving significant advancements across various industries. AI encompasses the development of computer systems capable of performing tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language translation [1, 2, 3, 4, 5, 6].

AI operates through the creation and application of algorithms, which are sets of rules or instructions that enable a computer to learn and make decisions autonomously. A pivotal element of AI is machine learning, where computers are trained to learn from and interpret data without human intervention. Within machine learning, deep learning stands out for its use of neural networks—complex algorithms designed to mimic the operations of the human brain. These networks can learn unsupervised from unstructured or unlabelled data, making them incredibly powerful for various applications.

### 2.1.1 Theoretical Frameworks and Learning Paradigms in AI

Understanding AI's theoretical frameworks and learning paradigms is crucial for grasping its capabilities and limitations. One of the main approaches in AI and the one used for this project is supervised learning [7].

#### Supervised Learning

In supervised learning, the algorithm is trained on a labelled dataset, which means the model is provided with the correct answers beforehand. This method involves learning the mapping between inputs and outputs, allowing the machine to make accurate predictions based on new data (Fig 2.1). Mathematically, supervised learning attempts to approximate the function $f$ :

$X \to Y$, where $X$ is the input space and $Y$ is the output space. The objective is to minimize the loss function $L(y, \hat{y})$, where $y$ is the true label and $\hat{y}$ is the predicted label [8].


**Figure 2.1** Model predicts this image as a dog

### 2.1.2 Fundamental Methods in Deep Learning

#### Neural Networks

Neural networks [9] are the foundational building blocks of deep learning. They consist of layers of interconnected neurons, where each neuron performs a weighted sum of its inputs, adds a bias term, and applies an activation function. Mathematically, the output of a single neuron $y$ can be expressed as:

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

where $x_i$ are the input features, $w_i$ are the weights, $b$ is the bias, and $\phi$ is the activation function.

### Single-layer Perceptron

A single-layer perceptron [10] can be viewed as a Generalized Linear Model (GLM), where the perceptron acts similarly to logistic regression but with multiple layers. The perceptron, a basic unit of a neural network, calculates the weighted sum of the input values and applies a step function to determine its output. For instance, with two inputs $x_1$ and $x_2$, and weights $w_1$ and $w_2$, the weighted sum is:

$$y = \begin{cases} 1 & \text{if } w_1 x_1 + w_2 x_2 > \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

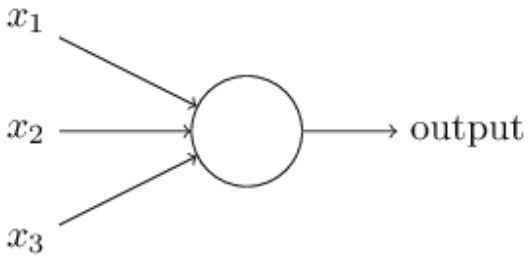



**Figure 2.2** Structure of a Single-layer Perceptron [11].

### Multilayer Perceptron:

Multilayer Perceptron (MLP), a neural network that consists of an input layer, one or more hidden layers, and an output layer (Fig 2.3). Each layer transforms the input data through weights, biases, and activation functions, enabling the network to learn complex patterns. The network is trained using backpropagation, which involves computing the gradient of a loss function with respect to the weights and biases, and updating them using gradient descent [2].

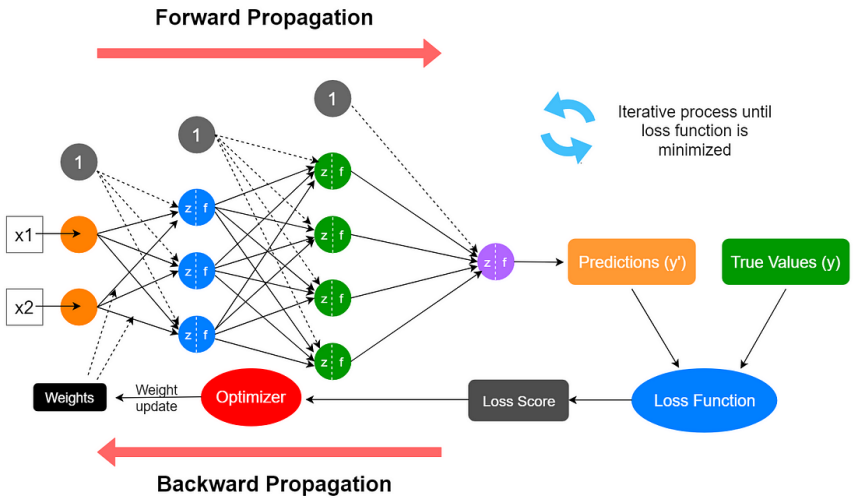



**Figure 2.3** Structure of a Multilayer Perceptron [12].

### Loss Functions

Loss functions are critical in training neural networks, as they quantify how well the model's predictions match the actual data. Common loss functions include:

- **Mean Squared Error (MSE):** Often used for regression tasks, it calculates the average squared difference between predicted and actual values.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.3}$$

- **Cross-Entropy Loss:** Commonly used for classification tasks, it measures the performance of a classification model, with the cross-entropy output being a probability value between 0 and 1.

$$\text{Cross-Entropy} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{2.4}$$

### Backpropagation and Gradient Descent

Backpropagation is the algorithm used to train neural networks by minimizing the loss function. It calculates the gradient of the loss function with respect to each weight by the chain rule, iterating backwards from the output layer to the input layer. The weights are then updated using gradient descent [1]:

$$\theta := \theta - \eta \nabla_\theta J(\theta) \tag{2.5}$$

where $\theta$ represents the model parameters (weights and biases), $\eta$ is the learning rate, and $J(\theta)$ is the loss function.

The backpropagation process can be summarized in the following steps:

- **Forward Pass:** Compute the output of the network by passing the input data through each layer.
- **Compute Loss:** Calculate the loss function based on the network's output and the actual target values.
- **Backward Pass:** Compute the gradient of the loss function with respect to each weight using the chain rule.
- **Update Weights:** Adjust the weights using the gradients to minimize the loss function.

The backpropagation algorithm ensures that the network learns to adjust its weights and biases to reduce the error, making the model more accurate over time.

This process is repeated for multiple iterations or epochs until the model converges to a minimum loss. Backpropagation, combined with optimization techniques like gradient descent, forms the backbone of neural network training, enabling the development of highly accurate and efficient models for various applications.

### Multilayer Perceptron Problems

A problem arises as the input layer size increases, leading to an exponential growth in the number of parameters. This makes MLPs computationally expensive and prone to overfitting when dealing with high-dimensional data, such as images.

Another significant issue with MLPs, particularly in deep networks, is the vanishing/exploding gradient problem. In deep neural networks, the gradients used for updating the weights can become very small (vanishing) or very large (exploding) as they are propagated backward through the layers during training. This makes it difficult for the network to learn in the earlier layers, slowing down convergence and potentially leading to poor performance.

### Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) [13] were developed to address the issues posed by Multilayer Perceptrons when handling high-dimensional data. CNNs are highly effective for tasks involving image and video recognition. A CNN consists of layers of neurons that process input data through a series of convolutional, pooling, and fully connected layers (Fig 2.4). The convolutional layers apply filters to the input data, detecting various features such as edges, textures, and shapes. Mathematically, a convolution operation of functions $f$ and $g$ is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \tag{2.6}$$

In discrete terms, for a 2D input image $I$ and a filter $K$, the convolution operation can be expressed as:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{2.7}$$

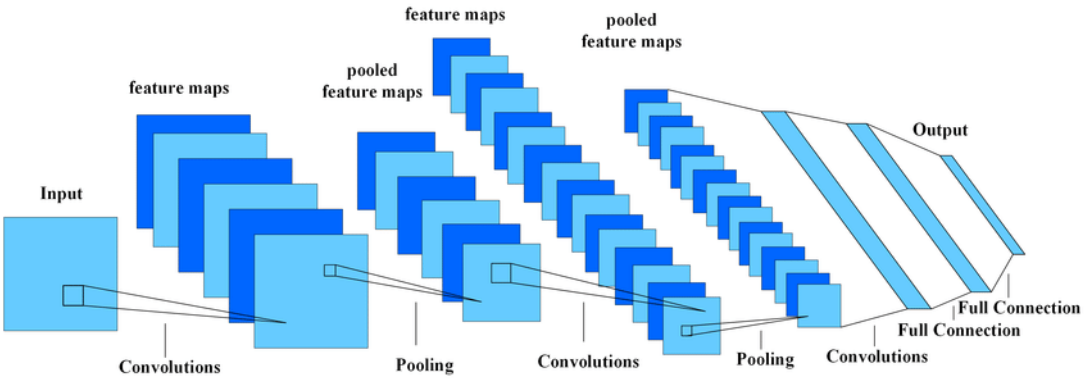



**Figure 2.4** Process of CNNs [14].

### Convolutional Layers

Convolutional layers are the core building blocks of a CNN. Each convolutional layer consists of a set of learnable filters (kernels) that slide over the input image to produce feature maps.

The filter $K$ slides over the input image $I$, applying the convolution operation to generate the output feature map $S$. This process allows the network to detect various features such as edges, textures, and shapes within the image. The result of the convolution operation is then passed through an activation function, which will be explained in a later section.

Filters can be designed/emphasize to detect specific features in an image, such as vertical and horizontal lines. Here are examples of such filters:

A filter to detect vertical lines:

$$K_{\text{vertical}} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

A filter to detect horizontal lines:

$$K_{\text{horizontal}} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

When these filters are applied to an input image $I$, the resulting feature maps $S_{\text{vertical}}$ and $S_{\text{horizontal}}$ will highlight the vertical and horizontal lines, respectively.

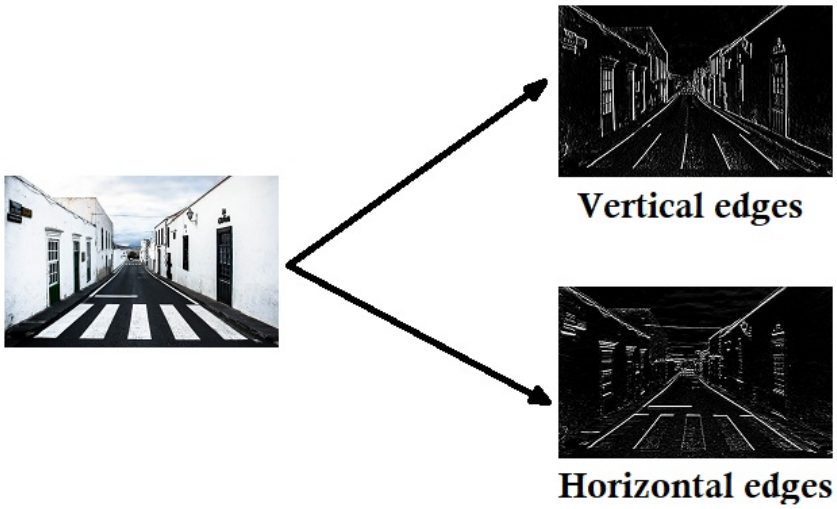



**Figure 2.5** Example of vertical and horizontal line detection using convolution filters. The original image (left) is processed by the vertical line filter (top right) and the horizontal line filter (bottom right) [15].

As shown in Figure 2.5, the original image is processed by the vertical and horizontal filters to produce feature maps that emphasize vertical and horizontal lines, respectively. These feature maps can then be used in subsequent layers of the CNN to detect more complex patterns and objects.

### Pooling Layers

Pooling layers are used to reduce the spatial dimensions of the data, thereby decreasing the number of parameters and computational load in the network. Two common types of pooling are max pooling and average pooling (Fig 2.6) [6].

Max pooling takes the maximum value in each patch of the feature map. The general formula for max pooling over a window of size $k \times k$ is:

$$y(i,j) = \max_{0 \leq m < k, 0 \leq n < k} \{x(i \cdot s + m, j \cdot s + n)\} \tag{2.8}$$

Average pooling, on the other hand, calculates the average value of the elements in each patch of the feature map. The general formula for average pooling over a window of size $k \times k$ is:

$$y(i,j) = \frac{1}{k^2} \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \{x(i \cdot s + m, j \cdot s + n)\} \tag{2.9}$$

In these equations:

- $x$ represents the input feature map.
- $y$ represents the output feature map after pooling.

- $i$ and $j$ are the indices of the pooled output feature map.
- $m$ and $n$ are the indices within the pooling window.
- $k$ is the size of the pooling window (e.g., $k = 2$ for a 2x2 window).
- $s$ is the stride, which is the number of pixels by which the window moves after each operation.

Stride controls the step size of the pooling operation. For instance, a stride of 1 means the pooling window moves one pixel at a time, while a stride of 2 means it moves two pixels at a time. Larger strides result in greater reduction of the spatial dimensions.

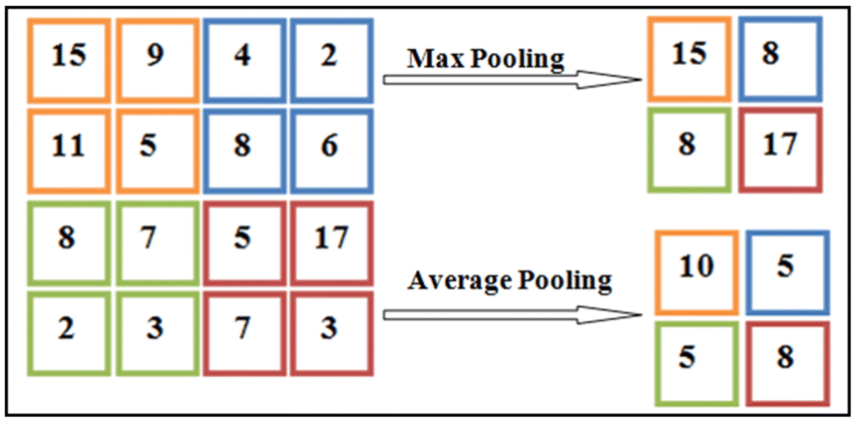



**Figure 2.6** Examples of max and average pooling operations over a 2x2 window with a stride of 2 [16].

Pooling layers help in reducing the dimensions of the input, thereby speeding up the computation and making the model less prone to overfitting. While max pooling captures the most prominent features, average pooling provides a smoother result by averaging the values. Both techniques are crucial in making the neural network invariant to small translations of the input data, thus improving the model's generalization capability.

### Activation Functions

Activation functions [17] introduce non-linearity into the network, allowing it to learn complex patterns. Common activation functions include:

- **Sigmoid:** The sigmoid function is widely used, especially in logistic regression and binary classification tasks. It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.10}$$

  The sigmoid function maps any real-valued number into the range (0, 1), making it useful for probabilistic interpretations.

  Here is a plot of the sigmoid function:

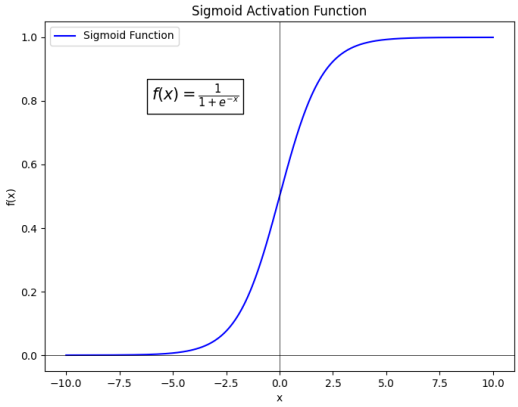



**Figure 2.7** Sigmoid activation function

- **Softmax:** The softmax function is commonly used in the output layer of a neural network for classification tasks. It converts the output scores into probabilities, which sum to 1. The softmax function for a vector **x** is defined as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.11}$$

Here is a plot of the softmax function:

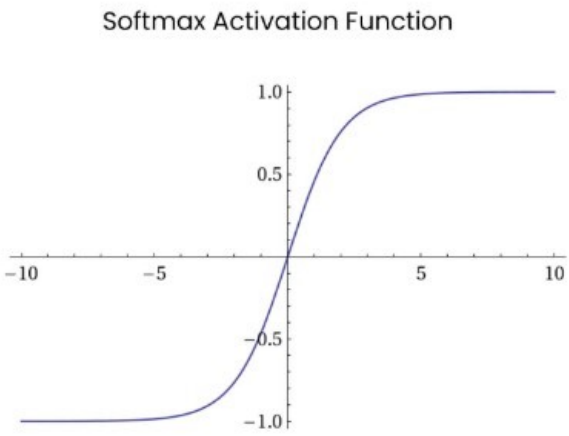



**Figure 2.8** Softmax activation function [15].

- **Hyperbolic Tangent (Tanh):** The hyperbolic tangent (tanh) function maps input values to the range (-1, 1). It is defined as:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.12}$$

Tanh is often preferred over the sigmoid function because its output is zero-centered, which can make training more efficient.

Here is a plot of the Tanh function:

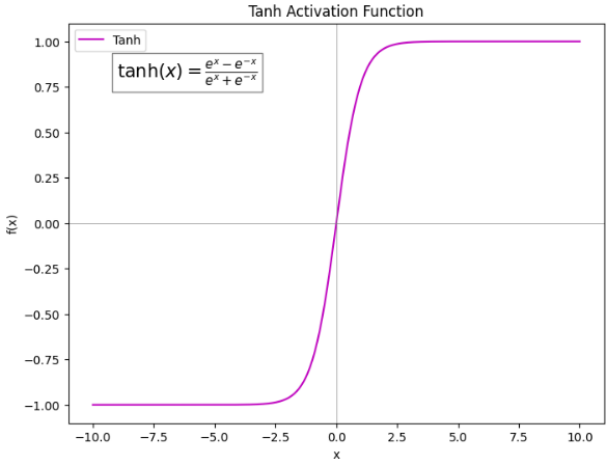



**Figure 2.9** Tanh activation function

- **ReLU (Rectified Linear Unit):** ReLU is particularly popular in deep learning due to its simplicity and effectiveness. Mathematically, ReLU is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{2.13}$$

ReLU has several advantages over the sigmoid and tanh functions, making it more suitable for deep learning neural networks. Firstly, ReLU helps mitigate the vanishing gradient problem that can occur with sigmoid and tanh. In ReLU, gradients are not squashed into a small range, so the backpropagation of errors does not diminish as much, allowing for faster and more efficient training of deep networks. Secondly, ReLU introduces sparsity in the network by setting negative values to zero, which can lead to a more efficient and interpretable model. Additionally, ReLU is computationally less expensive than sigmoid and tanh because it requires only a simple thresholding operation.

Here is a plot of the ReLU function:

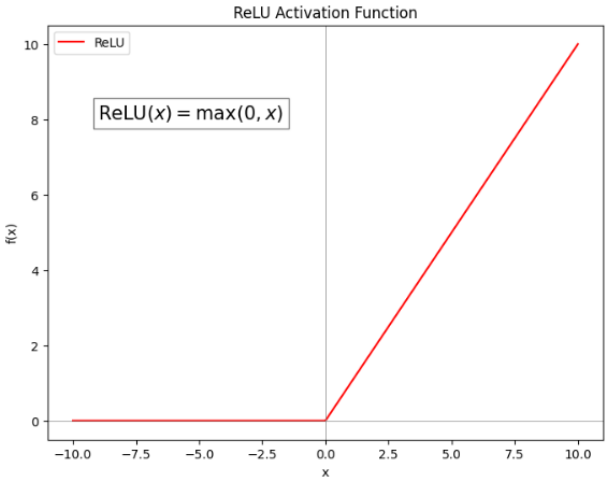



**Figure 2.10** ReLU activation function

- **SiLU (Sigmoid Linear Unit):** Also known as the swish function, SiLU is a newer activation function that has shown promise in recent research. It is defined as:

$$\text{SiLU}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1 + e^{-x}} \tag{2.14}$$

SiLU combines the properties of both linear and non-linear activation functions, allowing it to retain the input value for positive values while introducing non-linearity for negative values. This can help improve the learning capacity of the network, and it's used in models like YOLO [18].

Here is a plot of the SiLU function:

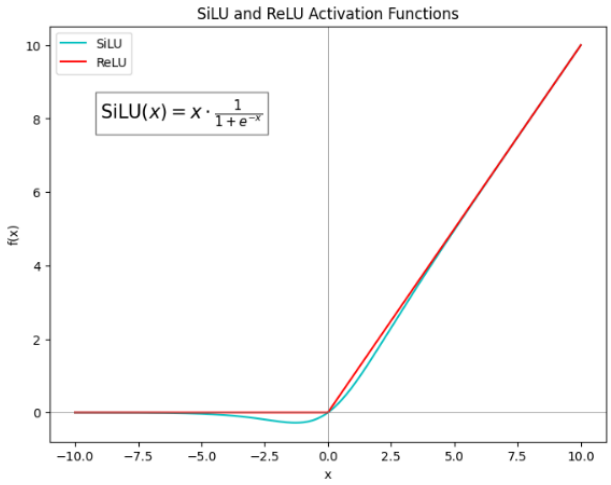



**Figure 2.11** SiLU function compared to ReLU

## Normalization Layers

Normalization layers [2], such as batch normalization, standardize the inputs to each layer, improving training speed and stability. Batch normalization normalizes the output of the previous layer by subtracting the batch mean and dividing by the batch standard deviation. The normalized output $\hat{x}$ is given by:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.15}$$

where $\mu$ is the mean, $\sigma^2$ is the variance, and $\epsilon$ is a small constant added for numerical stability.

## Dropout

Dropout [2] is a regularization technique used to prevent overfitting in neural networks. During training, dropout randomly sets a fraction of the input units to zero at each update. This prevents

the network from relying too heavily on any single neuron, encouraging the model to learn more robust features. If $p$ is the dropout probability, the output $y$ during training is:

$$y = \frac{d \cdot x}{1 - p} \tag{2.16}$$

where $d$ is a binary mask with each element $d_i$ being zero with probability $p$.

### 2.1.3 Training Techniques and Regularization

Training deep learning models, involves various techniques to improve performance and prevent overfitting. Here, some crucial methods are discussed:

#### Data Augmentation

Data augmentation involves creating variations of the training data to artificially expand the dataset. This technique helps the model generalize better by learning from diverse data samples. Common augmentation methods include random cropping, flipping, rotation, colour adjustments, and altering the background with random patterns or noise [2].

Mathematically, if $x$ is an input image and $T$ is a transformation function, then the augmented image $x'$ can be expressed as:

$$x' = T(x) \tag{2.17}$$

Different data augmentation techniques are showcased in Figure 2.12.

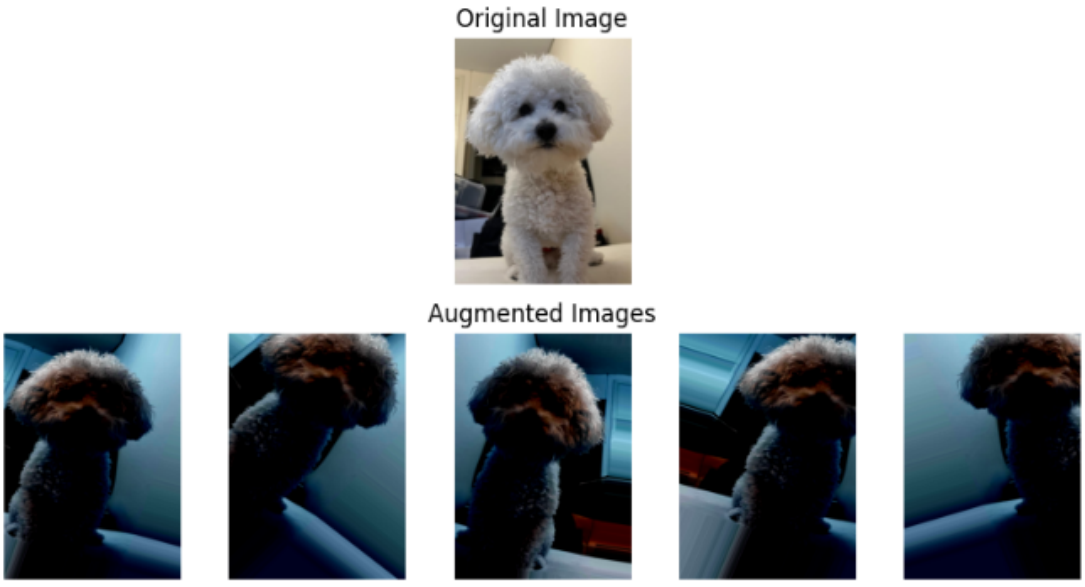



**Figure 2.12** Data augmentation example

### Early Stopping

Early stopping is a regularization technique used to prevent overfitting by terminating the training process when the model's performance on a validation set starts to degrade. This approach monitors the validation loss and stops training when there is no improvement for a pre-defined number of epochs. Let $L_{\text{val}}$ be the validation loss, then training stops at epoch $t$ if [1]:

$$L_{\text{val}}(t) > L_{\text{val}}(t - n) \tag{2.18}$$

for $n$ consecutive epochs.

### Transfer Learning

Transfer learning leverages pre-trained models on large datasets, such as ImageNet, to initialize the weights of a new model. This technique allows the new model to benefit from the pre-learned features, significantly reducing the amount of data and training time required for the new task.

The fundamental idea behind transfer learning is that the features learned by a model trained on a large, diverse dataset can be useful for a variety of different tasks. For instance, a model trained on ImageNet learns to recognize edges, textures, shapes, and other low-level features in the early layers, while higher layers learn more abstract concepts such as object parts and entire objects. These learned features can be transferred to a new model, providing a strong starting point for learning a new task.

Transfer learning is particularly beneficial in scenarios where:

- **Limited Data Availability:** When the new task has a limited amount of labelled data, leveraging the features learned from a larger dataset helps improve performance.
- **Computational Resources:** Training deep neural networks from scratch is computationally expensive and time-consuming. Transfer learning reduces the computational burden by starting with a pre-trained model.
- **Quick Prototyping:** Transfer learning allows for rapid development and prototyping of models, accelerating the experimentation and iteration process.

Overall, transfer learning is a powerful technique that takes advantage of the knowledge embedded in pre-trained models, leading to improved performance and faster convergence for new tasks. By leveraging pre-trained weights, the new model can achieve higher accuracy with less data and fewer computational resources, making it a practical approach in many real-world applications [19, 20].

# Chapter 3

# Company Description

## 3.1 Overview of Infraestruturas de Portugal

Infraestruturas de Portugal (IP) [21] is a pivotal entity in Portugal's transportation sector, responsible for the management and modernization of both road and rail infrastructures. Formed in 2015 through the merger of Rede Ferroviária Nacional (REFER) and Estradas de Portugal, IP embodies a strategic consolidation aimed at enhancing the efficiency and effectiveness of the country's transportation infrastructure management. As a state-owned enterprise, IP operates under the auspices of the Portuguese government, reflecting its national significance.

## 3.2 Mission and Strategic Importance

IP's mission is to design, project, construct, finance, maintain, operate, requalify, expand, and modernize the national road and railway networks. This includes the command and control of railway and road circulation. IP acts as a catalyst for technological innovation and sustainability in the transport sector making IP a crucial supporter of multimodal mobility and bolstering Portugal's economic and social development.

## 3.3 Operations and Management

IP's operations are extensive, managing 14,042 km of direct road networks and 1,014 km of sub-concessioned roads, alongside 2,527 km of rail tracks. Annually, these networks facilitate the circulation of approximately 24.8 billion people by road and 34.9 million people by rail. These figures underscore the vast scale and critical importance of IP's operational capabilities.

## 3.4 Commitment to Innovation and Sustainability

IP is committed to driving innovation and sustainable practices within the transport sector. This commitment is illustrated by its involvement in high-profile projects such as the development of a high-speed rail line between Lisbon and Porto and the FCH2Rail hydrogen train

project. These initiatives highlight IP's dedication to reducing environmental impacts and enhancing the efficiency and sustainability of Portugal's transportation infrastructure. The use of drones and AI for infrastructure inspection and asset management is another example of IP's forward-thinking approach.

## 3.5 Future Directions

Looking forward, IP is poised to continue its pivotal role in transforming Portugal's transportation landscape. With plans to implement significant projects and adopt cutting-edge technologies, IP aims to enhance the connectivity and competitiveness of Portugal's transport networks at both the Iberian and European levels. The organization's forward-looking strategies focus on improving transport efficiencies in the immediate term and contributing to long-term economic growth and environmental sustainability.

## 3.6 Internship Experience

The internship at IP gave an opportunity to contribute to innovative projects by applying AI to analyse high-resolution drone images for the recognition of railway assets. The project involved developing and training models, particularly YOLOv9 [22], to automate the detection and cataloguing of various railway components which aims to enhance the efficiency of railway asset management, significantly reducing operational costs and inspection times. The internship also provided vital hands-on experience, offering practical knowledge about the implementation of AI and a wider grasp of its possible applications, providing significant real-world experience and a deep comprehension of the company's operations.

## 3.7 Strategic Vision and Impact

IP stands as a cornerstone in Portugal's infrastructure landscape, driven by a progressive vision for a connected and sustainable future. The company's commitment to innovation and sustainability is evident in its strategic projects and initiatives, which keep Portugal at the forefront of transportation technology. By continuously enhancing its infrastructure capabilities, IP ensures the nation's competitiveness and efficiency in the transportation sector.

IP's forward-thinking approach, exemplified by its use of advanced technologies like AI and drones, significantly improves operational efficiency and environmental sustainability. This dedication to technological advancement and sustainable practices underscores IP's critical role in shaping the future of transportation in Portugal.

In summary, IP's strategic vision and impactful initiatives are vital in driving progress within the transportation sector, enhancing connectivity, and fostering sustainable development. Through

its unwavering commitment to innovation and strategic growth, IP continues to play an indispensable role in the advancement of Portugal's infrastructure.

# Chapter 4

# Data Acquisition and Preprocessing

## 4.1 Data Obtainment

The journey of creating a robust AI model begins with the acquisition of high-resolution aerial imagery. These images are essential for the precise identification and monitoring of railway assets, providing the detailed visual data necessary for accurate object detection and subsequent analytical assessments.

### 4.1.1 Image Acquisition

The image acquisition process involves a coordinated effort by the field team to capture high-quality aerial images. Preparation starts with selecting an optimal date for the flight, considering critical factors such as weather conditions and lighting, which are crucial for obtaining clear and consistent imagery. On the designated day, the team deploys to the site with the drone equipment, ready to execute the planned operation.

The drone, shown in figure 4.1, is programmed to follow a predetermined flight path over the area of interest. Equipped with advanced sensors, the drone captures images at high resolutions and throughout the flight, the drone systematically documents every portion of the site, ensuring precise and thorough coverage.

**Figure 4.1** Trinity F9 drone owned by IP

### 4.1.2 Image Processing by the Company

After capturing the aerial imagery, the data undergoes a processing procedure conducted by the company. This phase transforms the raw aerial images into standardized 5000x5000 pixel .tif files. Processing steps include:

- **Stitching multiple images together:** Combining several images to create a seamless and comprehensive view of the surveyed area.
- **Correcting for distortions:** Addressing any distortions or environmental factors such as shadows or cloud cover that might affect the clarity of the images.
- **Enhancing image clarity and colour accuracy:** Improving the overall quality and accuracy of the images to ensure they are suitable for detailed analysis.

The high resolution of these processed images is extremely important. It ensures that even the smallest details are visible and distinguishable, which is essential for accurately identifying specific objects within the images. The final high-resolution images serve as the foundational dataset for all subsequent object detection and analysis tasks, ensuring that the data used in model training and evaluation phases are high quality.

## 4.2 Initial Data Processing

### 4.2.1 Image Segmentation

The first step in preprocessing the high-resolution images involves segmenting these large 5000x5000 pixel images into smaller, more manageable sizes. This process is essential for two main reasons: firstly, it adapts the images to the input size requirements of the YOLOv9 (the model used to train this dataset) [22], which is optimized for smaller images; secondly, handling smaller images significantly reduces the computational load during the training process.

To achieve this, a Python [23] script was created (Appendix A), which divides each large image into segments of 640x640 pixels. This specific size aligns with the optimal input dimensions for the YOLOv9 model, ensuring that each image segment retains enough detail for effective object detection while being computationally feasible to process.

#### Code Overview

- **Calculate Segments**: The script calculates the number of 640x640 pixel segments that can be extracted both horizontally and vertically from each original image.
- **Crop Images**: It systematically crops the original image into these segments, ensuring that each part of the image is captured.
- **Save Segments**: Each segment is saved as a separate file, making them ready for individual processing and analysis in the subsequent steps of the model training pipeline.

For instance, the script uses the Python Imaging Library (PIL) to open the image, determine its dimensions, and then crop it into 640x640 pixel segments. Each segment is then saved in a specified output directory.

```
# Example code snippet for cropping images
left = j * seg_width if (j + 1) * seg_width <= img_width else img_width - seg_width
upper = i * seg_height if (i + 1) * seg_height <= img_height else img_height - seg_height
right = left + seg_width
lower = upper + seg_height
segment = large_image.crop((left, upper, right, lower))
```

This segmentation process not only makes the images more suitable for processing by the YOLOv9 architecture but also enhances the efficiency of the entire image processing workflow.

### 4.2.2 Annotations of Objects

Once the images are segmented, the next step involves uploading these smaller image segments to CVAT.ai [24], an online platform. The objects of interest are manually labelled within

each image segment. These annotations are crucial as they provide the training data for the AI model.

Each annotation involves drawing bounding boxes around the objects of interest and labelling them accordingly. Once all objects in a segment are annotated, the bounding box data is downloaded in a TXT format, containing coordinates relative to each segment. This annotated data forms the basis for training the YOLOv9 model.

**Figure 4.2** Annotation using the CVAT tool

Through these steps, the high-resolution images are processed and annotated, making them ready for training the YOLOv9 model. This approach ensures that the data used in the model is both high quality and highly detailed, facilitating accurate and efficient object detection in the final application.

## 4.3 Final Data Processing and Preparation for Model Training

### 4.3.1 Rotation of Images

While the past steps are enough to train the model, there is another step that is crucial, which is data augmentation. To augment the data, one of the python scripts used is a python script that is designed to generate rotated versions of each image (Appendix B). This process not only increases the volume of training data but also enhances the model's robustness by training it to recognize objects from various orientations (Figure 4.3).

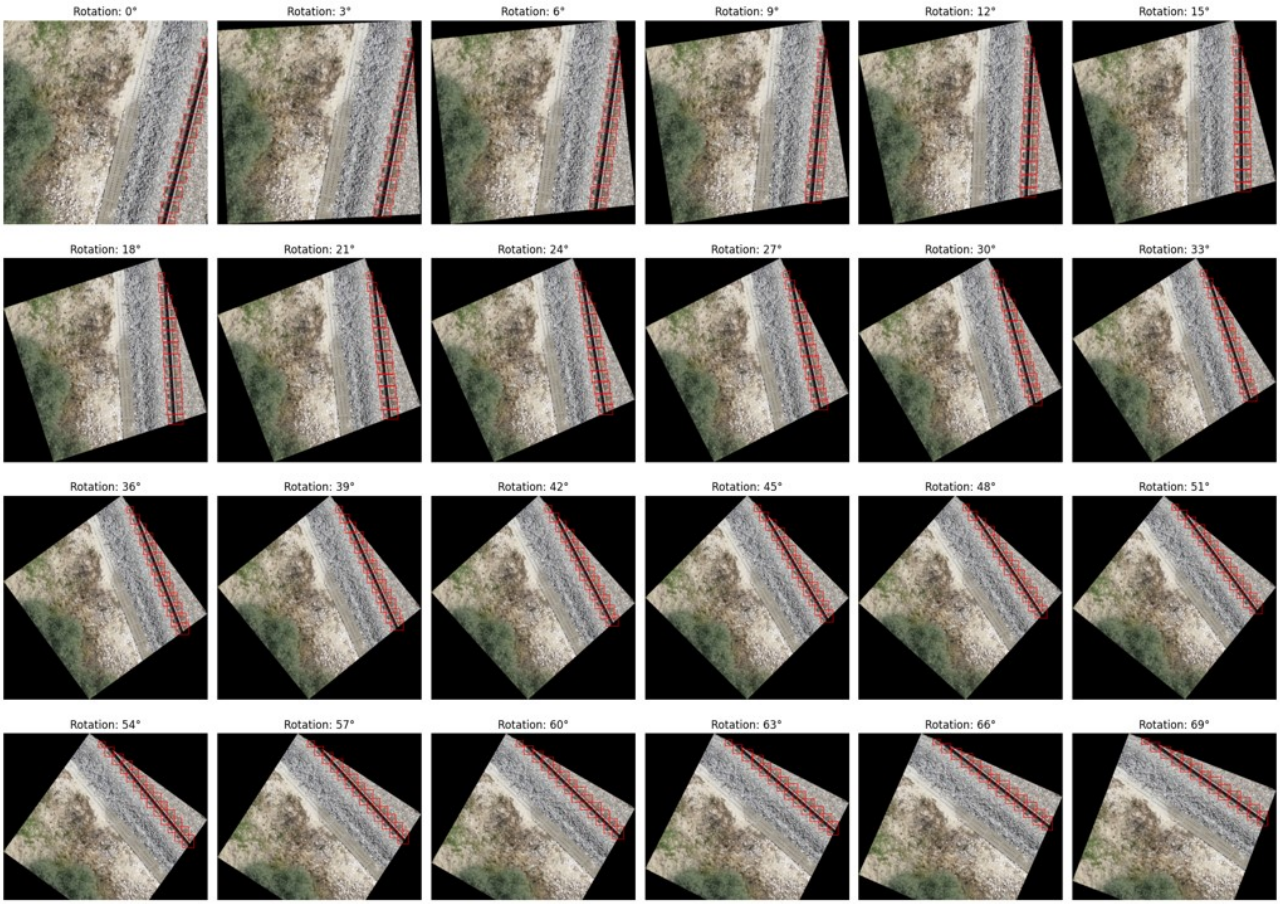



**Figure 4.3** Result from image rotation with bounding boxes

#### Code Overview

The script operates by taking each annotated image and rotating it around its centre through multiple angles. For each rotation, it recalculates the positions of the bounding boxes to correspond accurately to the new orientation of the objects within the image. The rotated images and their corresponding bounding boxes are then saved, ready to be used in training the model.

**Detailed Code Walkthrough**

**Rotation Functionality:**

Each image is rotated in k-degree increments from 0 to 360 degrees, ensuring comprehensive coverage of possible orientations. The rotation is performed around the image's centre to maintain the objects' relative positioning within the frame, which is crucial for preserving the integrity of the bounding box data.

```
import math
from PIL import Image
import numpy as np

def rotate_point(cx, cy, angle, px, py):
    """Rotate a point around a given center."""
    s, c = math.sin(angle), math.cos(angle)
    px -= cx
    py -= cy
    xnew = px * c - py * s
    ynew = px * s + py * c
    return xnew + cx, ynew + cy

def get_rotated_image_size(w, h, angle):
    """Calculate the size of the image after rotation."""
    radians = math.radians(angle)
    sin_angle = abs(math.sin(radians))
    cos_angle = abs(math.cos(radians))
    new_w = int(h * sin_angle + w * cos_angle)
    new_h = int(h * cos_angle + w * sin_angle)
    return new_w, new_h
```

The functions above are used to determine the new dimensions of the rotated image and to compute the coordinates of a point after rotation. The `rotate_point` function utilizes trigonometric transformations to rotate a point $(px, py)$ around a centre $(cx, cy)$ by an angle $\theta$. The new coordinates $(x', y')$ are calculated using the following equations:

$$x' = (px - cx)\cos(\theta) - (py - cy)\sin(\theta) + cx \tag{4.1}$$

$$y' = (px - cx)\sin(\theta) + (py - cy)\cos(\theta) + cy \tag{4.2}$$

The `get_rotated_image_size` function calculates the new dimensions of the image after rotation, considering the maximum extents of the rotated corners.

**Adjusting Bounding Boxes:**

Once an image is rotated, the bounding boxes must also be recalculated to fit the new orientation of the objects they encompass. This involves calculating the new coordinates for each corner of the bounding box after the image rotation and then determining the new bounding box's centre, width, and height from these coordinates.

```
def rotate_bbox(bbox, angle, original_size):
    """Rotates the bounding box coordinates."""
    cx, cy, w, h = bbox
    angle_rad = math.radians(angle)
    points = [
        [cx - w / 2, cy - h / 2],
        [cx + w / 2, cy - h / 2],
        [cx + w / 2, cy + h / 2],
        [cx - w / 2, cy + h / 2]
    ]
    new_points = [rotate_point(original_size / 2, original_size / 2, angle_rad, *p)\
    for p in points]
    new_cx, new_cy, new_w, new_h = calculate_new_bbox(new_points)
    return new_cx, new_cy, new_w, new_h
```

Here, the `rotate_bbox` function transforms the bounding box by rotating its corners using the same rotation matrix as before. The new bounding box is recalculated by finding the minimum and maximum coordinates of the rotated corners, ensuring that the bounding box accurately reflects the rotated object's position.

**Saving Rotated Images and Bounding Boxes:**

The script saves each rotated image along with its recalculated bounding box data. This ensures that each training instance (image and its annotations) maintains its utility for training the YOLOv9 model, reflecting accurate object locations and orientations.

### 4.3.2 Background Texture Application

After rotating the images, another python script is used to apply a gravel texture to the backgrounds within the bounding boxes, which is crucial for creating a uniform background, and helps to minimize background noise, thus focuses the model's training on the objects of interest rather than on variations in the background that could potentially be irrelevant to the task (Appendix E).

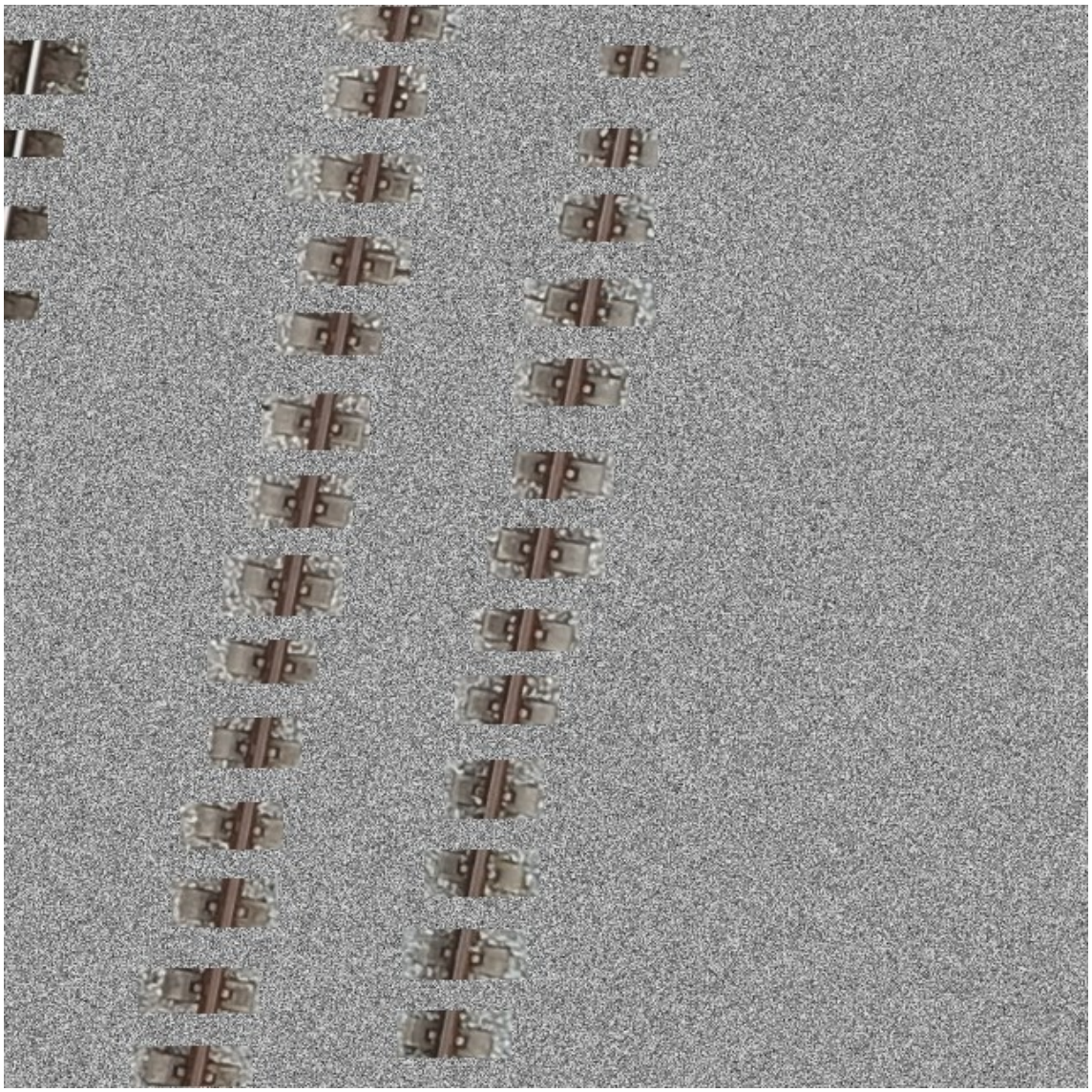


**Figure 4.4** Example of applying gravel texture to the background

The script follows a detailed process to ensure a natural and varied appearance for the background:

- **Random Texture Selection**: A random gravel texture image is selected from a predefined dataset to ensure variability across the images.
- **Pixel Sampling**: From the selected gravel texture, a random set of pixels is sampled. The number of pixels sampled is determined based on the area of the image that needs to be textured.
- **Background Application**: These sampled pixels are then applied to the areas outside the bounding boxes, effectively masking the original background. This step blends the edges of the objects into the background more seamlessly and maintains focus on the objects.

By applying these textures, the script ensures that the dataset used for training is consistent yet diverse. This focus on the relevant features of the objects enhances the accuracy and robustness of the object detection model, making it better suited for practical applications in diverse environmental conditions.

### 4.3.3 Inclusion of Pure Background Images

To enhance the model's ability not only to recognize objects of interest but also to understand what does not constitute an object of interest, an additional step is done, which involves incorporating pure background images into the dataset. These images, which do not contain any target objects, make up approximately 10% to 20% of the total image set [25].

This strategy is crucial for training a reliable object detection model. By including background images, we ensure that the model is well-rounded and capable of distinguishing between true objects and background noise, leading to more accurate and dependable performance in real-world applications.

### 4.3.4 Train-Test Split

To ensure the model's accuracy and robustness, the dataset is divided into training and validation sets using a 70/30 split. This division allows the model to be tested against unseen data, providing a reliable measure of its performance (Appendix C).

# Chapter 5

# YOLOv9 and Model Training

## 5.1 YOLO

The YOLO (You Only Look Once) [18] family of models represents a significant advancement in the field of real-time object detection. Unlike traditional object detection systems that typically use a sliding window approach or region proposal networks, YOLO approaches object detection as a single regression problem. This allows it to directly predict bounding boxes and class probabilities from full images in one evaluation, which greatly enhances its speed and efficiency [26].

**How YOLO Works:**

YOLO divides an input image into a $S \times S$ grid. Each grid cell is responsible for predicting $B$ bounding boxes and their corresponding confidence scores, which indicate how confident the model is that the box contains an object and how accurate it thinks the box is. Additionally, each cell predicts $C$ class probabilities for the object within the bounding box.

The predictions for each grid cell are encoded as a tensor of dimensions $S \times S \times (B \cdot 5 + C)$. The network architecture typically consists of a convolutional backbone for feature extraction, which processes the input image and extracts features. These features are then passed through a series of fully connected layers that output the final predictions. The model is trained to minimize a loss function that combines the errors in bounding box coordinates, confidence scores, and class probabilities.

This loss function can be broken down into three components:

- **Localization Loss:** Measures the error in the predicted bounding box coordinates compared to the ground truth.
- **Confidence Loss:** Measures the error in the confidence score predictions.
- **Classification Loss:** Measures the error in the predicted class probabilities.

**Advantages of YOLO:**

- **Speed:** One of the most significant advantages of YOLO is its speed. By framing object detection as a single regression problem, YOLO processes images in a single forward pass through the network, making it exceptionally fast. This allows YOLO to be used in real-time applications, such as, surveillance cameras where speed is critical.
- **High Accuracy:** YOLO achieves high accuracy, particularly for detecting large objects within images. Its ability to see the entire image during training helps the model understand the contextual information better, leading to improved accuracy in object localization and classification.
- **Generalization:** YOLO generalizes well to new domains and can be applied to various tasks with minimal retraining. Its design allows it to perform well across different types of images and detection tasks, making it a versatile tool in the field of computer vision.

Over the years, different iterations of YOLO have been developed to address various challenges and improve performance. Here is a brief overview of these iterations:

- **YOLOv1:** The initial version introduced the concept of framing object detection as a single regression problem. It provided a good balance between speed and accuracy but struggled with detecting small objects and accurately localizing bounding boxes [18].
- **YOLOv2:** This version improved detection capabilities by introducing batch normalization, high-resolution classifier, and the use of anchor boxes [27].
- **YOLOv3:** It brought significant enhancements in feature extraction by adopting a deeper architecture with residual connections. It also improved the detection of small objects by using feature maps at three different scales for predictions [28].
- **YOLOv4:** It focused on further increasing speed and accuracy by incorporating additional innovations like Cross-Stage Partial (CSP) connections, Mish activation function, and self-adversarial training. These changes resulted in a more robust and efficient model [29].

The YOLO models were trained on a large-scale datasets COCO (Common Objects in Context) [30], which contains over 330,000 images, including more than 200,000 labelled images with over 80 object categories. This dataset is designed to represent a wide range of everyday scenes, making it ideal for training object detection models. Some example classes in the COCO dataset include people, bicycles, cars, traffic lights, fire hydrants, and various animals like dogs, cats, and birds.

While the COCO dataset provides a broad range of object categories, it does not include specific classes related to railway assets. To adapt the YOLO model for detecting these specialized objects, transfer learning is employed. Transfer learning involves taking a pre-trained

model, such as YOLO trained on the COCO dataset, and fine-tuning it on a new, domain-specific dataset. This approach leverages the learned features from the large-scale dataset and adapts them to recognize the new classes more effectively.

In the context of detecting railroad assets, a new dataset, comprising high-resolution images of railway components, needs to be created. This dataset must be annotated accurately to indicate the locations and categories of the railway assets. The pre-trained YOLO model is then fine-tuned on this specialized dataset, allowing it to learn the specific features and characteristics of the railway assets. By using transfer learning, the model can achieve high accuracy and efficiency in detecting these new classes with relatively less training data compared to training a model from scratch.

Each iteration of YOLO has built upon the strengths of its predecessors while introducing new techniques to enhance performance. YOLOv9 , the latest iteration, continues this trend by addressing information loss and incorporating reversible functions, making it one of the most advanced object detection models available and the one used for this task.

## 5.2 YOLOv9

### 5.2.1 Architecture Overview

YOLOv9 builds upon the strengths of its predecessors while introducing significant innovations to address the challenges of deep learning. Its architecture consists of three primary components: the backbone, neck, and head, each playing a crucial role in the object detection process.

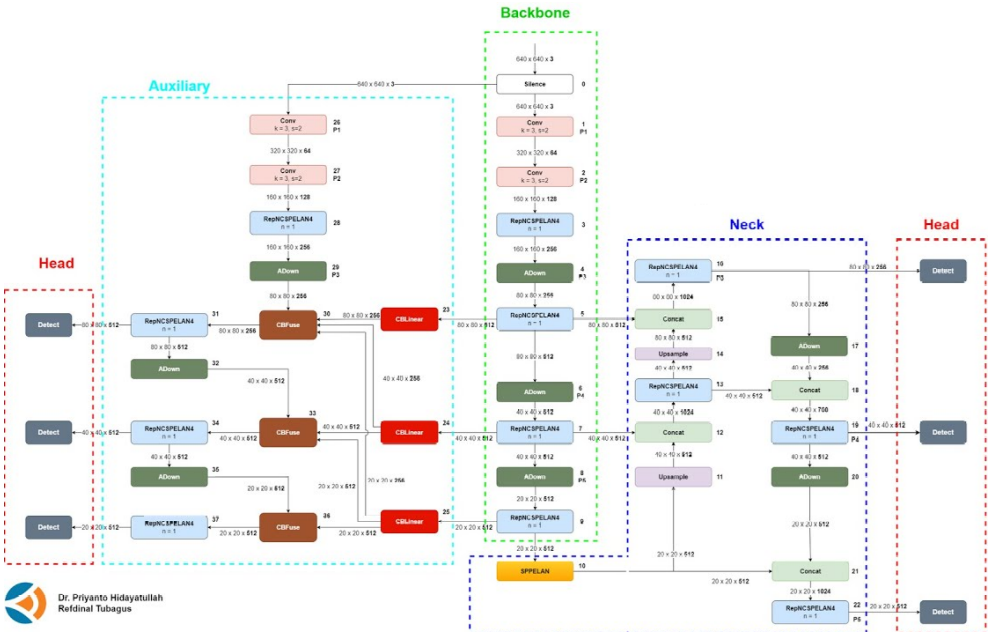



**Figure 5.1** The neural network structure of YOLOv9 [31].

## 5.3 Core Innovations of YOLOv9

YOLOv9's advancements are rooted in addressing the challenges posed by information loss in deep neural networks. The Information Bottleneck Principle and the innovative use of Reversible Functions are central to its design, ensuring YOLOv9 maintains high efficiency and accuracy. These innovations represent a key and revolutionary improvement over YOLOv8 [32], YOLOv7 [33], and older versions.

### 5.3.1 Information Bottleneck Principle

The Information Bottleneck Principle reveals a fundamental challenge in deep learning: as data passes through successive layers of a network, the potential for information loss increases. This phenomenon is mathematically represented as:

$$I(X, X) \geq I(X, f_\theta(X)) \geq I(X, g_\phi(f_\theta(X))) \tag{5.1}$$

where $I$ denotes mutual information, and $f$ and $g$ represent transformation functions with parameters $\theta$ and $\phi$, respectively. YOLOv9 counters this challenge by implementing Programmable Gradient Information (PGI), which aids in preserving essential data across the network's depth, ensuring more reliable gradient generation and, consequently, better model convergence and performance.

The implementation of PGI, by preserving more information through the network, is particularly revolutionary as it allows YOLOv9 to maintain higher accuracy and efficiency compared to YOLOv8, YOLOv7, and older versions.

### 5.3.2 Reversible Functions

The concept of Reversible Functions is another cornerstone of YOLOv9's design. A function is deemed reversible if it can be inverted without any loss of information, as expressed by:

$$X = v_\zeta(r_\psi(X)) \tag{5.2}$$

with $\psi$ and $\zeta$ as parameters for the reversible and its inverse function, respectively. This property is crucial for deep learning architectures, as it allows the network to retain a complete information flow, thereby enabling more accurate updates to the model's parameters. YOLOv9 incorporates reversible functions within its architecture to mitigate the risk of information degradation, especially in deeper layers, ensuring the preservation of critical data for object detection tasks.

This approach marks a significant advancement over previous YOLO versions, where information loss in deeper layers could compromise detection accuracy. By ensuring complete

information flow and enabling more accurate parameter updates, YOLOv9 achieves more reliable and precise object detection.

### 5.3.3 Backbone

The backbone serves as the feature extraction module, essential for the initial processing of input images. It employs a series of convolutional layers that vary in depth and complexity to extract salient features necessary for effective detection.

- **Initial Layers:** Images first pass through a series of convolutional layers that incrementally increase the depth from 64 to 1024 while reducing spatial dimensions. This step focuses on extracting low-level features such as edges, textures, and basic shapes, which are crucial for subsequent detection tasks.
- **RepNCSPELAN4 Blocks:** These blocks enhance feature recognition capabilities by capturing complex patterns and distinct object traits within the images. The RepNCSPELAN4 block is a variant of the CSP-ELAN architecture, which stands for Cross Stage Partial connections with Efficient Layer Aggregation Networks. CSP-ELAN architecture is designed to improve the gradient flow through the network by dividing the feature map into two parts, processing them separately, and then merging them. This approach helps in reducing computational cost and improving learning efficiency by reusing feature maps across different stages. The "Rep" prefix indicates the use of a re-parameterization technique to optimize the model for inference without changing its structure during training.
- **ADown Blocks:** These blocks are responsible for downsampling the feature maps while preserving important information, aiding in multi-scale feature extraction. Downsampling reduces the spatial resolution of the feature maps, which helps in capturing more abstract features at different scales. ADown blocks use techniques like strided convolutions to achieve this reduction while maintaining critical details needed for accurate detection.
- **SPPELAN Blocks:** SPPELAN blocks apply Spatial Pyramid Pooling (SPP) to the feature maps, which is a technique that allows the network to handle input images of varying sizes and scales. SPP divides the input feature map into different sized regions and pools each region separately, producing fixed-size output regardless of the input size. This aggregation of features at various scales enhances the network's ability to detect objects of different sizes and improves its robustness to changes in input dimensions. SPPELAN blocks normalize these pooled features to a consistent size, ensuring uniform inputs to subsequent layers.
- **Convolutional Layers:** In addition to the specialized blocks, the backbone includes convolutional layers that help refine the features extracted from the images.

### 5.3.4 Neck

The neck acts as an integration point for the features extracted by the backbone, employing upsampling and concatenation techniques to fuse detailed and semantic information effectively:

- **Upsampling:** This process uses nearest neighbour upsampling techniques to restore resolution to deeper feature maps, preserving crucial details necessary for detecting smaller objects. Mathematically, the upsampling process can be described as:

$$y_{ij} = x_{\lfloor \frac{i}{s} \rfloor \lfloor \frac{j}{s} \rfloor} \tag{5.3}$$

  where $y_{ij}$ is the upsampled output, $x$ is the input, and $s$ is the scaling factor.

- **Concatenation:** It merges feature maps from various depths. This feature set is pivotal for the model's high accuracy in object detection across scales. If $x$ and $y$ are two feature maps to be concatenated, the operation can be represented as:

$$z = \text{concat}(x, y) \tag{5.4}$$

  where $z$ is the concatenated output.

### 5.3.5 Auxiliary

The auxiliary components of the YOLOv9 architecture play a supporting role, enhancing the overall performance and stability of the model:

- **Silence Layers:** These layers are used to temporarily halt the gradient flow in certain parts of the network, which can help in stabilizing the training process and preventing gradient vanishing or exploding issues. This is particularly important because, although gradient vanishing and exploding issues are not as common in CNNs as in other types of neural networks, they can still occur in very deep CNN architectures like those used in YOLOv9.

- **CBLinear and CBFuse Blocks:** The CBLinear blocks are linear layers used for additional processing of feature maps, while the CBFuse blocks combine features from different layers and stages, ensuring that diverse information is passed through the network. These blocks contribute to better feature utilization and enhanced model performance.

### 5.3.6 Head

The head of YOLOv9 is dedicated to the detection task, utilizing the features to accurately predict object locations and characteristics:

- **Detection Layers:** These layers are tailored to predict multiple attributes of each detected object—bounding box coordinates, object class probabilities, and confidence scores. The detection layers perform the final predictions based on the processed feature maps.

- **Bounding Box Regression:** The bounding box coordinates are predicted as offsets relative to the anchor boxes. These offsets are adjusted to fit the ground truth objects more accurately. The bounding box regression is performed using the following equations:

$$\hat{t_x} = \frac{t_x - t_{xa}}{w_a} \tag{5.5}$$

$$\hat{t_y} = \frac{t_y - t_{ya}}{h_a} \tag{5.6}$$

$$\hat{t_w} = \log\left(\frac{w}{w_a}\right) \tag{5.7}$$

$$\hat{t_h} = \log\left(\frac{h}{h_a}\right) \tag{5.8}$$

  where $t_x, t_y, t_w, t_h$ are the centre coordinates, width, and height of the predicted bounding box, and $t_{xa}, t_{ya}, w_a, h_a$ are the centre coordinates, width, and height of the anchor box. The linear activation function is used for these predictions.

- **Objectness Score and Class Probability:** For each anchor box, the model predicts an objectness score and class probabilities.

  The objectness score indicates the likelihood that an anchor box contains an object and is predicted using the sigmoid activation function, which maps the output to a range between 0 and 1.

  The class probabilities represent the likelihood of each class for the detected object and are predicted using the softmax activation function, which converts the output into a probability distribution over the different classes.

- **Non-Maximum Suppression (NMS):** To remove redundant and overlapping bounding boxes, YOLOv9 applies Non-Maximum Suppression (NMS). This process selects the bounding box with the highest objectness score and suppresses all other overlapping boxes that have a lower score. The Intersection over Union (IoU) metric is used to measure the overlap between bounding boxes:

$$\text{IoU}(A, B) = \frac{A \cap B}{A \cup B} \tag{5.9}$$

  where $A$ and $B$ are two bounding boxes. NMS iteratively selects the bounding box with the highest score and suppresses all other boxes with IoU greater than a predefined threshold.

- **Final Output:** The final output consists of the bounding box coordinates, objectness scores, and class probabilities for each detected object. These outputs are used to draw the bounding boxes and labels on the input image.

## 5.4 Model Training

### 5.4.1 How YOLOv9 Works: Training on a New Dataset

The final step after the data preparation process is over involves training the model using the `train.py` script (Appendix F). This script sets up the training parameters, loads the dataset, and executes the training process using the YOLOv9 architecture. Leveraging GPU acceleration, it handles the computationally intensive tasks required for training a deep neural network.

The training process involves several stages:

- **Neural Network Structure and Parameters**: The initial phase involves configuring the neural network's architecture and setting the training parameters. This includes defining the number of layers, learning rates, batch sizes, dropout rates, and other critical hyperparameters. Properly tuning these parameters is essential for achieving optimal performance.
- **Initial Epochs and Pre-processing**: During the early epochs, the model starts analysing the provided images, adjusting weights and biases to minimize the error in predictions. This stage also includes preprocessing tasks like normalization and augmentation to enhance the model's learning capabilities. Data augmentation techniques, included within the model such as scaling, and flipping, help the model generalize better by exposing it to various forms of the training data.
- **Intermediate Epochs and Regularization**: As training progresses, regularization techniques like dropout are applied to prevent overfitting. Dropout randomly sets a fraction of the input units to zero at each update during training time, which helps in making the network more robust by preventing it from becoming too reliant on any single feature.
- **Final Epochs and Convergence**: In the later epochs, the model fine-tunes its parameters, striving for convergence. This stage focuses on achieving optimal accuracy and reducing the loss, ensuring the model is well-trained and ready for real-world applications. Batch size and the number of epochs play a crucial role here; a larger batch size provides a more stable estimate of the gradient, while an appropriate number of epochs ensures the model is sufficiently trained without overfitting.

By carefully splitting the dataset and leveraging advanced training techniques, we ensure that the YOLOv9 model is both accurate and robust.

#### Challenges

Training the YOLOv9 model presented several challenges, primarily related to the high computational demands of deep learning. Lower computational power meant a limited number of workers that could be utilized during training. This constraint directly affected the batch size

and image resolution, as larger batch sizes and higher resolution images require more memory and processing power. Consequently, finding the optimal balance between these parameters was a complex task, often requiring iterative experimentation.

### Results

The results of the model training were impressive, demonstrating the effectiveness of the YOLOv9 architecture in detecting railway assets. For a specific asset, the model during training demonstrated exceptional performance metrics, achieving a Precision of 1, Recall of 1, and a mAP50-95 of 0.977. These metrics indicate that the model achieved perfect precision and recall during training, meaning it correctly identified all instances of the asset without any false positives or false negatives. The mAP50-95 score, which measures the mean average precision across different IoU thresholds, further underscores the model's high accuracy and robustness.

These results are remarkable and suggest that the model has effectively learned to identify the targeted railway assets with high confidence and reliability. Subsequent evaluations and real-world applications of the model confirmed its robustness and accuracy. The model consistently delivered high-performance metrics, reinforcing the initial results and demonstrating its practical applicability in the field.

The integration of this model into a user-friendly web application further amplifies its utility, making advanced AI accessible to railway infrastructure managers without requiring deep technical knowledge. A more in-depth description of the web application and the results of each railway asset's model will be done in later chapters.

# Chapter 6

# Discussion of Results and Model Performance Analysis

## 6.1 Introduction

The implementation of YOLOv9 [22] for railway asset detection has demonstrated promising capabilities in identifying various assets from high-resolution aerial imagery. YOLOv9's advanced architecture and real-time processing capabilities make it particularly suitable for tasks requiring precise and rapid detection of railway components. For this study, individual models were developed for each specific asset type, ensuring tailored training and optimized detection performance for each category. This chapter provides a comprehensive analysis of the performance metrics of these models, including precision, recall, mean average precision (mAP), and other key indicators. By critically evaluating these metrics, we aim to understand the strengths and limitations of each model and identify potential areas for improvement.

This chapter is structured as follows: First, we introduce and explain the various performance metrics and key concepts used to evaluate the models. Next, we present detailed analyses of the detection tasks for five specific railway assets. For each asset, we provide an overview, the results obtained, and a discussion of the findings. Finally, we conclude with a synthesis of the overall performance and recommendations for future work.

## 6.2 Performance Metrics and Key Concepts

Understanding the metrics and key concepts used in model evaluation is crucial for interpreting the results accurately. Below, we provide detailed explanations of each metric and concept:

### 6.2.1 Loss Metrics

#### Box Loss

Box loss measures the error in predicting the bounding box coordinates for detected objects. It indicates how well the model can localize objects within an image. Lower box loss values suggest better localization accuracy.

The bounding box coordinates are predicted as offsets relative to the anchor boxes. These offsets are adjusted to fit the ground truth objects more accurately. The bounding box regression is performed using the following equations:

$$\hat{t_x} = \frac{t_x - t_{xa}}{w_a} \tag{6.1}$$

$$\hat{t_y} = \frac{t_y - t_{ya}}{h_a} \tag{6.2}$$

$$\hat{t_w} = \log\left(\frac{w}{w_a}\right) \tag{6.3}$$

$$\hat{t_h} = \log\left(\frac{h}{h_a}\right) \tag{6.4}$$

where $t_x, t_y, t_w, t_h$ are the centre coordinates, width, and height of the predicted bounding box, and $t_{xa}, t_{ya}, w_a, h_a$ are the centre coordinates, width, and height of the anchor box. The linear activation function is used for these predictions.

#### Classification Loss (cls_loss)

Classification loss evaluates the error in predicting the class labels of detected objects. It reflects the model's ability to correctly classify different types of railway assets. Accurate classification is critical for effective asset management.

For each anchor box, the model predicts an objectness score and class probabilities. The objectness score indicates the likelihood that an anchor box contains an object and is predicted using the sigmoid activation function, which maps the output to a range between 0 and 1. The class probabilities represent the likelihood of each class for the detected object and are predicted using the softmax activation function, which converts the output into a probability distribution over the different classes.

#### Distribution Focal Loss (dfl_loss)

Distribution focal loss is a specialized loss function that focuses on improving the precision of object localization by emphasizing hard-to-detect objects. This loss helps the model to better identify and localize challenging objects within the dataset.

### 6.2.2 Confusion Matrix Terms

#### True Positive (TP)

Instances where the model correctly predicts the presence of an object.

#### False Positive (FP)

Instances where the model incorrectly predicts the presence of an object.

#### True Negative (TN)

Instances where the model correctly predicts the absence of an object.

#### False Negative (FN)

Instances where the model incorrectly predicts the absence of an object.

**Table 6.1** Confusion Matrix

| | | Predicted | |
|---|---|---|---|
| | | Negative | Positive |
| Actual | Negative | TN | FP |
| | Positive | FN | TP |

### 6.2.3 Performance Metrics

In object detection, True Negatives (TN) are not well-defined because the background areas (where no objects are present) are not explicitly labelled and evaluated for the presence or absence of detections. Instead, the focus is on the presence of True Positives (TPs), False Positives (FPs), and False Negatives (FNs). As a result, metrics that rely on the accurate identification of TNs, such as accuracy, are not applicable in this context (Eq 6.5). While accuracy is a valuable metric for many machine learning models and AI applications, it can't be used for this analysis due to the lack of a clear definition for TNs in object detection [34].

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.5}$$

#### Mean Average Precision (mAP)

- **mAP50:** The mean average precision at an Intersection over Union (IoU) threshold of 50%. It measures the model's accuracy in detecting objects with at least 50% overlap with the ground truth. This metric provides an indication of the model's ability to make correct detections.

- **mAP50-95:** The mean average precision averaged over IoU thresholds ranging from 50% to 95%. This provides a more comprehensive evaluation of the model's detection performance across different levels of overlap with the ground truth.

#### Precision

Precision is the ratio of true positive predictions to the total number of positive predictions (true positives and false positives). It indicates how many of the predicted positive instances are actually correct. High precision means that the model makes few false positive errors.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.6}$$

#### Recall

Recall is the ratio of true positive predictions to the total number of actual positive instances (true positives and false negatives). It reflects the model's ability to identify all relevant instances in the dataset. High recall indicates that the model detects most actual positives, making few false negative errors.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.7}$$

#### Precision (B) and Recall (B)

Precision (B) and Recall (B) refer to the precision and recall metrics tracked during the model training process, evaluated on a separate validation set within the training epochs. These metrics provide insights into the model's learning progress and help in monitoring overfitting or underfitting trends.

#### F1-Score

The F1-score is the harmonic mean of precision and recall. It balances the trade-off between precision and recall, providing a single metric to evaluate the model's performance. A high F1-score indicates that the model has a good balance between precision and recall.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.8}$$

### 6.2.4 Key Concepts

#### Overfitting

Overfitting occurs when a model learns the training data too well, including its noise and outliers, resulting in poor performance on new, unseen data. Techniques such as early stopping, regularization, and cross-validation are used to mitigate overfitting.

### Retraining

Retraining involves updating a model with new data to improve its performance and generalization capabilities. This process can help the model adapt to new patterns and variations in the data, enhancing its overall effectiveness.

## 6.3 Wooden Sleepers

Wooden Sleepers (Figure 6.1), are essential components of the railway track structure. They support the rails and maintain the correct gauge, ensuring the stability and safety of the railway. Wooden Sleepers are used in many older railway tracks and are valued for their durability and shock-absorbing properties.

**Figure 6.1** Example of Wooden Sleepers

### 6.3.1 Model Training

A smaller dataset was derived from the complete dataset to train a model specifically for detecting wooden railway sleepers. The following plots show the results from the model training:

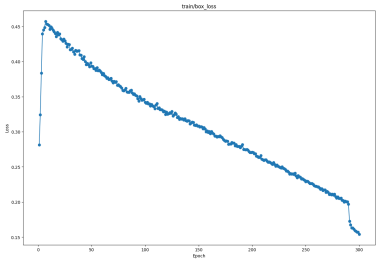


**Figure 6.2** Train Box Loss over Epochs

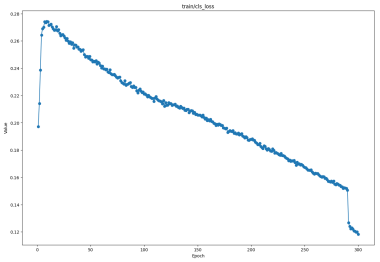


**Figure 6.3** Train Classification Loss over Epochs

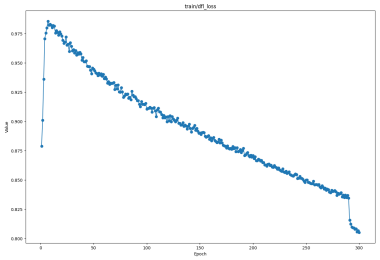


**Figure 6.4** Train Distribution Focal Loss over Epochs

Figures 6.2, 6.3, and 6.4 show that the training loss decreases as the number of epochs increases. This indicates that the model is learning from the training dataset effectively. The consistent downward trend suggests that the model still has the potential to learn more, although care must be taken to avoid overfitting.

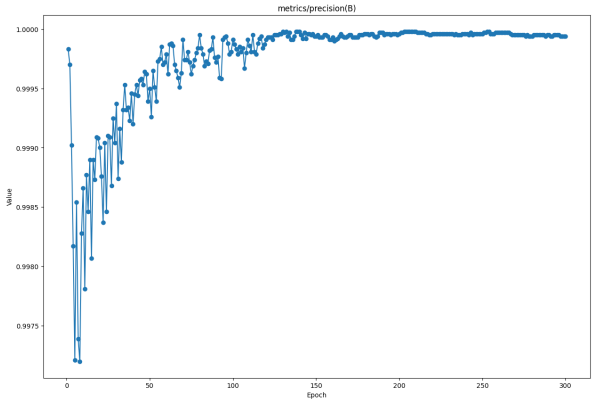



**Figure 6.5** Precision (B) over Epochs

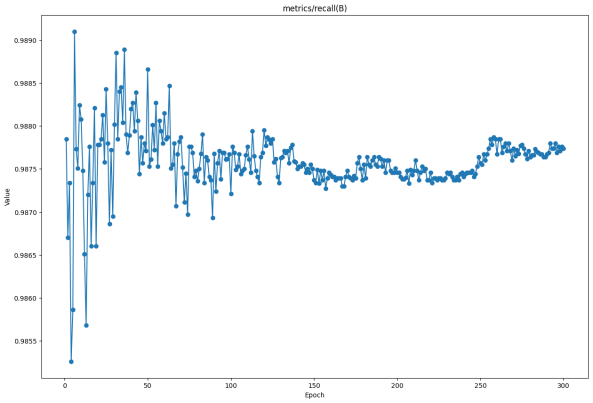



**Figure 6.6** Recall (B) over Epochs

Figures 6.5 and 6.6 indicate a significant variance in these metrics during the initial epochs. However, as training progresses, the model stabilizes with final values of 0.9999 for precision and 0.9877 for recall. This stability demonstrates the model's convergence and reliable performance in distinguishing between true positives and false positives.

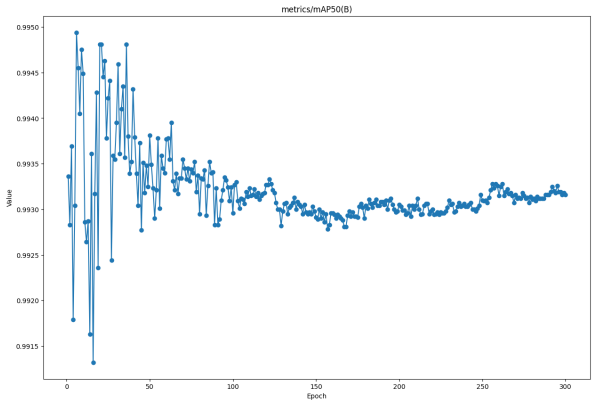



**Figure 6.7** mAP50 (B) over Epochs

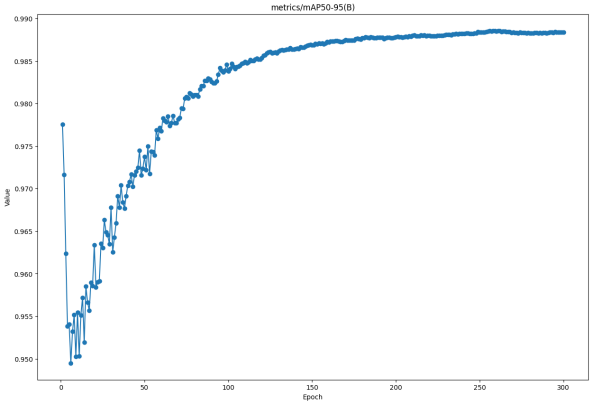



**Figure 6.8** mAP50-95 (B) over Epochs

Figures 6.7 and 6.8 exhibit similar convergence patterns for mean average precision. The final values of 0.9932 for mAP50 and 0.9883 for mAP50-95 reflect the model's high accuracy in detecting objects with varying levels of overlap with the ground truth.

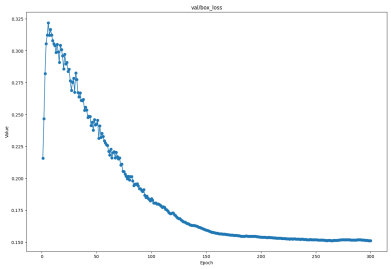


**Figure 6.9** Train Box Loss over Epochs

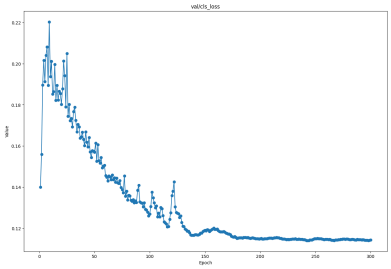


**Figure 6.10** Train Classification Loss over Epochs

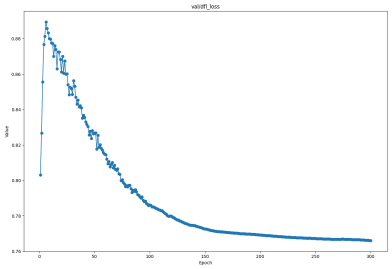


**Figure 6.11** Train Distribution Focal Loss over Epochs

Figures 6.9, 6.10, and 6.11 show the validation losses over epochs. These metrics stagnate at certain values, suggesting that the model may not improve much further without overfitting. This stagnation indicates the need for techniques such as early stopping or regularization to prevent overfitting while maintaining the model's generalization capabilities.

### 6.3.2 Model Testing

After evaluating the model's performance metrics during training, the model was tested on the entire dataset to assess its ability to detect wooden sleepers. The following performance metrics were calculated:

**Table 6.2** Performance Metrics for Wooden Sleepers

| **Metric** | **Value** |
|---|---|
| Total number of predictions | 3094 |
| Number of true positive predictions | 2863 |
| Number of false positive predictions | 231 |
| Number of false negative predictions | 137 |
| Precision | 0.9543 |
| Recall | 0.9253 |
| F1 Score | 0.9396 |

**Table 6.3** Confusion Matrix for Wooden Sleepers

| | | Predicted | |
|---|---|---|---|
| | | Negative | Positive |
| Actual | Negative | - | 231 |
| | Positive | 137 | 2863 |

Analysing the confusion matrix, we see that there are 231 false positives and 137 false negatives. This breakdown provides significant insights into the model's performance in detecting

wooden sleepers:

#### True Positives and False Negatives

With 2863 true positives and only 137 false negatives, the model successfully detected almost every instance of wooden sleepers. This demonstrates the model's high recall and ability to identify the presence of wooden sleepers accurately in most cases.

#### False Positives

The 231 false positives, in the context of over 3000 predictions, indicate that the model made relatively few mistakes. This low number of false positives highlights the model's precision and reinforces its reliability in distinguishing wooden sleepers from other objects.

#### Performance Emphasis

The high precision (0.9543) and F1 score (0.9396) further emphasize the model's strong performance. These metrics collectively show that the model is proficient in both detecting wooden sleepers and minimizing incorrect predictions.

Additionally, it is important to note that this model was retrained using a new dataset based on the results from the previous version. The retraining process involved picking up from where the last dataset left off and training on additional images. This new dataset was specifically designed to address the false positives encountered previously and to boost the number of true positives, further enhancing the model's performance.

### 6.3.3 Examples of Predictions

## False Positives

The following examples illustrate false positive predictions, demonstrating the model's precision and the challenges in distinguishing wooden sleepers:

- **Example 1:** An image where an object resembling the shape of a wooden sleeper was predicted as a sleeper. This showcases that, visually, such instances can be mistaken for wooden sleepers, validating the model's high precision.


**Figure 6.12** False Positive Example 1 for wooden sleeper

- **Example 2:** Another material that has similar features to a wooden sleeper. The similarity in appearance explains why the model identified it as a wooden sleeper, reflecting its sensitivity to potential wooden sleeper characteristics.


**Figure 6.13** False Positive Example 2 for wooden sleeper

### False Negative

The following example illustrates a false negative prediction, highlighting the variability in the dataset and the challenges for the model:

- **Example 3:** An image where a wooden sleeper is present but was not detected due to partial occlusion. This case underscores the inherent variability in the dataset and the difficulty for the model to recognize all possible instances of wooden sleepers.

**Figure 6.14** False Negative Example for wooden sleeper

In summary, the high precision and F1 score underscore the model's effectiveness at detecting wooden sleepers in an extensive railway line of over 30 km, emphasizing a few examples of errors showcasing that the model makes reasonable mistakes, validating its overall good performance. Figure 6.15 showcases this result.

**Figure 6.15** Example of the model detecting a wooden sleeper

## 6.4 Track Clearance Marker

The Track Clearance Marker (Figure 6.16) is a crucial marker used in railway systems to indicate the boundary beyond which rolling stock should not be parked to avoid collisions. This limit point, always present where a railway splits into two. It ensures safe operations by preventing converging movements from colliding.

**Figure 6.16** Example of a Track Clearance Marker

### 6.4.1 Model Training

A smaller dataset was derived from the complete dataset to train a model specifically for detecting track clearance markers. The following plots show the results from the model training:

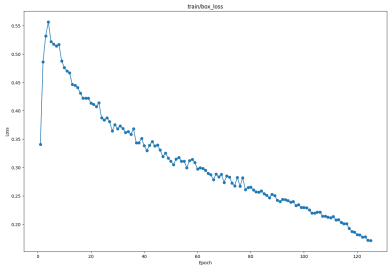


**Figure 6.17** Train Box Loss over Epochs

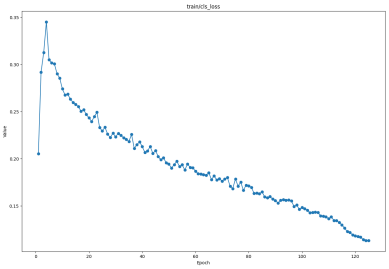


**Figure 6.18** Train Classification Loss over Epochs

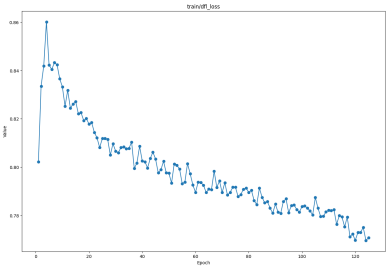


**Figure 6.19** Train Distribution Focal Loss over Epochs

Figures 6.17, 6.18, and 6.19 show that the training loss decreases as the number of epochs increases. This indicates that the model is learning effectively from the training data, with the potential to improve further as the training continues.

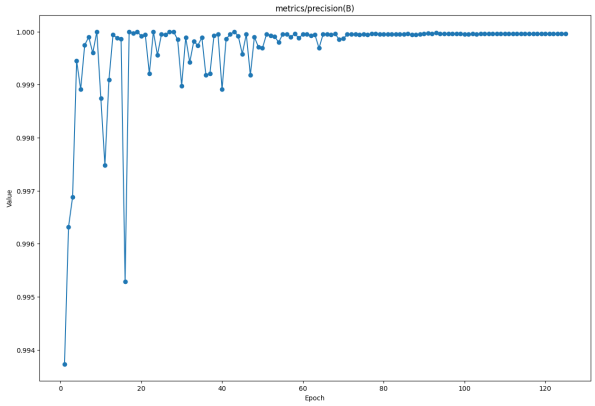



**Figure 6.20** Precision (B) over Epochs

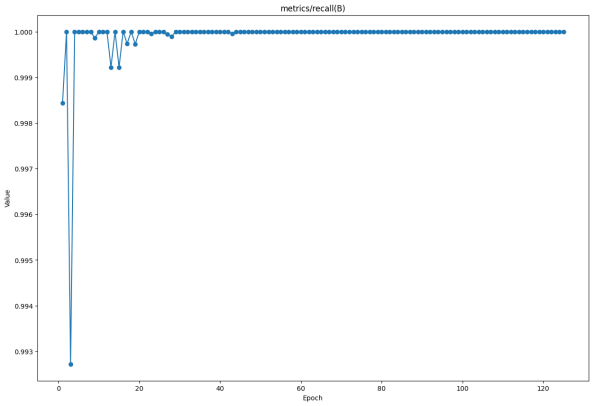



**Figure 6.21** Recall (B) over Epochs

Figures 6.20 and 6.21 indicate a significant variance in these metrics during the initial epochs, which stabilizes as the model converges. The final values for Precision and Recall are 1.0000 and 1.0000, respectively, demonstrating high accuracy and coverage of the model.

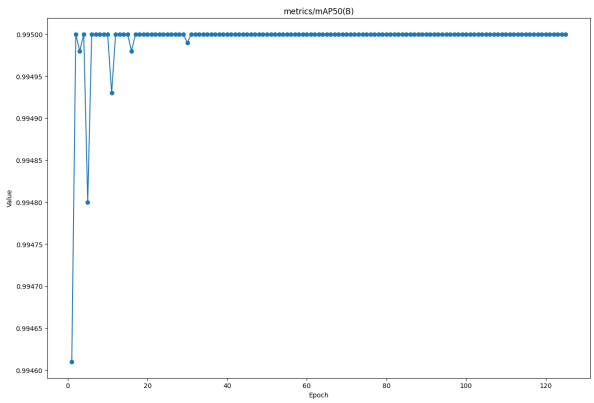



**Figure 6.22** mAP50 (B) over Epochs

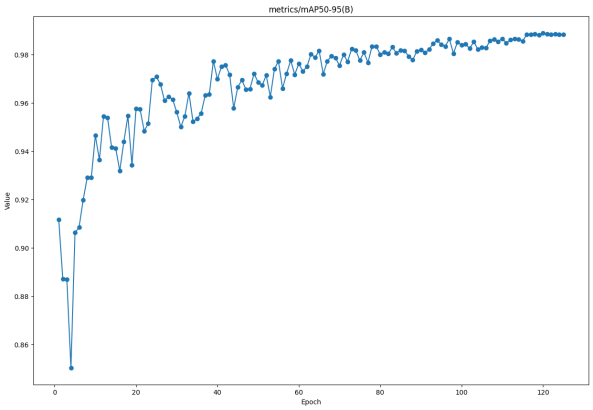



**Figure 6.23** mAP50-95 (B) over Epochs

Figures 6.22 and 6.23 show that the model's detection performance converges with final values of 0.9950 and 0.9883, respectively. This comprehensive evaluation indicates the model's high accuracy in detecting objects across various IoU thresholds.

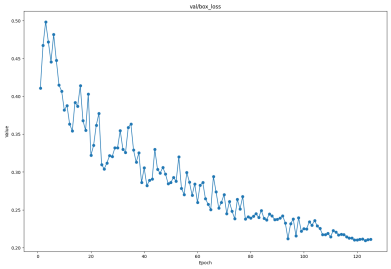


**Figure 6.24** Validation Box Loss over Epochs

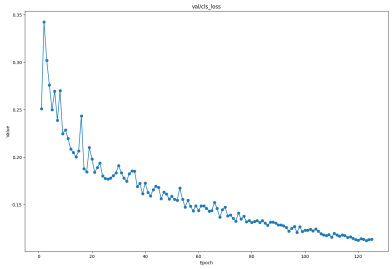


**Figure 6.25** Validation Classification Loss over Epochs

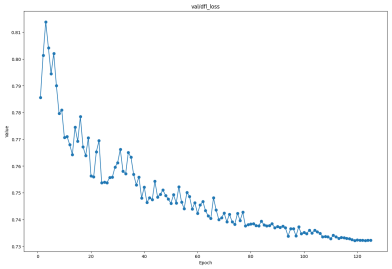


**Figure 6.26** Validation Distribution Focal Loss over Epochs

Figures 6.24, 6.25, and 6.26 reveal that the validation metrics stagnate at a certain value, indicating that the model may not learn much more without overfitting.

### 6.4.2 Model Testing

After evaluating the model's performance metrics during training, the model was tested on the entire dataset to assess its ability to detect Track Clearance Markers. The following performance metrics were calculated:

**Table 6.4** Performance Metrics for Track Clearance Marker

| **Metric** | **Value** |
|---|---|
| Total number of predictions | 22 |
| Number of true positive predictions | 20 |
| Number of false positive predictions | 2 |
| Number of false negative predictions | 1 |
| Precision | 0.9524 |
| Recall | 0.9091 |
| F1 Score | 0.9302 |

**Table 6.5** Confusion Matrix for Track Clearance Marker

| | | Predicted | |
|---|---|---|---|
| | | Negative | Positive |
| Actual | Negative | - | 2 |
| | Positive | 1 | 20 |

Analysing the confusion matrix, we see that there are 2 false positives and 1 false negative. This breakdown provides significant insights into the model's performance in detecting the Track Clearance Markers:

#### True Positives and False Negatives

With 20 true positives and only 1 false negative, the model successfully detected almost every instance of the track clearance markers. This demonstrates the model's high recall and ability to identify the presence of clearance markers accurately in most cases.

#### False Positives

The 2 false positives, in the context of 22 predictions, indicate that the model made relatively few mistakes. This low number of false positives highlights the model's precision and reinforces its reliability in distinguishing clearance markers from other objects.

#### Performance Emphasis

The high precision (0.9524) and F1 score (0.9302) further emphasize the model's strong performance. These metrics collectively show that the model is proficient in both detecting the track clearance markers and minimizing incorrect predictions.

Additionally, it is important to note that this model was retrained using a new dataset based on the results from the previous version. The retraining process involved picking up from where the last dataset left off and training on additional images. This new dataset was specifically designed to address the false positives encountered previously and to boost the number of true positives, further enhancing the model's performance.

### 6.4.3 Examples of Predictions

### False Positives

The following examples illustrate false positive predictions, demonstrating the model's precision and the challenges in distinguishing clearance markers:

- **Example 1:** An image where an object resembling the shape of a clearance marker was predicted as a marker. This showcases that, visually, such instances can be mistaken for clearance markers, validating the model's high precision.

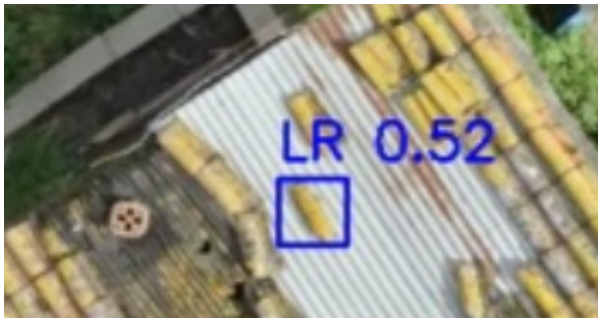



**Figure 6.27** False Positive Example 1 for Track Clearance Markers

- **Example 2:** Another material that has similar features to a clearance marker. The similarity in appearance explains why the model identified it as a clearance marker, reflecting its sensitivity to potential clearance marker characteristics.


**Figure 6.28** False Positive Example 2 for Track Clearance Markers

### False Negative

The following example illustrates a false negative prediction, highlighting the variability in the dataset and the challenges for the model:

- **Example 3:** An image where a clearance marker is present but was not detected due to partial occlusion. This case underscores the inherent variability in the dataset and the difficulty for the model to recognize all possible instances of track clearance markers.

**Figure 6.29** False Negative Example for Track Clearance Markers

In summary, the high precision and F1 score underscore the model's effectiveness at detecting the track clearance markers in an extensive railway line of over 30 km, emphasizing a few examples of errors showcasing that the model makes reasonable mistakes, validating its overall good performance. Figure 6.30 showcases this result.


**Figure 6.30** Example of the model detecting a Track Clearance Markers

## 6.5 Railway Switch - Track

A railway switch, also known as a turnout, consists of rails, switch blades, and other mechanical parts, designed to ensure the tangential connection of two tracks, allowing trains to move from one track to another (Figure 6.31).

**Figure 6.31** Example of Railway Switch

### 6.5.1 Model Training

A smaller dataset was derived from the complete dataset to train a model specifically for detecting railway switches. The following plots show the results from the model training:

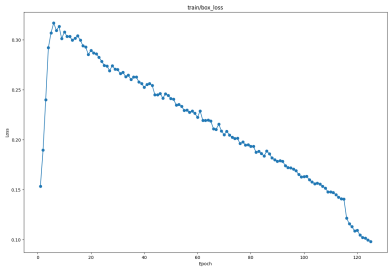



**Figure 6.32** Train Box Loss over Epochs

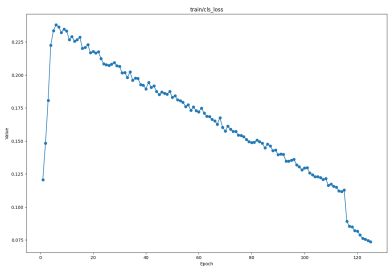



**Figure 6.33** Train Classification Loss over Epochs

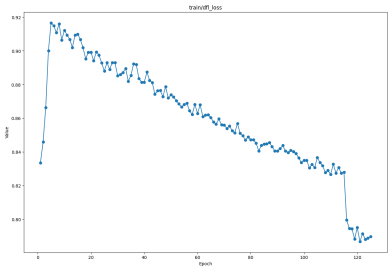



**Figure 6.34** Train Distribution Focal Loss over Epochs

Figures 6.32, 6.33, and 6.34 show that the training loss decreases as the number of epochs increases. This indicates that the model is learning effectively from the training data, with the potential to improve further as the training continues.

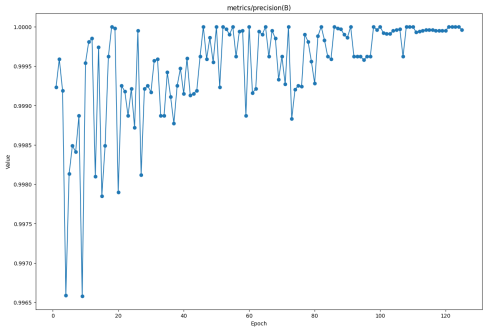



**Figure 6.35** Precision (B) over Epochs

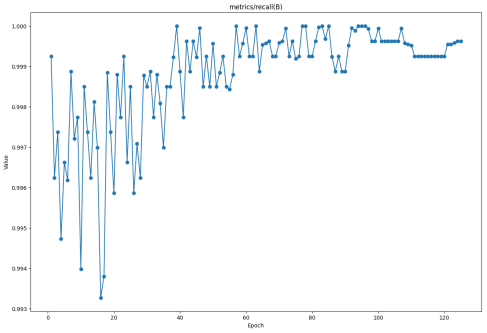



**Figure 6.36** Recall (B) over Epochs

Figures 6.35 and 6.36 indicate a significant variance in these metrics during the initial epochs, which stabilizes as the model converges. The final values for Precision and Recall are 1.0000 and 0.9999, respectively, demonstrating high accuracy and coverage of the model.

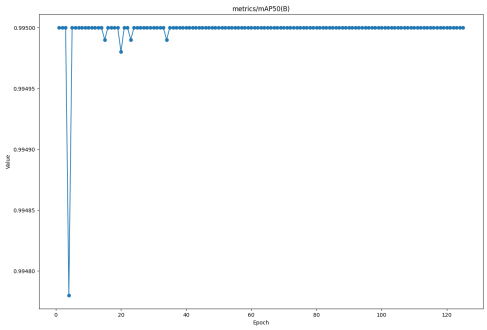



**Figure 6.37** mAP50 (B) over Epochs

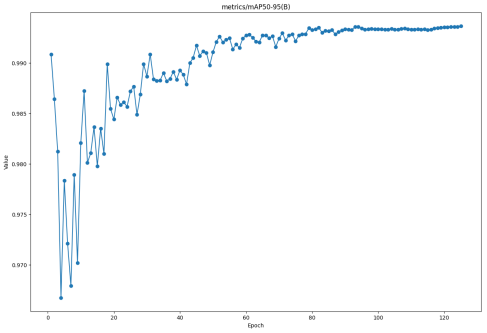



**Figure 6.38** mAP50-95 (B) over Epochs

Figures 6.37 and 6.38 show that the model's detection performance converges with final values of 0.9950 and 0.9933, respectively. This comprehensive evaluation indicates the model's high accuracy in detecting objects across various IoU thresholds.

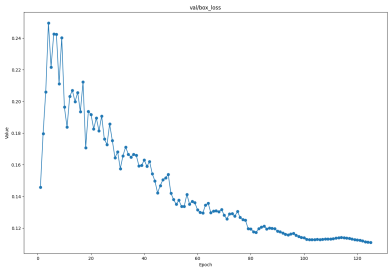


**Figure 6.39** Validation Box Loss over Epochs

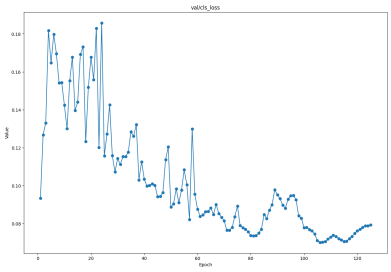


**Figure 6.40** Validation Classification Loss over Epochs

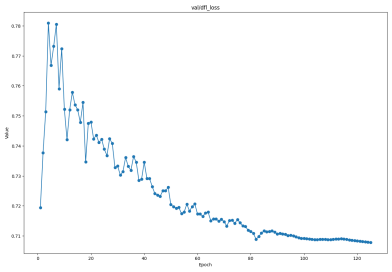


**Figure 6.41** Validation Distribution Focal Loss over Epochs

Figures 6.39, 6.40, and 6.41 reveal that the validation metrics stagnate at a certain value, indicating that the model may not learn much more without overfitting.

### 6.5.2 Model Testing

After evaluating the model's performance metrics during training, the model was tested on the entire dataset to assess its ability to detect railway switches. The following performance metrics were calculated:

**Table 6.6** Performance Metrics for Railway Switch - Track

| **Metric** | **Value** |
|---|---|
| Total number of predictions | 28 |
| Number of true positive predictions | 27 |
| Number of false positive predictions | 1 |
| Number of false negative predictions | 1 |
| Precision | 0.9643 |
| Recall | 0.9643 |
| F1 Score | 0.9643 |

**Table 6.7** Confusion Matrix for Railway Switch - Track

| | | Predicted | |
|---|---|---|---|
| | | Negative | Positive |
| Actual | Negative | - | 1 |
| | Positive | 1 | 27 |

Analysing the confusion matrix, we see that there is 1 false positive and 1 false negative. This breakdown provides significant insights into the model's performance in detecting railway switches:

#### True Positives and False Negatives

With 27 true positives and only 1 false negative, the model successfully detected almost every instance of the railway switches. This demonstrates the model's high recall and ability to identify the presence of switches accurately in most cases.

#### False Positives

The 1 false positive, in the context of 28 predictions, indicates that the model made very few mistakes. This low number of false positives highlights the model's precision and reinforces its reliability in distinguishing railway switches from other objects.

#### Performance Emphasis

The high precision (0.9643) and F1 score (0.9643) further emphasize the model's strong performance. These metrics collectively show that the model is proficient in both detecting railway switches and minimizing incorrect predictions.

Additionally, it is important to note that this model was retrained using a new dataset based on the results from the previous version. The retraining process involved picking up from where the last dataset left off and training on additional images. This new dataset was specifically designed to address the false positives encountered previously and to boost the number of true positives, further enhancing the model's performance.

### 6.5.3 Examples of Predictions

### False Positives

The following example illustrates a false positive prediction, demonstrating the model's precision and the challenges in distinguishing railway switches:

- **Example 1:** An image where an object resembling the shape of a railway switch was predicted as a switch. This showcases that, visually, such instances can be mistaken for switches, validating the model's high precision.


**Figure 6.42** False Positive Example for railway switch

### False Negatives

The following example illustrates a false negative prediction, highlighting the variability in the dataset and the challenges for the model:

- **Example 2:** An image where a railway switch is present but was not detected due to a partial blur of the image with also a change of background since it's not present in gravel. This case underscores the inherent variability in the dataset and the difficulty for the model to recognize all possible instances of railway switches.

**Figure 6.43** False Negative Example for railway switch

In summary, the high precision and F1 score underscore the model's effectiveness at detecting railway switches in an extensive railway line of over 30 km, emphasizing a few examples of errors showcasing that the model makes reasonable mistakes, validating its overall good performance. Figure 6.44 showcases that.

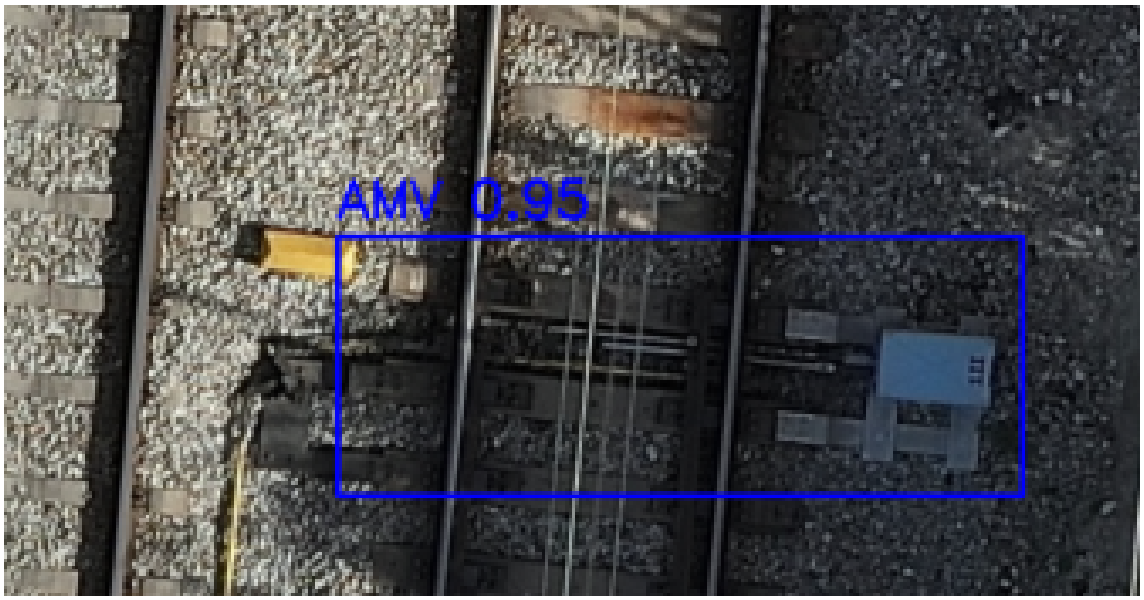



**Figure 6.44** Example of model detecting a Railway Switch

## 6.6 Transponders - Track

Automatic Speed Control Equipment installed on the track axis, containing variable information usually from lateral signalling and fixed predefined information. The information from the track is collected by the driving unit when passing over the transponders, preventing the trains from exceeding the speeds imposed by safety regulations.

**Figure 6.45** Example of Transponders

### 6.6.1 Model Training

A smaller dataset was derived from the complete dataset to train a model specifically for detecting transponders. The following plots show the results from the model training:

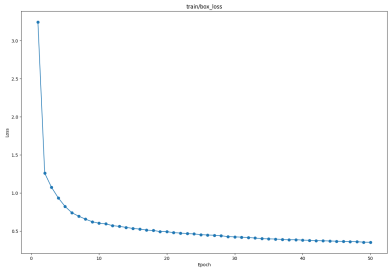



**Figure 6.46** Train Box Loss over Epochs

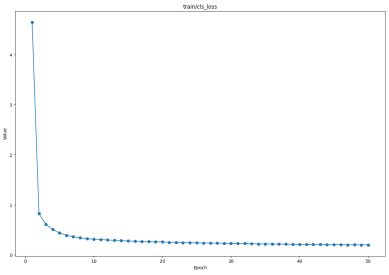



**Figure 6.47** Train Classification Loss over Epochs

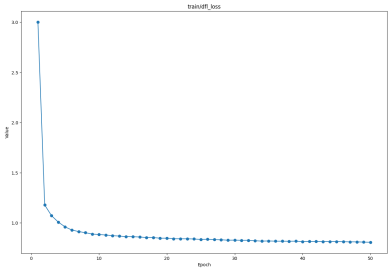



**Figure 6.48** Train Distribution Focal Loss over Epochs

Figures 6.46, 6.47, and 6.48 show that the training loss decreases as the number of epochs increases. It stagnates after a few epochs, indicating the potential of overfitting if further training is done.

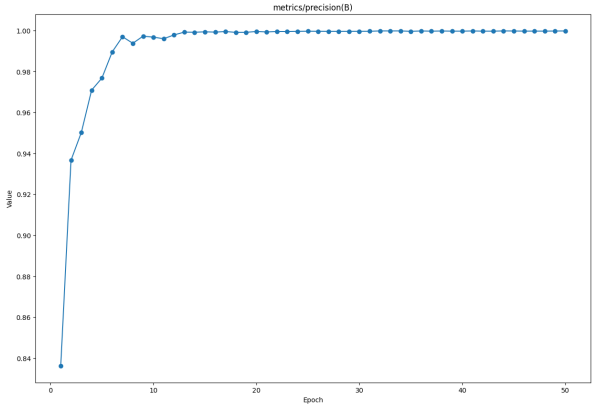



**Figure 6.49** Precision (B) over Epochs

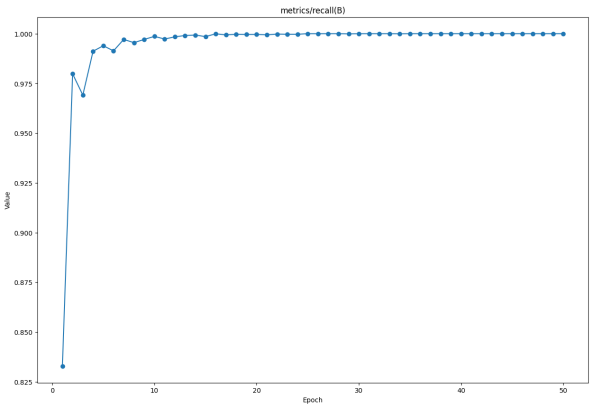



**Figure 6.50** Recall (B) over Epochs

Figures 6.49 and 6.50 indicate a significant increase in these metrics during the initial epochs, which stabilizes as the model converges. The final values for Precision and Recall are 0.9998 and 1.0000, respectively, demonstrating high accuracy and coverage of the model.

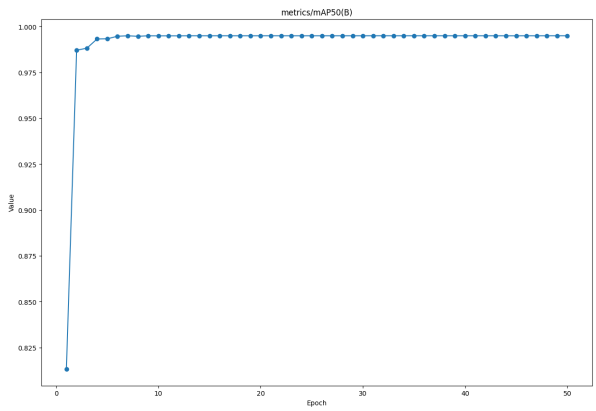



**Figure 6.51** mAP50 (B) over Epochs

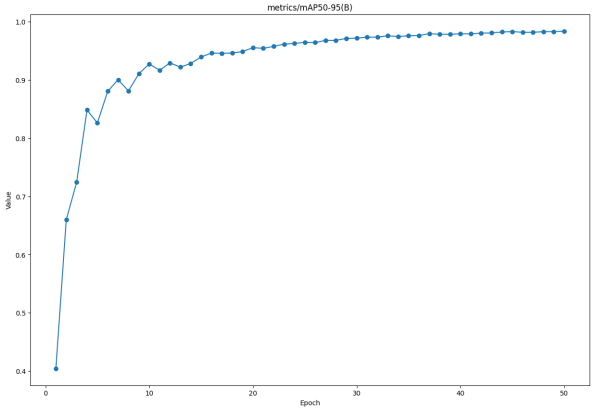



**Figure 6.52** mAP50-95 (B) over Epochs

Figures 6.51 and 6.52 show that the model's detection performance converges with final values of 0.9950 and 0.9837, respectively. This comprehensive evaluation indicates the model's high accuracy in detecting objects across various IoU thresholds.

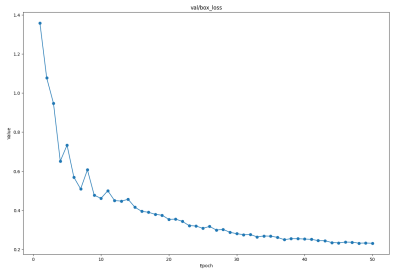


**Figure 6.53** Validation Box Loss over Epochs

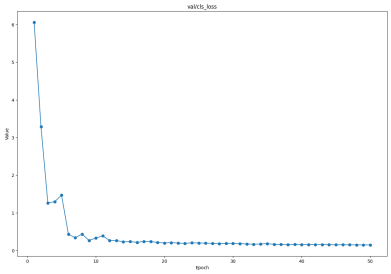


**Figure 6.54** Validation Classification Loss over Epochs

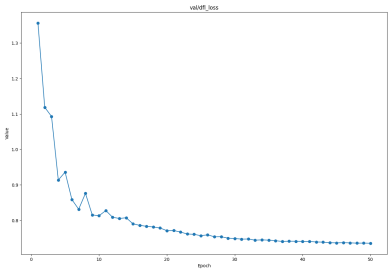


**Figure 6.55** Validation Distribution Focal Loss over Epochs

Figures 6.53, 6.54, and 6.55 reveal that the validation metrics stagnate at a certain value, indicating that the model may not learn much more without overfitting.

### 6.6.2 Model Testing

After evaluating the model's performance metrics during training, the model was tested on the entire dataset to assess its ability to detect transponders. The following performance metrics were calculated:

**Table 6.8** Confusion Matrix for Transponders

| | | Predicted | |
|---|---|---|---|
| | | Negative | Positive |
| Actual | Negative | - | 5 |
| | Positive | 6 | 276 |

**Table 6.9** Performance Metrics for Transponders

| Metric | Value |
|---|---|
| Total number of predictions | 281 |
| Number of true positive predictions | 276 |
| Number of false positive predictions | 5 |
| Number of false negative predictions | 6 |
| Precision | 0.9787 |
| Recall | 0.9822 |
| F1 Score | 0.9805 |

Analysing the confusion matrix, we see that there are 5 false positives and 6 false negatives. This breakdown provides significant insights into the model's performance in detecting the railway asset:

#### True Positives and False Negatives

With 276 true positives and only 6 false negatives, the model successfully detected almost every instance of the asset. This demonstrates the model's high recall and ability to identify the presence of Transponders accurately in most cases.

#### False Positives

The 5 false positives, in the context of 281 predictions, indicate that the model made relatively few mistakes. This low number of false positives highlights the model's precision and reinforces its reliability in distinguishing Transponders from other objects.

#### Performance Emphasis

The high precision (0.9787) and F1 score (0.9805) further emphasize the model's strong performance. These metrics collectively show that the model is proficient in both detecting Transponders and minimizing incorrect predictions.

Additionally, it is important to note that this model was retrained using a new dataset based on the results from the previous version. The retraining process involved picking up from where the last dataset left off and training on additional images. This new dataset was specifically designed to address the false positives encountered previously and to boost the number of true positives, further enhancing the model's performance.

### 6.6.3 Examples of Predictions

#### False Positives

The following example illustrates a false positive prediction, demonstrating the model's precision and the challenges in distinguishing Transponders:

- **Example 1:** An image where an object resembling the shape of a transponder was predicted as a transponder. This showcases that, visually, such instances can be mistaken for transponders.


**Figure 6.56** False Positive Example for Transponders

#### False Negative

The following examples illustrate false negative predictions, highlighting the variability in the dataset and the challenges for the model:

- **Examples 2 and 3:** Images where a Transponder is present but was not detected due to partial colour fading. This case underscores the inherent variability in the dataset and the difficulty for the model to recognize all possible instances of Transponders.


**Figure 6.57** False Negative Example 1 for Transponders


**Figure 6.58** False Negative Example 2 for Transponders

In summary, the high precision and F1 score underscore the model's effectiveness at detecting Transponders in an extensive railway line of over 30 km. The examples of errors demonstrate that the model makes reasonable mistakes, validating its overall good performance. Figure 6.59 showcases this result, correctly detecting the asset while avoiding other yellow objects.


**Figure 6.59** Example of model detecting a Transponder

## 6.7 Concrete Twin-block Sleeper

Concrete Twin-block sleepers consist of two reinforced concrete blocks, each providing bearing surfaces for the rails, connected by a metal profile that ensures the gauge of the track (Figure 6.60). These sleepers offer high stability and durability, making them suitable for various railway applications.

**Figure 6.60** Example of Concrete Twin-block Sleeper

### 6.7.1 Model Training

A smaller dataset was derived from the complete dataset to train a model specifically for detecting concrete twin-block sleepers. The following plots show the results from the model training:

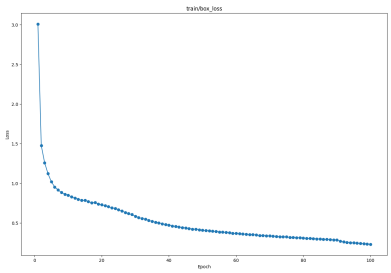



**Figure 6.61** Train Box Loss over Epochs

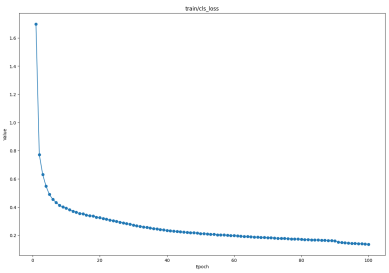



**Figure 6.62** Train Classification Loss over Epochs

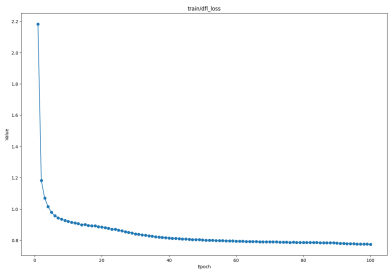



**Figure 6.63** Train Distribution Focal Loss over Epochs

Figures 6.61, 6.62, and 6.63 show that the training loss decreases as the number of epochs increases. This indicates that the model is learning effectively from the training data.

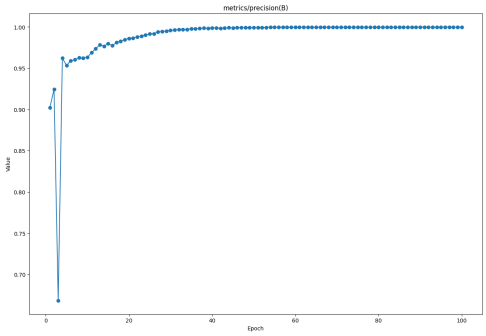



**Figure 6.64** Precision (B) over Epochs

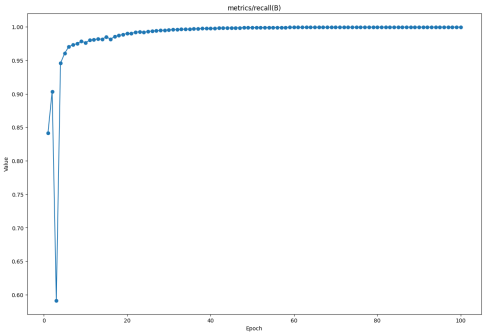



**Figure 6.65** Recall (B) over Epochs

Figures 6.64 and 6.65 indicate a significant variance in these metrics during the initial epochs, which stabilizes as the model converges. The final values for Precision and Recall are 0.9999 and 0.9996, respectively, demonstrating high accuracy and coverage of the model.

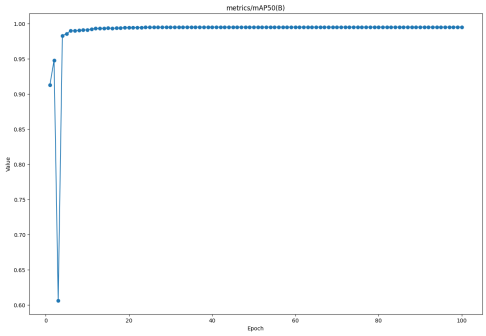



**Figure 6.66** mAP50 (B) over Epochs

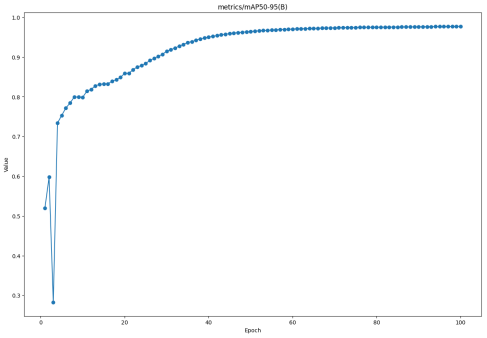



**Figure 6.67** mAP50-95 (B) over Epochs

Figures 6.66 and 6.67 show that the model's detection performance converges with final values of 0.9950 and 0.9774, respectively. This comprehensive evaluation indicates the model's high accuracy in detecting objects across various IoU thresholds.

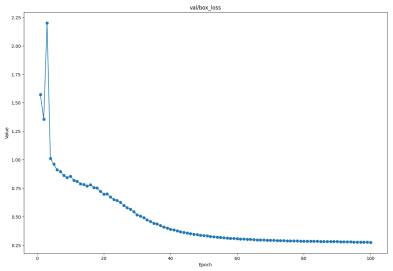


**Figure 6.68** Validation Box Loss over Epochs

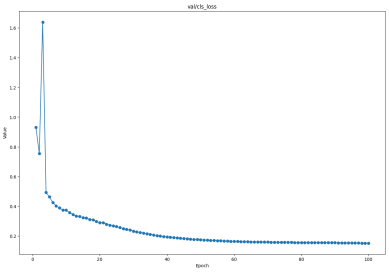


**Figure 6.69** Validation Classification Loss over Epochs

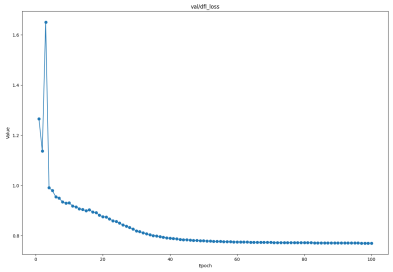


**Figure 6.70** Validation Distribution Focal Loss over Epochs

Figures 6.68, 6.69, and 6.70 reveal that the validation metrics stagnate at a certain value, indicating that the model may not learn much more without overfitting.

### 6.7.2 Model Testing

After evaluating the model's performance metrics during training, the model was tested on the entire dataset to assess its ability to detect concrete twin-block sleepers. The following performance metrics were calculated:

**Table 6.10** Performance Metrics for Concrete Twin-block Sleeper

| **Metric** | **Value** |
|---|---|
| Total number of predictions | 148368 |
| Number of true positive predictions | 147959 |
| Number of false positive predictions | 409 |
| Number of false negative predictions | 2477 |
| Precision | 0.9835 |
| Recall | 0.9972 |
| F1 Score | 0.9903 |

**Table 6.11** Confusion Matrix for Concrete Twin-block Sleeper

| | | Predicted | |
|---|---|---|---|
| | | Negative | Positive |
| Actual | Negative | - | 409 |
| | Positive | 2477 | 147959 |

Analysing the confusion matrix, we see that there are 409 false positives and 2477 false negatives. This breakdown provides significant insights into the model's performance in detecting concrete twin-block sleepers:

#### True Positives and False Negatives

With 147959 true positives and only 2477 false negatives, the model successfully detected almost every instance of concrete twin-block sleepers. This demonstrates the model's high recall and ability to identify the presence of sleepers accurately in most cases. The low number of false negatives indicates the model's effectiveness in recognizing the concrete twin-block sleepers, despite potential variations in the dataset.

#### False Positives

The 409 false positives, in the context of over 148000 predictions, indicate that the model made relatively few mistakes. This low number of false positives highlights the model's precision and reinforces its reliability in distinguishing concrete twin-block sleepers from other objects.

#### Performance Emphasis

The high precision (0.9835) and F1 score (0.9903) further emphasize the model's strong performance. These metrics collectively show that the model is proficient in both detecting concrete twin-block sleepers and minimizing incorrect predictions.

Additionally, it is important to note that this model was trained multiple times using a new dataset based on the results from the previous version. This new dataset was specifically designed to address the false positives encountered previously and to boost the number of true positives, further enhancing the model's performance.

### 6.7.3 Examples of Predictions

### False Positives

The following example illustrates a false positive prediction, demonstrating the model's precision and the challenges in distinguishing concrete twin-block sleepers:

- **Example 1:** An image where an object resembling the shape of a concrete twin-block sleeper was predicted as a sleeper. This showcases that, visually, such instances can be mistaken for concrete twin-block sleepers, validating the model's high precision.

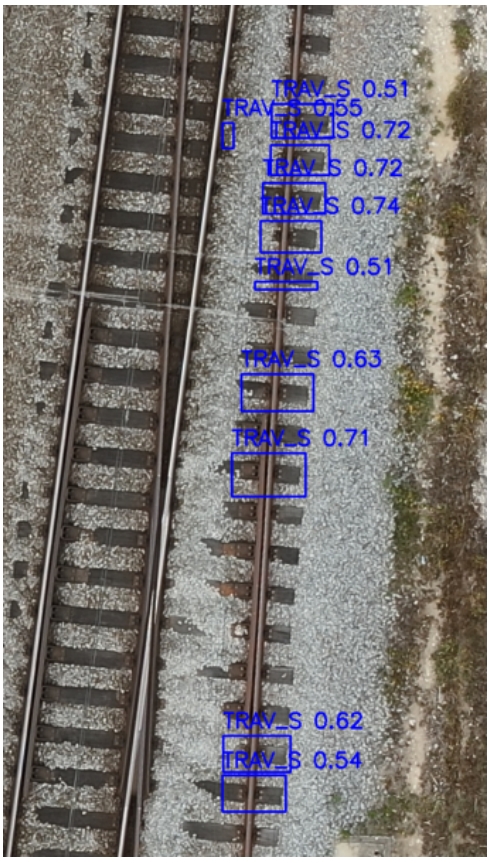



**Figure 6.71** False Positive Example for concrete twin-block sleeper

### False Negatives

The following example illustrates a false negative prediction, highlighting the variability in the dataset and the challenges for the model:

- **Example 2:** An image where a concrete twin-block sleeper is present but was not detected due to partial occlusion. This case underscores the inherent variability in the dataset and the difficulty for the model to recognize all possible instances of concrete twin-block sleepers.

**Figure 6.72** False Negative Example for concrete twin-block sleeper

In summary, the high precision and F1 score underscore the model's effectiveness at detecting concrete twin-block sleepers in an extensive railway line of over 30 km, emphasizing a few examples of errors showcasing that the model makes reasonable mistakes, validating its overall good performance. Figure 6.73 showcases this result.

**Figure 6.73** Example of model detecting multiple concrete twin-block sleepers

## 6.8 Conclusion

The implementation of YOLOv9 for railway asset detection has demonstrated significant potential in identifying various railway components from high-resolution aerial imagery. The models exhibit robust performances across different asset types, such as Wooden Sleepers, Track Clearance Markers, Railway Switches, Transponders, and Concrete Twin-block Sleepers. Each asset detection task provided insights into the strengths and limitations of YOLOv9, allowing for a comprehensive evaluation of its capabilities.

The varying number of observations for different asset types significantly influenced the models' performance metrics. For instance, the relatively small dataset for certain assets, like the Track Clearance Markers and Railway Switches, contributed to challenges in training due to

insufficient data, resulting in less reliable model. Conversely, the larger dataset for Concrete Twin-block Sleepers provided a more comprehensive training set, resulting in lower error rates and a more reliable model.

The characteristics of the assets themselves also played a crucial role in the models' performance. The distinct visual features of the Transponders and Track Clearance Markers, such as their yellow colour and specific shapes (square for Transponders and rectangular for clearance markers), made them easier for the models to learn and detect. In contrast, the variability in the appearance of Wooden Sleepers and Concrete Twin-block Sleepers, affected by factors such as occlusion by grass, damage, background clutter, and similarities with other assets, posed significant challenges. Having the most observations throughout the railway, the detection of Concrete Twin-block Sleepers achieved the highest precision of 98.35% and a recall of 99.72%. Wooden Sleepers, due to their size, occasional occlusions, and the particularity of being made of wood made it prone to be confused with wooden structures, despite this challenges it achieved a high performance with a precision of 95.43% and a recall of 92.53%. This highlights the complexity in training models for assets with high variability.

The Track Clearance Markers detection showed promising results, with a precision of 95.24% and a recall of 90.91%. Railway Switches detection demonstrated high performance, with a precision of 96.43% and a recall of 96.43%.Transponders detection achieved impressive performance with a precision of 97.87% and a recall of 98.22%, which could be due to the fact that it has a higher number of observations throughout the railway, allowing for a bigger dataset sample to train the model. The same conclusions apply to F1-score, since it's derived from the other two previously mentioned metrics. These metrics collectively show that the models are proficient in detecting these assets, albeit with varying degrees of success depending on the dataset size and asset characteristics. This is shown in Table 6.12.

**Table 6.12** Performance Metrics for Railway Assets

| **Asset** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|
| Concrete Twin-block Sleeper | 0.9835 | 0.9972 | 0.9903 |
| Transponder | 0.9787 | 0.9822 | 0.9805 |
| Railway Switch | 0.9643 | 0.9643 | 0.9643 |
| Wooden Sleepers | 0.9543 | 0.9253 | 0.9396 |
| Track Clearance Marker | 0.9524 | 0.9091 | 0.9302 |

It is essential to consider the context and limitations of these results. The relatively small datasets for some assets and the inherent variability in others influenced the models' performance.

Overall, the models' performance highlights their potential for automated railway asset detection using aerial imagery. The findings suggest that with continued improvement, these models could become invaluable tools for railway infrastructure management, offering accurate and efficient asset monitoring capabilities. Future work should focus on expanding the dataset, particularly for less frequently observed assets, and refining the models to reduce false positives while maintaining high precision and recall. Additionally, addressing the variability in asset appearance and improving the robustness of detection under various conditions will be crucial for enhancing overall performance.

# Chapter 7

# Development of a User Interface and Technology Transfer

## 7.1 Development of a User Interface for Enhanced Accessibility of YOLOv9 Models

This section presents the development of a web-based interface designed to simplify the interaction between end-users and the advanced functionalities of the YOLOv9 model [22]. This interface allows users to upload images or entire directories for object detection, select the desired detection model, and retrieve results efficiently. The primary goal of this interface is to make the advanced capabilities of YOLOv9 more accessible to users without requiring deep technical knowledge in machine learning or computer vision.

### 7.1.1 Application Framework and Interface Design

The application is built using Flask, a lightweight web framework ideal for handling web requests and serving HTML pages (Appendix J). Flask's ability to seamlessly integrate with Python scripts [23] is particularly advantageous for deploying machine learning models. For the front-end, React [35][36] was employed to create a dynamic and responsive user interface. React's component-based architecture and efficient state management allowed for a smooth and interactive user experience [37] (Appendix K). The interface design [38] prioritizes user-friendliness, ensuring that users can easily navigate through the process of file upload, model selection, and results retrieval (Appendix L).

### 7.1.2 Implementation Details

The implementation involves setting up a Flask server that handles HTTP requests. Users can upload images through a simple web form created with React, which supports uploading both individual files and directories. This flexibility is crucial for processing multiple images in batch mode, enhancing the usability of the application for practical scenarios.

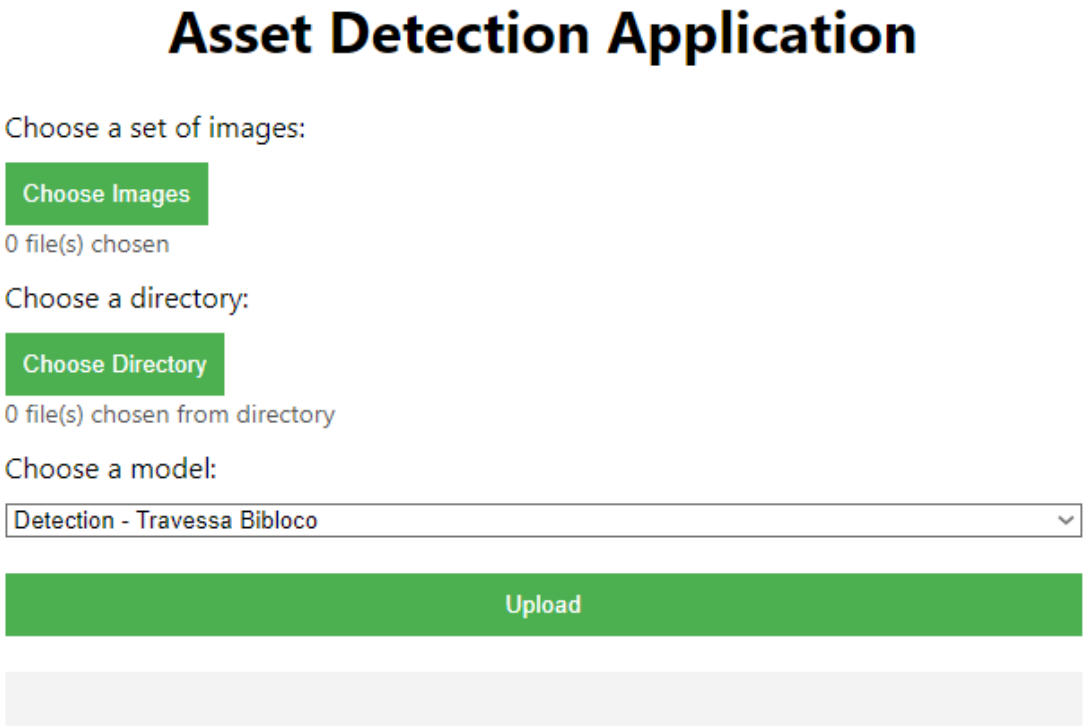



**Figure 7.1** Web application interface

The backend process includes:

- **File Upload and Handling**: Users can upload multiple image files or entire directories. The uploaded files are saved in a designated upload folder. Here is an example of how files are saved:

```
files = request.files.getlist('files')
for file in files:
    filename = secure_filename(file.filename)
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
```

- **Model Processing**: The uploaded images are processed using the YOLO model. The images are divided into smaller segments, and the model predicts bounding boxes for each segment. These predictions are then saved as text files. An example function to process images:

```
def process_image_and_save_txt(image_path, model):
    # Image processing logic here
    return output_filepath
```

- **Results**: The text files with predictions are combined into a ZIP file. If geospatial data is available, shape files are created and added to the ZIP file. The application provides the final ZIP file for download.

- **Clean-up**: A clean-up function ensures temporary files are deleted after processing.

The interface provides immediate feedback to the user about the status of the uploaded files and the ongoing processing, facilitating its usage.

### 7.1.3 Additional Features for Technology Transfer

To further facilitate technology transfer and ease of use, additional features were developed:

- **Batch File for Easy Launch**: A .bat file was created to allow the user to simply press an app icon and open the web application directly. This removes the need for users to run command-line scripts manually.
- **Dependency Installation Script**: Another .bat file was created to automatically install any dependencies or libraries needed for the app to work. This ensures that users can set up the application with minimal effort.

#### Batch File for Easy Launch

The following batch file allows users to open the web application directly by simply double-clicking an icon:

```
@echo off
cd path\to\your\webapp
start python app.py
```

This script changes the directory to the location of the web application and starts the Flask server by running the `app.py` script.

#### Dependency Installation Script

The following batch file installs all necessary dependencies for the web application:

```
@echo off
pip install -r requirements.txt
```

This script uses `pip` to install all the required Python libraries listed in the `requirements.txt` file, ensuring that the environment is correctly set up for the application to run.

### 7.1.4 Steps to Train a Model for a New Railway Asset

To replicate the process of training a YOLOv9 model for a new railway asset, a video tutorial was created showcasing the step-by-step guide. The steps outlined in the video are as follows:

1. **Data Collection**: A diverse set of images containing the new railway asset should be gathered. The dataset should include various lighting conditions, variations on the assets, and backgrounds to enhance the model's robustness.

2. **Annotation**: The images should be annotated with bounding boxes around the railway assets using an annotation tool (e.g., CVAT). The annotations should be saved in YOLO format.

3. **Data Preparation**: The annotated images and labels should be organized into training and validation sets. A balanced distribution should be ensured to avoid bias.

4. **Configuration File Setup**: A configuration file specifying the model architecture, hyperparameters, and paths to the training and validation datasets should be created.

5. **Training Script**: The `train.py` script should be used to initiate the training process. Parameters such as learning rate, batch size, and number of epochs should be adjusted as needed.

6. **Deployment**: Once the model meets the desired performance criteria, it should be deployed using the developed web interface. The model weights should be updated in the application to incorporate the new asset detection capability.

In addition to the video tutorial, a presentation was created for the team. These resources provide a comprehensive walkthrough, ensuring that all team members can follow along and successfully train and deploy models for new railway assets.

By utilizing these resources, users can efficiently train and deploy YOLOv9 models for new railway assets, leveraging the user-friendly web interface to simplify interactions and enhance accessibility.

By developing this user-friendly web interface, the gap between complex AI technologies and practical, everyday applications is bridged. Additionally, the creation of batch files for easy launch and dependency installation makes it much easier to access the web application without any need to know how to use a command line or an IDE to run the Python code.

# Chapter 8

# Conclusion

## 8.1 Restate the Research Problem and Objectives

Ensuring the safety and operational continuity of railway infrastructure demands precision, efficiency, and reliability in maintenance and management practices. Traditional asset management methods, characterized by manual processes and high error rates, often fall short due to their labor-intensive nature and significant costs. This thesis set out to address these challenges by integrating advanced artificial intelligence (AI) technologies, specifically deep learning for automated object detection, into the asset management process. The primary objective was to explore the application of the YOLOv9 algorithm for identifying and cataloguing railway assets from high-resolution drone-captured images, thereby enhancing the efficiency and accuracy of railway infrastructure management.

## 8.2 Summary of Key Findings

This research demonstrated that AI, and specifically the YOLOv9 object detection algorithm, significantly improves the precision and efficiency of railway asset management. By training the YOLOv9 model on a carefully annotated dataset of drone-captured images, the study achieved high-accuracy detections of various railway assets. The model's performance, assessed through metrics such as precision, recall, and mean average precision (mAP), revealed its robustness in handling different environmental conditions and asset appearances. The findings showed a marked reduction in the time required for asset inspections, thereby contributing to safer railway operations.

## 8.3 Implications of the Findings

The successful application of YOLOv9 for railway asset detection has several important implications. Firstly, it demonstrates the scalability and flexibility of AI technologies in infrastructure management, offering valuable insights into their potential for broader applications across various sectors. The integration of AI into railway asset management not only streamlines the

monitoring process, but also significantly reduces operational costs and enhances accuracy. This research underscores the transformative potential of AI in improving the management of critical infrastructure and paves the way for future innovations in this field.

## 8.4 Limitations of the Study

While the research achieved its objectives, certain limitations were encountered. The model's performance is inherently tied to the quality and diversity of the training dataset. Environmental factors such as lighting, weather conditions, and seasonal changes could impact the accuracy of asset detection. Additionally, while the web-based application developed in this project is user-friendly, there is a continuous need for updates and improvements based on user feedback and technological advancements. Finally, the study focused on a specific set of railway assets, expanding this, to include a wider variety of assets could further enhance the web application's utility.

## 8.5 Recommendations for Future Research

Future research should consider expanding the dataset to include more diverse environmental conditions and a broader range of railway assets. Exploring other advanced AI algorithms and combining them with YOLOv9 could further enhance detection accuracy and efficiency. Additionally, integrating AI with other technologies such as Internet of Things (IoT) devices for real-time monitoring and predictive maintenance could provide even greater benefits. Finally, investigating the application of this AI-based approach to other types of infrastructure, such as roads and bridges, could demonstrate its versatility and broader impact.

## 8.6 Practical Applications

The practical applications of this research are substantial. Railway infrastructure managers can leverage the developed AI model and web-based application to conduct faster and more accurate inspections, reducing downtime and enhancing safety. The scalability of this solution allows it to be adapted for use in various geographical locations and under different environmental conditions. Moreover, the user-friendly interface ensures that non-technical users can easily access and benefit from advanced AI capabilities, promoting widespread adoption and integration into existing maintenance practices.

## 8.7 Personal Reflection

Reflecting on this research journey, the transformative potential of AI in real-world applications has been profoundly rewarding. The challenges encountered during data acquisition, model training, and application development provided invaluable learning opportunities. This

project has not only contributed to the field of AI and railway infrastructure management but has also underscored the importance of interdisciplinary collaboration and continuous innovation. The experiences gained through overcoming technical hurdles and refining the AI model have highlighted the dynamic nature of this field and the ongoing need for adaptability and learning.

## 8.8 Concluding Statement

In conclusion, this thesis has demonstrated the significant benefits of integrating advanced AI technologies into railway asset management. The successful application of the YOLOv9 algorithm for automated object detection from drone-captured images marks a substantial advancement in infrastructure monitoring and maintenance. By enhancing precision, efficiency, and safety, this research underscores the transformative potential of AI in improving critical infrastructure management. The insights gained from this study lay the groundwork for future innovations and broader applications of AI, heralding a new era of smart infrastructure management. The findings advocate for a continued exploration of AI-driven solutions, encouraging further research and development to enhance the resilience and efficiency of vital infrastructure systems.

# Appendix A

# Python Script: images_into_640.py

This appendix includes the Python script `images_into_640.py` used for segmenting images.

```
from PIL import Image
import os
import math

input_directory = "C:/path/to/images/"
output_directory = "C:/path/to/output/"
os.makedirs(output_directory, exist_ok=True)
seg_width, seg_height = 640, 640
tif_files = [f for f in os.listdir(input_directory) if f.endswith('.tif')]

for tif_file in tif_files:
    image_path = os.path.join(input_directory, tif_file)
    large_image = Image.open(image_path)
    img_width, img_height = large_image.size
    num_width_segs = math.ceil(img_width / seg_width)
    num_height_segs = math.ceil(img_height / seg_height)
    image_segments = []

    for i in range(num_height_segs):
        for j in range(num_width_segs):
            left = j * seg_width if (j + 1) * seg_width <= img_width else\
            img_width - seg_width
            upper = i * seg_height if (i + 1) * seg_height <= img_height else\
            img_height - seg_height
            right = left + seg_width
            lower = upper + seg_height
            segment = large_image.crop((left, upper, right, lower))
            image_segments.append(segment)
```

```
    for i, segment in enumerate(image_segments):
        filename = f"{os.path.splitext(tif_file)[0]}_segment_{i+1}.tif"
        full_path = os.path.join(output_directory, filename)
        segment.save(full_path, "TIFF")

    print(f"Exported {len(image_segments)} segments for {tif_file} as TIF files.")
print("Processing complete for all images.")
```

# Appendix B

# Python Script: rotate_images.py

This appendix includes the Python script `rotate_images.py` used for rotating images and their corresponding bounding boxes.

```
import math
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image, ImageDraw
import numpy as np
import os

def rotate_point(cx, cy, angle, px, py):
    """Rotate a point around a given center."""
    s, c = math.sin(angle), math.cos(angle)
    px -= cx
    py -= cy
    xnew = px * c - py * s
    ynew = px * s + py * c
    return xnew + cx, ynew + cy

def get_rotated_image_size(w, h, angle):
    """Calculate the size of the image after rotation and expansion."""
    radians = math.radians(angle)
    sin_angle = abs(math.sin(radians))
    cos_angle = abs(math.cos(radians))
    new_w = int(h * sin_angle + w * cos_angle)
    new_h = int(h * cos_angle + w * sin_angle)
    return new_w, new_h

def rotate_bbox(bbox, angle, original_size):
    """Rotates the bounding box coordinates."""
```

```
    cx, cy, w, h = bbox  # cx, cy, w, h are in pixel coordinates
    angle_rad = math.radians(angle)
    points = np.array([
        [cx - w / 2, cy - h / 2],
        [cx + w / 2, cy - h / 2],
        [cx + w / 2, cy + h / 2],
        [cx - w / 2, cy + h / 2]
    ])
    new_points = np.array([rotate_point(original_size / 2, original_size / 2,\
    angle_rad, *point) for point in points])
    min_x, min_y = new_points.min(axis=0)
    max_x, max_y = new_points.max(axis=0)
    new_w = max_x - min_x
    new_h = max_y - min_y
    new_cx = min_x + new_w / 2
    new_cy = min_y + new_h / 2
    return new_cx, new_cy, new_w, new_h

def save_rotated_images_bboxes(image_path, bboxes, image_output_folder, bbox_output_folder):
    image = Image.open(image_path)
    original_image_name = os.path.splitext(os.path.basename(image_path))[0]
    original_w, original_h = image.size
    center_x, center_y = original_w / 2, original_h / 2
    for i, angle in enumerate(range(0, 360, 5)):
        new_img_w, new_img_h = get_rotated_image_size(original_w, original_h, angle)
        rotated_image = image.rotate(angle, expand=True)
        rotated_image_new = rotated_image.resize((640, 640))
        rotated_image_path = os.path.join(image_output_folder,\
        f"{original_image_name}_rotated_{angle}.tif")
        rotated_image_new.save(rotated_image_path, format='TIFF')

        if not bboxes:
            continue

        offset_x = (new_img_w - original_w) / 2
        offset_y = (new_img_h - original_h) / 2
        bbox_file_path = os.path.join(bbox_output_folder, \
        f"{original_image_name}_rotated_{angle}.txt")
        with open(bbox_file_path, 'w') as file:
            for bbox in bboxes:
```

```
                class_id, cx, cy, w, h = bbox
                cx, cy, w, h = cx * original_w, cy * original_h, w * \
                original_w, h * original_h
                new_cx, new_cy, new_w, new_h = rotate_bbox((cx, cy, w, h),\
                -angle, original_w)
                new_cx += offset_x
                new_cy += offset_y
                x1, y1, x2, y2 = new_cx - new_w / 2, new_cy - new_h / 2,\
                new_cx + new_w / 2, new_cy + new_h / 2
                w = x2 - x1
                h = y2 - y1
                cx = (x1 + x2) / 2
                cy = (y1 + y2) / 2
                file.write(f"{int(class_id)} {(cx)/new_img_w} {(cy)/new_img_h}\
                {(w)/new_img_w} {(h)/new_img_h}\n")

def read_bboxes_from_file(file_path):
    with open(file_path, 'r') as file:
        bboxes = [list(map(float, line.strip().split())) for line in file]
    return bboxes

base_dir = r"path/to/base/directory"
image_input_folder = os.path.join(base_dir, 'images/train_split/background')
bbox_input_folder = os.path.join(base_dir, 'labels/train_split/')
image_output_folder = os.path.join(base_dir, 'rotated/images/train/')
bbox_output_folder = os.path.join(base_dir, 'rotated/labels/train/')

os.makedirs(image_output_folder, exist_ok=True)
os.makedirs(bbox_output_folder, exist_ok=True)

for filename in os.listdir(image_input_folder):
    if filename.endswith('.tif'):
        image_path = os.path.join(image_input_folder, filename)
        bbox_file_path = os.path.join(bbox_input_folder, \
        os.path.splitext(filename)[0] + '.txt')
        bboxes = []
        if os.path.exists(bbox_file_path):
            bboxes = read_bboxes_from_file(bbox_file_path)

        save_rotated_images_bboxes(image_path, bboxes, image_output_folder,\
```

```
        bbox_output_folder)
    else:
        print(f"Skipping {filename}, not a .tif file.")
```

# Appendix C

# Python Script: train_val_split.py

This appendix includes the Python script `train_val_split.py` used for splitting the dataset into training and validation sets.

```
import os
import shutil
from sklearn.model_selection import train_test_split

def move_files(files, src_dir, dst_dir):
    for file in files:
        src_path = os.path.join(src_dir, file)
        if not os.path.exists(src_path):
            continue
        dst_path = os.path.join(dst_dir, file)
        shutil.move(src_path, dst_path)

# Define directory paths
image_dir = 'path/to/images/train'
bbox_dir = 'path/to/labels/train'

image_train_dir = 'path/to/images/train'
image_val_dir = 'path/to/images/val'
bbox_train_dir = 'path/to/labels/train'
bbox_val_dir = 'path/to/labels/val'

# Get all image filenames without extensions
all_images = [os.path.splitext(f)[0] for f in os.listdir(image_dir) if f.endswith('.tif')]

# Split the data into training and validation sets
train_images, val_images = train_test_split(all_images, test_size=0.3, random_state=42)
```

```
# Move files to the corresponding directories
move_files([f + '.tif' for f in train_images], image_dir, image_train_dir)
move_files([f + '.txt' for f in train_images], bbox_dir, bbox_train_dir)
move_files([f + '.tif' for f in val_images], image_dir, image_val_dir)
move_files([f + '.txt' for f in val_images], bbox_dir, bbox_val_dir)

print(f"Moved {len(train_images)} files to the training set.")
print(f"Moved {len(val_images)} files to the validation set.")
```

# Appendix D

# Python Script: apply_random_noise.py

This appendix includes the Python script `apply_random_noise.py` used for applying random noise to images outside of bounding boxes.

```
from PIL import Image, ImageDraw
import numpy as np
import os
import shutil

def load_bboxes(txt_file_path):
    with open(txt_file_path, 'r') as file:
        bboxes = [list(map(float, line.strip().split())) for line in file.readlines()]
    return [bbox[1:] for bbox in bboxes]

def create_bbox_mask(image_size, bboxes):
    mask = np.zeros((image_size[1], image_size[0]), dtype=bool)
    for bbox in bboxes:
        cx, cy, w, h = bbox
        x1 = int((cx - w / 2) * image_size[0])
        y1 = int((cy - h / 2) * image_size[1])
        x2 = int((cx + w / 2) * image_size[0])
        y2 = int((cy + h / 2) * image_size[1])
        mask[y1:y2, x1:x2] = True
    return mask

def apply_random_noise(image, mask):
    noise = np.random.normal(128, 37, (image.height, image.width, 3)).astype(np.uint8)
    noise = np.clip(noise, 0, 255)
    if image.mode == 'RGBA':
```

```
        image = image.convert('RGB')
    image_array = np.array(image)
    inverted_mask = ~mask
    for c in range(3):
        image_array[:, :, c][inverted_mask] = noise[:, :, c][inverted_mask]
    return Image.fromarray(image_array)

def process_images_and_bboxes(image_dir, bbox_dir):
    for filename in os.listdir(image_dir):
        if filename.endswith('.tif'):
            base_name = filename[:-4]
            image_path = os.path.join(image_dir, filename)
            bbox_file_path = os.path.join(bbox_dir, base_name + '.txt')

            if os.path.exists(bbox_file_path):
                bboxes = load_bboxes(bbox_file_path)
                image = Image.open(image_path)
                mask = create_bbox_mask(image.size, bboxes)
                image_with_noise = apply_random_noise(image, mask)

                output_image_path = os.path.join(image_dir, base_name + '_random_noise.tif')
                output_bbox_path = os.path.join(bbox_dir, base_name + '_random_noise.txt')

                image_with_noise.save(output_image_path)
                shutil.copyfile(bbox_file_path, output_bbox_path)
                print(f"Processed and saved: {output_image_path} and {output_bbox_path}")

# Define directory paths
image_dir = 'path/to/images/train'
bbox_dir = 'path/to/labels/train'

process_images_and_bboxes(image_dir, bbox_dir)
```

# Appendix E

# Python Script: apply_gravel_texture.py

This appendix includes the Python script `apply_gravel_texture.py` used for applying gravel textures to the background areas outside of bounding boxes in images.

```
from PIL import Image
import numpy as np
import os
import random
import shutil

def is_image_processed(image_path, suffix='_processed'):
    """Check if the image has already been processed."""
    base, ext = os.path.splitext(image_path)
    processed_path = f"{base}{suffix}{ext}"
    return os.path.exists(processed_path)

def load_bboxes(txt_file_path):
    """Load bounding boxes from a text file."""
    with open(txt_file_path, 'r') as file:
        bboxes = [list(map(float, line.strip().split())) for line in file.readlines()]
    return [bbox[1:] for bbox in bboxes]

def create_inverted_bbox_mask(image_size, bboxes):
    """Create a mask that is True outside bounding boxes."""
    mask = np.ones((image_size[1], image_size[0]), dtype=bool)
    for bbox in bboxes:
        cx, cy, w, h = bbox
        x1 = int((cx - w / 2) * image_size[0])
        y1 = int((cy - h / 2) * image_size[1])
```

```
            x2 = int((cx + w / 2) * image_size[0])
            y2 = int((cy + h / 2) * image_size[1])
            mask[y1:y2, x1:x2] = False
    return mask

def sample_colors_from_texture(texture, num_samples=100):
    """Sample colors from a texture."""
    texture_array = np.array(texture)
    sample_indices = [(random.randint(0, texture.height - 1), \
    random.randint(0, texture.width - 1)) for _ in range(num_samples)]
    return [tuple(texture_array[idx]) for idx in sample_indices]

def apply_color_from_samples(image, mask, gravel_texture_dir):
    """Apply sampled colors from a random texture to the masked areas of the image."""
    gravel_textures = [os.path.join(gravel_texture_dir, f) for f in \
    os.listdir(gravel_texture_dir) if f.endswith('.jpg')]
    if not gravel_textures:
        raise FileNotFoundError("No texture files found in the specified directory.")
    gravel_texture_path = random.choice(gravel_textures)
    gravel_texture = Image.open(gravel_texture_path).convert('RGB')

    color_samples = sample_colors_from_texture(gravel_texture, 1000)

    image_array = np.array(image)
    mask_indices = np.argwhere(mask)
    for idx in mask_indices:
        random_color = random.choice(color_samples)
        if image_array.shape[2] == 4:  # Check if image has an alpha channel
            image_array[idx[0], idx[1], :3] = random_color
        else:
            image_array[idx[0], idx[1], :] = random_color

    return Image.fromarray(image_array)

def process_single_image(image_path, bbox_file_path, gravel_texture_dir, base_name):
    if not is_image_processed(image_path):
        bboxes = load_bboxes(bbox_file_path)
        output_bbox_path = os.path.join(os.path.dirname(bbox_file_path), \
        base_name + '_processed.txt')
        image = Image.open(image_path).convert('RGB')
```

```
        mask = create_inverted_bbox_mask(image.size, bboxes)
        image_with_texture = apply_color_from_samples(image, mask, gravel_texture_dir)
        output_image_path = image_path.replace('.tif', '_processed.tif')
        image_with_texture.save(output_image_path)
        shutil.copyfile(bbox_file_path, output_bbox_path)
        print(f"Processed and saved: {output_image_path} and {output_bbox_path}")
    else:
        print(f"Skipping already processed file: {image_path}")

# Define directory paths
image_dir = 'path/to/images/train'
bbox_dir = 'path/to/labels/train'
gravel_texture_dir = 'path/to/gravel/textures'

for filename in os.listdir(image_dir):
    if filename.endswith('.tif'):
        image_path = os.path.join(image_dir, filename)
        base_name = os.path.splitext(filename)[0]
        bbox_file_path = os.path.join(bbox_dir, base_name + '.txt')
        if os.path.exists(bbox_file_path):
            process_single_image(image_path, bbox_file_path, gravel_texture_dir, base_name)
        else:
            print(f"No bounding box file found for {filename}, skipping.")
```

# Appendix F

# Python Script: train.py

This appendix includes the Python script `train.py` used for training the YOLO model on the specified dataset.

```
import torch
from IPython.display import Image
from ultralytics import YOLO
import os

# Function to run training
def run_training():
    print(torch.cuda.device_count())
    print(torch.cuda.get_device_name(0))

    # Load a model
    model = YOLO("yolov9c.yaml")  # build a new model from scratch

    # Define dataset and save directories
    dataset = "path/to/dataset.yaml"
    save_dir = 'path/to/save/directory'
    save_name = 'my_training_run'

    # Use the model
    results = model.train(data=dataset, batch=16, epochs=200, patience=300, \
    project=save_dir, name=save_name)  # train the model

# Main guard
if __name__ == '__main__':
    run_training()
```

# Appendix G

# Python Script: test_export_bbox.py

This appendix includes the Python script `test_export_bbox.py` used for processing images, predicting bounding boxes using the YOLO model, and exporting the bounding boxes to text files.

```
from ultralytics import YOLO
import cv2
from PIL import Image
import numpy as np
from pathlib import Path
import os

# Define directory paths and model path
input_directory = Path('path/to/input_directory')
output_directory = Path('path/to/output_directory')
model_path = 'path/to/model/weights/last.pt'

# Load the YOLO model
model = YOLO(model_path)

# Create output directory if it doesn't exist
if not output_directory.exists():
    output_directory.mkdir(parents=True)

# Define image and segment dimensions
img_width, img_height = 5120, 5120
seg_width, seg_height = 640, 640

# Process each image file in the input directory
for image_file in input_directory.glob('*.tif'):
    large_image = Image.open(image_file)
```

```
original_width, original_height = large_image.size
resized_image = large_image.resize((img_width, img_height))
scale_x = original_width / img_width
scale_y = original_height / img_height
all_bboxes = []

# Segment the image and predict bounding boxes
for i in range(0, img_height, seg_height):
    for j in range(0, img_width, seg_width):
        segment = resized_image.crop((j, i, j + seg_width, i + seg_height))
        segment_cv = cv2.cvtColor(np.array(segment), cv2.COLOR_RGB2BGR)
        pred = model(segment_cv, imgsz=seg_width)
        boxes = pred[0].boxes.xyxy.tolist()
        classes = pred[0].boxes.cls.tolist()
        conf = pred[0].boxes.conf.tolist()
        names = pred[0].names

        for (x1, y1, x2, y2), cls, conf_level in zip(boxes, classes, conf):
            if conf_level >= 0.0:
                all_bboxes.append((names[cls], x1 * scale_x + j * scale_x, y1 *\
                scale_y + i * scale_y, x2 * scale_x + j * scale_x, y2 *\
                scale_y + i * scale_y, conf_level))

# Save the bounding boxes to a text file
output_filepath = output_directory / (image_file.stem + '.txt')
with open(output_filepath, 'w') as f:
    for cls, x1, y1, x2, y2, conf_level in all_bboxes:
        f.write(f"{cls} {x1} {y1} {x2} {y2} {conf_level}\n")

print(f"Processed and saved predictions for {image_file.name} in {output_filepath}")
```

# Appendix H

# Python Script: plot_bboxes.py

This appendix includes the Python script `plot_bboxes.py` used for plotting bounding boxes on images to verify the accuracy of the model's predictions.

```
from PIL import Image
import numpy as np
import cv2
from pathlib import Path
import matplotlib.pyplot as plt
import os

# Define directory paths
image_directory = Path('path/to/image_directory')
bbox_directory = Path('path/to/bbox_directory')

for filepath in image_directory.glob('*.tif'):
    bbox_file_path = bbox_directory / (filepath.stem + '.txt')
    try:
        pil_image = Image.open(filepath)
        image = np.array(pil_image)
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    except IOError:
        print(f"Failed to load image: {filepath}")
        continue

    if bbox_file_path.exists():
        with open(bbox_file_path, 'r') as f:
            for line in f:
                cls, x1, y1, x2, y2, conf = line.strip().split()
                if float(conf) < 0.00:
                    continue
```

```
            x1 = int(float(x1))
            y1 = int(float(y1))
            x2 = int(float(x2))
            y2 = int(float(y2))
            cv2.rectangle(image, (x1, y1), (x2, y2), (255, 0, 0), 2)
            cv2.putText(image, f"{cls} {float(conf):.2f}", (x1, y1 - 10), \
            cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(12, 8))
plt.imshow(image)
plt.title(filepath.name)
plt.axis('off')
plt.show()
```

# Appendix I

# Python Script: bbox_to_arcgis.py

This appendix includes the Python script `bbox_to_arcgis.py` used for converting bounding box text files into shapefiles compatible with ArcGIS Pro.

```
# import arcpy
import os

# Define directory paths
image_directory = r"path/to/image_directory"
bbox_directory = r"path/to/bbox_directory"

def get_transformation_params(tfw_path):
    with open(tfw_path, 'r') as f:
        params = [float(line.strip()) for line in f.readlines()]
    return params

def bbox_to_polygon(cx, cy, w, h, params):
    a, d, b, e, c, f = params
    half_width = w / 2.0
    half_height = h / 2.0
    geo_cx = a * cx + b * cy + c
    geo_cy = d * cx + e * cy + f
    geo_half_width = a * half_width
    geo_half_height = e * half_height
    return [
        arcpy.Point(geo_cx - geo_half_width, geo_cy + geo_half_height),
        arcpy.Point(geo_cx + geo_half_width, geo_cy + geo_half_height),
        arcpy.Point(geo_cx + geo_half_width, geo_cy - geo_half_height),
        arcpy.Point(geo_cx - geo_half_width, geo_cy - geo_half_height),
    ]
```

```
for image_filename in os.listdir(image_directory):
    if image_filename.endswith(".tif"):
        base_name = os.path.splitext(image_filename)[0]
        output_shapefile = os.path.join(image_directory, f"{base_name}_BoundingBoxes.shp")
        bbox_file_path = os.path.join(bbox_directory, base_name + ".txt")
        tfw_file_path = os.path.join(image_directory, base_name + ".tfw")
        prj_file_path = os.path.join(image_directory, base_name + ".prj")

        if os.path.exists(bbox_file_path) and os.path.exists(tfw_file_path)\
        and os.path.exists(prj_file_path):
            spatial_ref = arcpy.SpatialReference(prj_file_path)
            params = get_transformation_params(tfw_file_path)

            arcpy.CreateFeatureclass_management(
                out_path=os.path.dirname(output_shapefile),
                out_name=os.path.basename(output_shapefile),
                geometry_type="POLYGON",
                spatial_reference=spatial_ref
            )

            arcpy.AddField_management(output_shapefile, "Label", "TEXT")

            with open(bbox_file_path, 'r') as bbox_file:
                with arcpy.da.InsertCursor(output_shapefile, ['SHAPE@', 'Label']) as cursor:
                    for line in bbox_file:
                        parts = line.strip().split()
                        if len(parts) == 5:
                            label, cx, cy, w, h = parts
                            cx, cy, w, h = map(float, [cx, cy, w, h])
                            polygon = arcpy.Polygon(arcpy.Array(bbox_to_polygon(cx, cy,\
                            w, h, params)), spatial_ref)
                            cursor.insertRow([polygon, label])
                        else:
                            print(f"Invalid bbox format in file {bbox_file_path}")

print("Finished creating individual bounding box shapefiles for each image.")
```

# Appendix J

# Python Script: app.py

This appendix includes the Python script `app.py` used for creating a web application that performs inference on uploaded images.

```
import sys
import os
import tempfile
import shutil
from zipfile import ZipFile
from flask import Flask, request, send_file, jsonify
from flask_cors import CORS
from werkzeug.utils import secure_filename
from ultralytics import YOLO
from PIL import Image
import cv2
import numpy as np
import logging
import time
from waitress import serve
import geopandas as gpd
from shapely.geometry import Polygon
import threading
import re

# Ensure no prompts
def ensure_no_prompts():
    os.environ['DEBIAN_FRONTEND'] = 'noninteractive'
    os.environ['INTERACTIVE'] = 'false'
    os.environ['PYTHONUNBUFFERED'] = '1'  # Ensure stdout and stderr are not buffered

ensure_no_prompts()
```

```
# logging errors to debug
# logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(levelname)s %(message)s')

base_dir = os.path.dirname(os.path.realpath(__file__))
app = Flask(__name__, static_folder='build', static_url_path='')
CORS(app)
app.config['UPLOAD_FOLDER'] = 'uploads/'
app.config['MAX_CONTENT_LENGTH'] = 20000 * 1024 * 1024
app.secret_key = 'hello_world'
OUTPUT_FOLDER = os.path.join(app.config['UPLOAD_FOLDER'], 'output_files')
os.makedirs(OUTPUT_FOLDER, exist_ok=True)

def secure_filename_special_characters(filename):
    return re.sub(r'[^a-zA-Z0-9_.\-+]', '_', filename)

def process_image_and_save_txt(image_path, model, name_file):
    os.makedirs(OUTPUT_FOLDER, exist_ok=True)
    large_image = Image.open(image_path)
    img_width, img_height = 5120, 5120
    seg_width, seg_height = 640, 640
    resized_image = large_image.resize((img_width, img_height))
    scale_x = large_image.width / img_width
    scale_y = large_image.height / img_height
    all_bboxes = []

    for i in range(0, img_height, seg_height):
        for j in range(0, img_width, seg_width):
            segment = resized_image.crop((j, i, j + seg_width, i + seg_height))
            segment_cv = cv2.cvtColor(np.array(segment), cv2.COLOR_RGB2BGR)
            pred = model(segment_cv, imgsz=seg_width)
            boxes = pred[0].boxes.xyxy.tolist()
            classes = pred[0].boxes.cls.tolist()
            conf = pred[0].boxes.conf.tolist()
            names = pred[0].names
            for (x1, y1, x2, y2), cls, conf_level in zip(boxes, classes, conf):
                if conf_level >= 0.0:
                    all_bboxes.append((names[cls], x1 * scale_x + j * scale_x, y1 *\
                    scale_y + i * scale_y,x2 * scale_x + j * scale_x, y2 * scale_y +\
                    i * scale_y, conf_level))
```

```
    output_filename = secure_filename_special_characters(os.path.basename(name_file).replace\
    ('.tif', '.txt').replace('.jpg', '.txt'))
    output_filepath = os.path.join(OUTPUT_FOLDER, output_filename)

    with open(output_filepath, 'w') as f:
        for cls, x1, y1, x2, y2, conf_level in all_bboxes:
            f.write(f"{cls} {x1} {y1} {x2} {y2} {conf_level}\n")
        f.flush()

    return output_filepath

def get_transformation_params(georef_path):
    with open(georef_path, 'r') as f:
        params = [float(line.strip()) for line in f.readlines()]
    return params

def bbox_to_polygon(cx, cy, w, h, params):
    a, d, b, e, c, f = params
    half_width = w / 2.0
    half_height = h / 2.0
    geo_cx = a * cx + b * cy + c
    geo_cy = d * cx + e * cy + f
    geo_half_width = a * half_width
    geo_half_height = e * half_height
    return Polygon([
        (geo_cx - geo_half_width, geo_cy + geo_half_height),
        (geo_cx + geo_half_width, geo_cy + geo_half_height),
        (geo_cx + geo_half_width, geo_cy - geo_half_height),
        (geo_cx - geo_half_width, geo_cy - geo_half_height)
    ])

def create_shapefiles_from_bboxes(image_directory, bbox_directory):
    shapefile_paths = []
    for image_filename in os.listdir(image_directory):
        if image_filename.endswith((".tif", ".jpg")):
            base_name = os.path.splitext(image_filename)[0]
            bbox_file_path = os.path.join(bbox_directory, base_name + ".txt")
            georef_file_path = os.path.join(image_directory, base_name + (".tfw" \
            if image_filename.endswith(".tif") else ".jgw"))
            prj_file_path = os.path.join(image_directory, base_name + (".prj" \
```

```
            if image_filename.endswith(".tif") else ".jpg.xml"))

            if os.path.exists(bbox_file_path) and os.path.exists(georef_file_path) and\
            os.path.exists(prj_file_path):
                params = get_transformation_params(georef_file_path)
                bbox_data = []
                with open(bbox_file_path, 'r') as bbox_file:
                    for line in bbox_file:
                        parts = line.strip().split()
                        if len(parts) == 6:
                            label, x1, y1, x2, y2, conf = parts
                            x1, y1, x2, y2 = map(float, [x1, y1, x2, y2])
                            w, h = x2 - x1, y2 - y1
                            cx, cy = x1 + w / 2, y1 + h / 2
                            polygon = bbox_to_polygon(cx, cy, w, h, params)
                            bbox_data.append({'geometry': polygon, 'Label': label, \
                            'Confidence': conf})
                        else:
                            logging.debug(f"Invalid bbox format in file {bbox_file_path}")
                if bbox_data:
                    output_shapefile = os.path.join(image_directory,\
                    f"{base_name}_BoundingBoxes_{label}.shp")
                    # Set the CRS to ETRS_1989_Portugal_TM06 (EPSG:3763)
                    gdf = gpd.GeoDataFrame(bbox_data, crs="EPSG:3763")
                    gdf.set_geometry('geometry', inplace=True)
                    gdf.to_file(output_shapefile)
                    shapefile_paths.append(output_shapefile)
                else:
                    logging.debug(f"No valid bbox data found in file {bbox_file_path}")
    return shapefile_paths


@app.route('/api/upload', methods=['POST'])
def upload_files():
    model_selection = request.form['model']
    model_path = os.path.join(base_dir, model_selection)
    model = YOLO(model_path)
    zip_dir = tempfile.mkdtemp()
    zip_path = os.path.join(zip_dir, 'output_files.zip')
    files = request.files.getlist('files')
```

```
    if not files:
        logging.debug('No files uploaded.')
        return jsonify({'error': 'No files or directories were uploaded.'}), 400

    all_files = []
    for file in files:
        if file:
            filename = secure_filename_special_characters(os.path.basename(file.filename))
            base_filename = os.path.basename(filename)
            save_path = os.path.join(app.config['UPLOAD_FOLDER'], base_filename)
            os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
            file.save(save_path)
            logging.debug(f'File saved: {save_path}')
            sys.stdout.flush()  # Explicitly flush the stdout buffer
            all_files.append(save_path)

    with ZipFile(zip_path, 'w') as zipf:
        for file_path in all_files:
            if file_path.endswith(('.tif', '.jpg')):
                filename = secure_filename_special_characters(os.path.basename(file_path))
                txt_file_path = process_image_and_save_txt(file_path, model, filename)
                zipf.write(txt_file_path, arcname=os.path.basename(txt_file_path))
                logging.debug(f"TXT file added to zip: {txt_file_path}")
                sys.stdout.flush()  # Explicitly flush the stdout buffer

    logging.debug(f'Zip file created at: {zip_path}')
    sys.stdout.flush()  # Explicitly flush the stdout buffer
    return jsonify({'zip_path': zip_path, 'all_files': all_files})

@app.route('/api/combine_zips', methods=['POST'])
def combine_zips():
    zip_paths = request.json.get('zip_paths', [])
    all_files = request.json.get('all_files', [])
    combined_zip_dir = tempfile.mkdtemp()
    combined_zip_path = os.path.join(combined_zip_dir, 'combined_output_files.zip')

    create_shapefiles = any(file.endswith(('.tfw', '.jgw')) for file in all_files) \
    and any(file.endswith(('.prj', '.xml')) for file in all_files)
```

```
    try:
        with ZipFile(combined_zip_path, 'w') as combined_zip:
            for zip_path in zip_paths:
                with ZipFile(zip_path, 'r') as zipf:
                    for file_info in zipf.infolist():
                        if create_shapefiles and file_info.filename.endswith('.txt'):
                            logging.debug(f"Skipping .txt file: {file_info.filename}")
                            continue
                        with zipf.open(file_info.filename) as source_file:
                            combined_zip.writestr(file_info, source_file.read())

            if create_shapefiles:
                shapefile_paths = create_shapefiles_from_bboxes(app.config['UPLOAD_FOLDER'],
                for shapefile in shapefile_paths:
                    for ext in ['.shp', '.shx', '.dbf', '.prj', '.xml']:
                        shapefile_path = shapefile.replace('.shp', ext)
                        if os.path.exists(shapefile_path):
                            combined_zip.write(shapefile_path, arcname=os.path.basename\
                            (shapefile_path))logging.debug(f"Shapefile added\
                            to zip: {shapefile_path}")

        file_handle = open(combined_zip_path, 'rb')
        response = send_file(file_handle, as_attachment=True,\
        download_name='combined_output_files.zip')
        response.headers["Content-Disposition"] = "attachment;\
        filename=combined_output_files.zip"
        cleanup_thread = threading.Thread(target=cleanup, args=(zip_paths,\
        combined_zip_dir, file_handle))
        cleanup_thread.start()
        return response
    except Exception as e:
        logging.error(f"Error during processing: {e}")
        return jsonify({'error': str(e)}), 500

def cleanup(zip_paths, combined_zip_dir, file_handle):
    logging.debug('Cleanup started.')
    sys.stdout.flush()  # Explicitly flush the stdout buffer
    try:
        time.sleep(5)
        file_handle.close()
```

```
    for zip_path in zip_paths:
        dir_path = os.path.dirname(zip_path)
        if os.path.exists(dir_path):
            shutil.rmtree(dir_path)
            logging.debug(f'Removed directory: {dir_path}')
            sys.stdout.flush()  # Explicitly flush the stdout buffer
    if os.path.exists(combined_zip_dir):
        shutil.rmtree(combined_zip_dir)
        logging.debug(f'Removed combined zip directory: {combined_zip_dir}')
        sys.stdout.flush()  # Explicitly flush the stdout buffer
    if os.path.exists(app.config["UPLOAD_FOLDER"]):
        shutil.rmtree(app.config["UPLOAD_FOLDER"])
        logging.debug(f'Removed upload folder: {app.config["UPLOAD_FOLDER"]}')
        sys.stdout.flush()  # Explicitly flush the stdout buffer
except PermissionError as e:
    logging.error(f"PermissionError during cleanup: {e}")
    sys.stdout.flush()  # Explicitly flush the stdout buffer
    time.sleep(5)
    try:
        for zip_path in zip_paths:
            dir_path = os.path.dirname(zip_path)
            if os.path.exists(dir_path):
                shutil.rmtree(dir_path)
                logging.debug(f'Removed directory on second attempt: {dir_path}')
                sys.stdout.flush()  # Explicitly flush the stdout buffer
        if os.path.exists(combined_zip_dir):
            shutil.rmtree(combined_zip_dir)
            logging.debug(f'Removed combined zip directory on second attempt:\
            {combined_zip_dir}')
            sys.stdout.flush()  # Explicitly flush the stdout buffer
        if os.path.exists(app.config["UPLOAD_FOLDER"]):
            shutil.rmtree(app.config["UPLOAD_FOLDER"])
            logging.debug(f'Removed upload folder on second attempt:\
            {app.config["UPLOAD_FOLDER"]}')
            sys.stdout.flush()  # Explicitly flush the stdout buffer
    except Exception as e:
        logging.error(f"Failed to clean up on second attempt: {str(e)}", exc_info=True)
        sys.stdout.flush()  # Explicitly flush the stdout buffer
except Exception as e:
    logging.error(f"Error during cleanup: {e}")
```

```
        sys.stdout.flush()  # Explicitly flush the stdout buffer

@app.route('/', methods=['GET'])
def serve_react_app():
    return app.send_static_file('index.html')

if __name__ == '__main__':
    print('\nRunning on http://localhost:8000')
    serve(app, host='0.0.0.0', port=8000, connection_limit=3000, channel_timeout=5000)
```

# Appendix K

# React Component: FileUpload.jsx

This appendix includes the React component `FileUpload.jsx` used for the frontend of the web application, allowing users to upload image files or directories and select a model for processing.

```
import React, { useState } from 'react';
import axios from 'axios';
import './FileUpload.css';

const FileUpload = () => {
  const [files, setFiles] = useState([]);
  const [directoryFiles, setDirectoryFiles] = useState([]);
  const [model, setModel] = useState('Bibloco.pt');
  const [uploadProgress, setUploadProgress] = useState(0);
  const [errorMessage, setErrorMessage] = useState('');
  const [successMessage, setSuccessMessage] = useState('');

  const handleFileChange = (event) => {
    setFiles([...event.target.files]);
  };

  const handleDirectoryChange = (event) => {
    setDirectoryFiles([...event.target.files]);
  };

  const handleModelChange = (event) => {
    setModel(event.target.value);
  };

  const uploadFiles = async () => {
    const totalFiles = [...files, ...directoryFiles];
```

```
const chunkSize = 20; // Adjust the chunk size as needed
const totalChunks = Math.ceil(totalFiles.length / chunkSize);

let zipPaths = [];
let allFiles = [];

setUploadProgress(0);
setErrorMessage('');
setSuccessMessage('');

for (let chunkIndex = 0; chunkIndex < totalChunks; chunkIndex++) {
  const chunk = totalFiles.slice(chunkIndex * chunkSize, (chunkIndex + 1) * chunkSize);
  const formData = new FormData();

  chunk.forEach(file => {
    formData.append('files', file);
  });
  formData.append('model', model);

  try {
    console.log(`Uploading chunk ${chunkIndex + 1} of ${totalChunks}`);
    const response = await axios.post(`http://localhost:8000/api/upload`, formData, {
      onUploadProgress: (progressEvent) => {
        const progress = ((chunkIndex + 1) / totalChunks) * 100;
        setUploadProgress(progress);
      },
    });

    console.log('Response:', response.data);
    zipPaths.push(response.data.zip_path);
    allFiles = allFiles.concat(response.data.all_files);
  } catch (error) {
    console.error('Upload failed', error);
    setErrorMessage('Upload failed. Please try again.');
    return;
  }
}

// Combine all the zip parts into one file
try {
```

```
    console.log('Combining zips');
    const combineResponse = await axios.post(`http://localhost:8000/api/combine_zips`,\
    { zip_paths: zipPaths, all_files: allFiles }, {
      responseType: 'blob',
    });

    const url = window.URL.createObjectURL(combineResponse.data);

    const a = document.createElement('a');
    a.href = url;
    a.download = 'combined_output_files.zip';
    document.body.appendChild(a);
    a.click();
    window.URL.revokeObjectURL(url);
    document.body.removeChild(a);

    setSuccessMessage('Upload completed! Downloading the ZIP file.');
  } catch (error) {
    console.error('Combining zips failed', error);
    setErrorMessage('Combining zips failed. Please try again.');
  }
};

return (
  <div className="file-upload-container">
    <h1>Aplicação para deteção de ativos</h1>
    <form>
      <div className="file-input-container">
        <label htmlFor="files">Escolha um conjunto de imagens:</label>
        <input type="file" id="files" name="files" multiple onChange={handleFileChange} />
        <button type="button" onClick={() => document.getElementById('files').click()}>\
        Escolher Imagens</button>
        <span id="file-chosen">{files.length} file(s) chosen</span>
      </div>

      <div className="file-input-container">
        <label htmlFor="directory">Escolha uma diretoria:</label>
        <input type="file" id="directory" name="directory" webkitdirectory="true"\
        multiple onChange={handleDirectoryChange} />
        <button type="button" onClick={() => document.getElementById('directory').click()}>
```

```
          Escolher Diretoria</button>
          <span id="directory-chosen">{directoryFiles.length} file(s) chosen from directory\
          </span>
        </div>

        <label htmlFor="model">Escolha um modelo:</label>
        <select name="model" id="model" value={model} onChange={handleModelChange}>
          <option value="Bibloco.pt">Detecção - Travessa Bibloco</option>
          <option value="Baliza.pt">Detecção - Baliza de Convel</option>
          <option value="Travessa_madeira.pt">Detecção - Travessa de Madeira</option>
          <option value="Travessa_betao.pt">Detecção - Travessa de Betão</option>
          <option value="AMV.pt">Detecção - AMV</option>
          <option value="lr.pt">Detecção - Lr</option>
        </select>

        <button type="button" onClick={uploadFiles}>Upload</button>
      </form>

      <div className="progress">
        <div className="progress-bar" style={{ width: `${uploadProgress}%` }}>
          {uploadProgress > 0 && `${Math.round(uploadProgress)}%`}
        </div>
      </div>

      {errorMessage && <div className="error-message">{errorMessage}</div>}
      {successMessage && <div className="success-message">{successMessage}</div>}
    </div>
  );
};

export default FileUpload;
```

# Appendix L

# React component CSS Stylesheet: FileUpload.css

This appendix includes the CSS stylesheet `FileUpload.css` used for styling the frontend of the web application.

```
.file-upload-container {
  width: 100%;
  max-width: 600px;
  margin: 0 auto;
  padding: 20px;
  text-align: center;
}

form {
  display: flex;
  flex-direction: column;
  gap: 10px;
}

label {
  text-align: left;
}

input[type="file"] {
  display: none;
}

span {
  font-size: 14px;
  color: #555;
```

```
}

button {
  padding: 10px;
  background-color: #4CAF50;
  color: white;
  border: none;
  cursor: pointer;
  margin-top: 10px;
}

button:hover {
  background-color: #45a049;
}

.file-input-container {
  display: flex;
  flex-direction: column;
  align-items: flex-start;
}

.progress {
  width: 100%;
  background-color: #f3f3f3;
  margin-top: 20px;
}

.progress-bar {
  width: 0%;
  height: 30px;
  background-color: #4CAF50;
  text-align: center;
  line-height: 30px;
  color: white;
}

.error-message {
  color: red;
  margin-top: 20px;
}
```

```
.success-message {
  color: green;
  margin-top: 20px;
}
```

# Appendix M

# React Component: App.js

This appendix includes the React component `App.js` used for testing the application, ensuring it renders correctly.

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

# Bibliography


[1] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2016.

[2] Aurelien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*. O'Reilly, 2019.

[3] François Chollet. *Deep Learning with Python*. Manning Pub. Comp., 2017.

[4] Mohamed Elgendy. *Deep Learning for Vision Systems*. Manning Pub. Comp., 2020.

[5] Aston Zhang, Zachary Lipton, Mu Li, and Alexander Smola. *Dive into Deep Learning*. OnLine, 2022.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[7] Qiong Liu and Ying Wu. Supervised learning. 01 2012. doi:10.1007/978-1-4419-1428-6_451.

[8] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. pages 91–99, 2015.

[9] Enzo Grossi and Massimo Buscema. Introduction to artificial neural networks. *European journal of gastroenterology hepatology*, 19:1046–54, 01 2008. doi:10.1097/MEG.0b013e3282f198a0.

[10] Samandarov Erkaboy Karimboyevich and Abdurakhmonov Olim Nematullayevich. Single layer artificial neural network: Perceptron. *European Multidisciplinary Journal of Modern Science*, 5:230–238, Apr. 2022. URL `https://emjms.academicjournal.io/index.php/emjms/article/view/253`.

[11] Abdulhafis Abdulazeez Osuwa, Esosa Blessing Ekhoragbon, and Lai Tian Fat. Application of artificial intelligence in internet of things. pages 169–173, 2017. doi:10.1109/CICN.2017.8319379.

[12] Data Science 365. Overview of a neural network's learning process. 2023. `https://medium.com/data-science-365/overview-of-a-neural-networks-learning-process-61690a502fa`.

[13] Laith Alzubaidi, Jinglan Zhang, Ali Jamal Humaidi, Abdulmohsin Al-Dujaili, Yong Duan, Omar Al-Shamma, Javier Santamaría, Muhammad Ali Fadhel, Majdi Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(53), 2021. doi:10.1186/s40537-021-00444-8.

[14] S. M. Vanitha, V. Venkatesa Kumar, and S. Punitha. Edge detection: Concepts and techniques. *Processes*, 8(3):295, 2020. doi:10.3390/pr8030295.

[15] DataHacker. Edge detection: Concepts and techniques. `https://datahacker.rs/edge-detection/`, 2023.

[16] Rinku Rakshit, Dakshina Ranjan Kisku, Phalguni Gupta, and Jamuna Sing. Cross-resolution face identification using deep-convolutional neural network. *Multimedia Tools and Applications*, 80, 06 2021. doi:10.1007/s11042-021-10745-y.

[17] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. 2022. doi:10.48550/arXiv.2109.14545.

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. doi:10.1109/CVPR.2016.91.

[19] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014. doi:doi.org/10.48550/arXiv.1411.1792.

[20] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009. doi:doi.org/10.1109/TKDE.2009.191.

[21] Infraestruturas. Infraestruturas. `https://www.infraestruturasdeportugal.pt`, 2024. Accessed: 2024-05-15.

[22] Chien-Yao Wang and Hong-Yuan Mark Liao. YOLOv9: Learning what you want to learn using programmable gradient information. *arXiv preprint arXiv:2402.13616*, 2024.

[23] Python Software Foundation. Python (version 3.11) [computer software]. `https://www.python.org/`, 2024. Accessed: 2024-03-01.

[24] CVAT. Computer vision annotation tool. `https://app.cvat.ai`, 2024. Accessed: 2024-05-15.

[25] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolo, 2023. URL `https://ultralytics.com`.

[26] Ultralytics. Ultralytics (version 9.0) [python package]. `https://github.com/ultralytics/ultralytics`, 2024. Accessed: 2024-04-17.

[27] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017. doi:10.1109/CVPR.2017.690.

[28] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. 2018. doi:10.48550/arXiv.1804.02767.

[29] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. doi:10.48550/arXiv.2004.10934.

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context. 2015. doi:10.48550/arXiv.1405.0312.

[31] Dr. Priyanto Hidayatullah. Introduction to machine learning. YouTube, 2024. URL `https://www.youtube.com/watch?v=oZ6I1VHpil0&ab_channel=Dr.PriyantoHidayatullah`. Accessed: 2024-06-20.

[32] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. Real-time flying object detection with yolov8. 2024. doi:10.48550/arXiv.2305.09972.

[33] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. pages 7464–7475, 2023. doi:10.1109/CVPR52729.2023.00721.

[34] Kun Liu, Lei Peng, and Shanran Tang. Underwater object detection using tc-yolo with attention mechanisms. *Sensors*, 23(5), 2023. ISSN 1424-8220. doi:10.3390/s23052567.

[35] Facebook, Inc. React (version 18.0) [computer software]. `https://reactjs.org/`, 2024. Accessed: 2024-05-15.

[36] Node.js Foundation. Node.js (version 20.0) [computer software]. `https://nodejs.org/`, 2024. Accessed: 2024-05-15.

[37] Ecma International. Javascript (ecmascript 2024) [computer software]. `https://www.ecma-international.org/publications-and-standards/standards/ecma-262/`, 2024. Accessed: 2024-05-15.

[38] World Wide Web Consortium (W3C). Cascading style sheets (css) [computer software]. `https://www.w3.org/Style/CSS/`, 2024. Accessed: 2024-05-15.