AUTOMATIC ROAD DETECTION USING OBJECT ORIENTED DEEP LEARNING ALGORITHMS AND GLOBAL TRAINING DATA

by

Ahmed Nabil Elbahlol Ahmed Ali

A thesis submitted to the Faculty of
Engineering-Shoubra, Benha University in partial fulfillment of the
requirements for the Master Degree of Geomatics Engineering, In The Specialty of
Photogrammetry and Remote Sensing

Supervised by

Prof. Mahmoud M. Hamed
Professor of Surveying and Photogrammetry
Faculty of Engineering, Shoubra
Cairo, Egypt

Prof. Mahmoud Salah
Professor of Surveying and Photogrammetry
Faculty of Engineering, Shoubra
Cairo, Egypt

Faculty of Engineering, Shoubra, Benha University
Cairo, Egypt

September 2025

Acknowledgements

I would like to express my gratitude to Prof. Mahmoud Hamed for his guidance and support throughout this research. I am deeply indebted to Prof. Mahmoud Salah, whose invaluable insights, continuous encouragement, and unwavering belief in my potential have been instrumental in the success of this work. His expertise and mentorship have profoundly shaped this study, and I am truly honored to have had the privilege of working under his guidance. Additionally, I extend my thanks to everyone who contributed to this study and supported me in any way.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Abbreviations and Symbols

AI	Artificial Intelligence
ANNs	Artificial Neural Networks
AP	Average Precision
API	Application Programming Interface
ASPP	Atrous Spatial Pyramid Pooling
AUC	Area Under Curve
AWS	Amazon Web Services
BCC	Background Detection Correctness Coefficient
BN	Batch Normaliztion
BNN	Back-propagation Neural Network
CLI	Command Line Interface
CMD	Command Prompt
CNNs	Convolutional Neural Networks
COCO	Common Objects in Context
CV	Computer Vision
DL	Deep Learning
FAIR	Facebook's AI Research lab
FCN	Fully Convolutional Neural Network
FNNs	Feedforward Neural Networks
FPS	Frame Per Second
GCS	Geographic Coordinate System
GIS	Geographical Information Systems
GPS	Global Positioning System
GPU	Graphical Processing Unit
GRD	Ground Range Detected
HSI	Hyperspectral Image
IoU	Intersection over Union
IW	Interferometric Wide
kNN	k-Nearest Neighbor
LSTM	Long Short-Term Memory
LU/LC	Land use and Land cover
ML	Maximum Likelihood
MLP	Multilayer Perceptron
MRF	Markov Random Fields
MS	Multi-Spectral
MSE	Mean Square Error
MTRE	Multi-Task Road Extractor
NAG	Nesterov Accelerated Gradient
NLP	Natural Language Processing
PAN	Panchromatic
PR	Precision-Recall
RCC	Road Detection Correctness Coefficient

R-CNN	Region-based Convolutional Neural Networks
ReLU	Rectified Linear Unit
ResUnet	Residual U-Net
RMSE	Root Mean Square Error
RNNs	Recurrent Neural Networks
RPN	Region Proposal Network
RS	Remote Sensing
SA	Simulated Annealing
SAR	Synthetic Aperture Radar
SE	Squeeze-and-Excitation
SGD	Stochastic Gradient Descent
SOM	Self-Organizing Mapping
SVMs	Support Vector Machines
TM	Thematic Mapper
UAVs	Unmanned Aerial Vehicles
YOLO	You Only Look Once

Abstract

Automatic road extraction from satellite imagery plays a vital role in remote sensing and urban planning. Its applications span transportation network analysis, infrastructure development, and smart city solutions. This thesis introduces a detailed methodology for road detection by combining object-oriented and pixel-based deep learning techniques, specifically integrating the Faster R-CNN architecture with the Multi-Task Road Extractor model, to improve road detection accuracy.

The study uses SpaceNet imagery data, focusing on urban environments, to train and evaluate the models. The Faster R-CNN model is first utilized to identify candidate road regions within the imagery. To enhance these results, the Multi-Task Road Extractor model is applied, employing a shared encoder that performs road segmentation and classification simultaneously. This dual approach enhances both pixel-level accuracy and the recognition of road attributes.

The experimental outcomes validate the effectiveness of the two models, showcasing notable improvements in average precision (AP) across various intersection-over-union (IoU) thresholds. Specifically, the Faster R-CNN model achieves an AP of 0.557 at a 0.6 detection threshold, while the training metrics for the Multi-Task indicated high performance, with accuracy reaching 0.986 and Dice coefficient achieving 0.861.

This study can support urban planning, smart city initiatives, and infrastructure development, providing a solid foundation for future work. Future research will focus on utilizing multi-source geospatial data, such as UAV imagery and laser scanning, to achieve the same objectives with enhanced precision.

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 Motivation

The current era is witnessing an Artificial Intelligence (AI) revolution that may surpass the impact of electricity. Deep Learning (DL), a subset of AI, is advancing frameworks aimed at achieving a comprehensive understanding of images, a field commonly referred to as Computer Vision (CV). Computer Vision encompasses tasks that are closely aligned with those in Remote Sensing (RS), such as classification, object detection, and more. The current trend in this era is leveraging DL to perform traditional RS tasks, which serve a vast range of applications across diverse fields including agriculture, industry, weather forecasting, mineral extraction, and infrastructure development.

One of the most prominent applications in infrastructure and urban planning is road extraction from aerial and satellite imagery. This task is critical for smart cities and plays a crucial role in disaster management and urban sustainability. Roads, as integral components of our environment, further emphasize the importance of this application, extending its relevance to environmental sustainability.

Road networks are integral to Geographical Information Systems (GIS), where software solutions are specifically designed to manage and analyze road networks. These solutions often rely on databases of road networks, which are frequently generated through road extraction processes.

Road extraction from images is a well-established task, with numerous algorithms and frameworks yielding varied results over the years. However, traditional methods often rely on labor-intensive and repetitive manual work. The advancements in computer hardware, which have fueled the resurgence of DL, have shifted the focus in research and professional practice towards leveraging DL for road detection, with the goal of achieving higher accuracy. In today's fast-paced and automated world, where drones can execute numerous applications, significant efforts have been made in CV to achieve real-time object detection, allowing DL algorithms to detect objects within seconds or less.

Road detection in satellite or aerial imagery falls under the domain of object detection in CV. However, the distinct characteristics of satellite images, compared to visual images, necessitate tailored approaches to address their unique challenges. Consequently, road detection from satellite images using DL remains an area requiring extensive research to achieve reliable, automatic, and real-time detection.

1.2 Fundamental Concepts

Satellite imagery is a highly suitable data source for extracting elongated features like roads. Numerous algorithms have been developed to extract roads from both satellite and aerial imagery. Early methods relied on traditional image processing techniques, such as edge detection, thresholding, and region-growing algorithms, to identify roads. Widely used classification methods, including Artificial Neural Networks (ANN), Support Vector Machines (SVM), Markov Random Fields (MRF), and Maximum Likelihood (ML), have also been applied extensively. However, these methods often face challenges in distinguishing roads from visually similar linear features (Weixing et al., 2016).

ANNs depend heavily on manual feature engineering or learn features inefficiently, requiring a large number of neurons and layers to process high-dimensional data like images. This inefficiency stems from their inability to exploit the spatial hierarchies and pixel correlations inherent in image data, such as edges and corners. Additionally, ANNs require images to be flattened into one-dimensional vectors, which destroys the two-dimensional spatial structure of the image and limits their effectiveness for image-related tasks.
In contrast, Convolutional Neural Networks (CNNs) are specifically designed to handle image data. They employ convolutional layers that preserve spatial hierarchies and local patterns, such as edges, textures, and objects, by using filters (kernels) that slide across the image. This approach maintains the 2D spatial relationships between pixels. CNNs automatically extract

features through convolutional operations, pooling layers, and activation functions, enabling them to capture simple patterns (e.g., edges) in early layers and progressively build more complex patterns (e.g., objects) in deeper layers. Their architecture is more efficient, requiring fewer parameters due to weight sharing in convolutional layers, and can handle large images effectively while reducing computational complexity using pooling layers like max pooling to downsample feature maps.

This study utilizes two state-of-the-art deep larning models, Faster R-CNN and the Multi-Task Road Extractor (MTRE), both of which are based on Convolutional Neural Networks (CNNs). These models are trained to identify roads, enabling them to learn the unique characteristics of road features and develop the capability to automatically detect roads in new satellite imagery.

1.3 Objectives

The primary objective of this study is to apply deep learning models for detecting roads from satellite imagery. Specifically, this objective will be achieved through the following steps:

- **Data Downloading:** Mastering Amazon Web Services (AWS) command-line tools to retrieve large-scale satellite imagery datasets.
- **Data Processing:** Managing and converting large volumes of files, including satellite images and GeoJSON files, to prepare them for model training.
- **Environment Setup:** Building a computational environment suitable for training deep learning models, including necessary libraries and frameworks.
- **Training the First Deep Learning Model:** Implementing Faster R-CNN and refining its weights through extensive training to capture road features accurately.
- **Performance Measurement of Faster R-CNN:** Evaluating the model's performance using object detection metrics such as training loss, validation loss, and detection thresholds.
- **Enhancing Results with MTRE:** Fine-tuning the Multi-Task Road Extractor (MTRE) model to optimize its parameters for the study area, enhancing the results obtained from Faster R-CNN.
- **Performance Measurement of MTRE:** Assessing the MTRE model's performance using advanced metrics, including training loss, validation loss, accuracy, and Intersection over Union (IoU) thresholds.

1.4 Outline of The Thesis

The structure of the thesis is organized as follows:

Chapter One introduces the overall context and motivation of the study. It outlines the key concepts related to road detection using deep learning, defines the research objectives, and presents the structure of the thesis.

Chapter Two provides background information relevant to the research. It begins with an overview of deep learning, followed by its applications in Remote Sensing (RS) and Computer Vision (CV). Subsequently, the chapter reviews existing deep learning approaches for road detection, summarizes previous methods used for extracting roads from satellite images, including approaches based on neural networks, and discuss the gaps in the previous literature.

Chapter Three introduces the theoretical background and concepts of the selected algorithms for road detection, namely Faster R-CNN and Multi-Task Road Extractor (MTRE).

Chapter Four focuses on data preparation. It begins by detailing the required AWS command lines for downloading data, describes the study areas and data sources, and introduces a novel approach for utilizing pre-processed data, eliminating the need for manual annotation.

Chapter Five provides a comprehensive overview of the training process for each model, detailing the frameworks employed, hardware specifications, and key factors affecting model performance. It also presents the results and analysis of the experiments, offering a comparative evaluation between the proposed model and three recent state-of-the-art deep learning approaches.

Chapter Six concludes the dissertation by providing a comprehensive summary of the step-by-step process involved in training two deep learning models, Faster R-CNN and MTRE, for a specific study area. It emphasizes the capability of these models to automatically detect roads from previously unseen satellite imagery. Additionally, recommendations and future research directions are outlined, with a focus on addressing environmental sustainability and advancing the application of these models in real-world scenarios.

To provide a clear overview of the research workflow, Figure 1.1 presents a flowchart that summarizes the main steps and processes of this thesis, which are further elaborated in the methodology chapter.

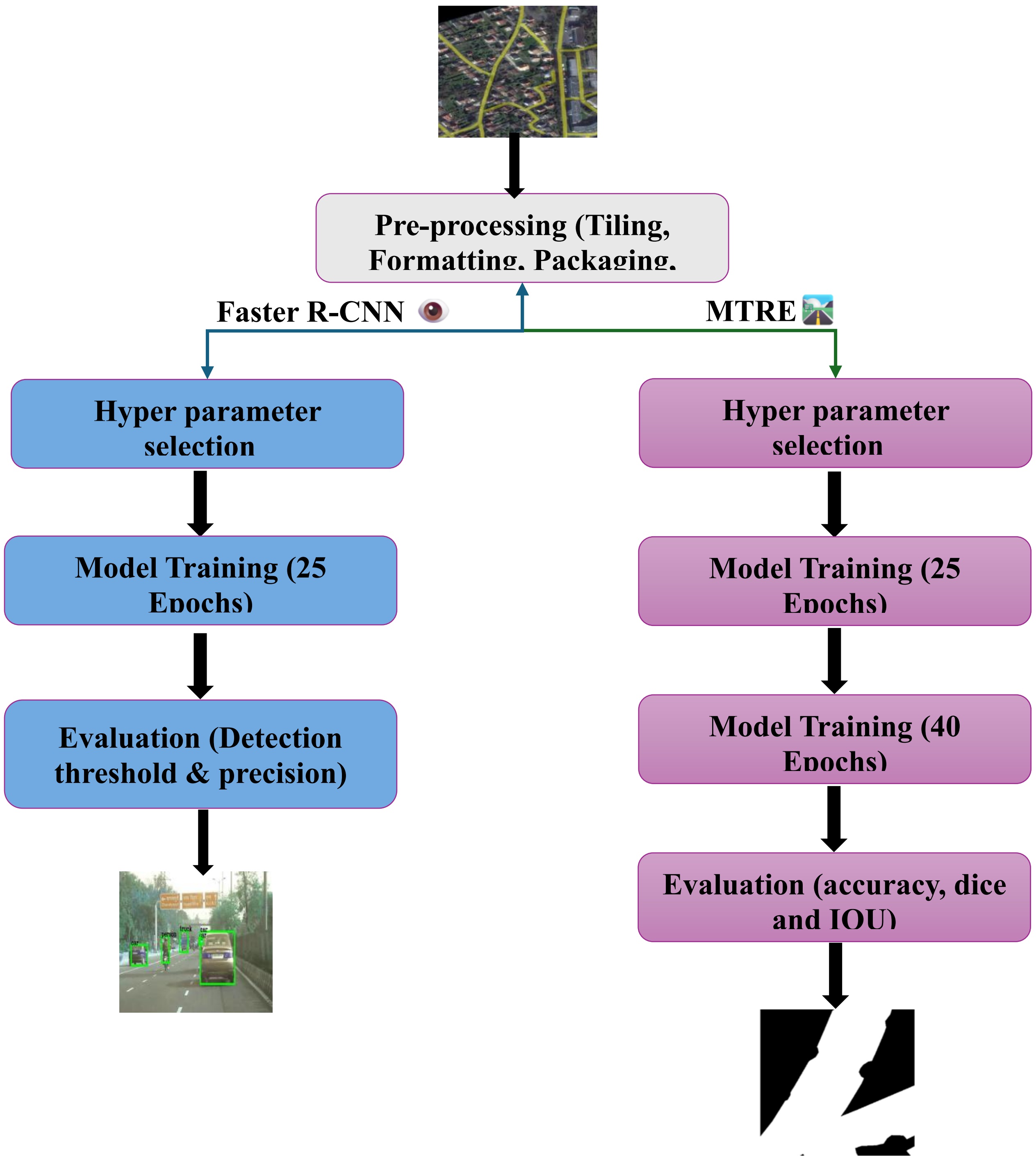

Pre-processing (Tiling, Formatting, Packaging,
Faster R-CNN
MTRE
Hyper parameter selection
Model Training (25 Epochs)
Evaluation (Detection threshold & precision)
Hyper parameter selection
Model Training (25 Epochs)
Model Training (40 Epochs)
Evaluation (accuracy, dice and IOU)

Figure 1.1: Overall Workflow of the Proposed Methodology

CHAPTER 2

BACKGROUND & LITERATURE REVIEW

CHAPTER 2

BACKGROUND & LITERATURE REVIEW

2.1 Introduction

This chapter provides an overview of the fundamental concepts and background information necessary to understand the research. The chapter is divided into three main sections: an introduction to deep learning, its applications in remote sensing (RS), and previous deep learning methods used for extracting roads from satellite images. These sections set the foundation for the integration of deep learning algorithms with geospatial data to develop solutions for applications such as urban planning, environmental sustainability and smart cities.

2.2 Overview of Neural Networks and Deep Learning

2.2.1 Neural Networks (NNs)

Deep learning is a subset of machine learning, which itself is a branch of artificial intelligence (AI). It utilizes artificial neural networks with multiple layers to learn from large datasets. The concept of deep learning mimics the human brain's ability to recognize patterns, make predictions, and learn progressively. It is theorized that the human brain can think and function as a natural biological system through the activity of neurons. Figure 2.1 illustrates the process of neuronal communication within the human brain. Neurons transmit electrical impulses to one another via axons, which serve as the connecting pathways between these cells.

An artificial neural network consists of layers of interconnected nodes (neurons), organized into three types of layers:

- **Input layer:** Receives the raw data (e.g., images or geospatial information).
- **Hidden layers:** Intermediate layers where computations occur. These layers extract features from the data through weights and activation functions.
- **Output layer:** Produces the final prediction or classification result.

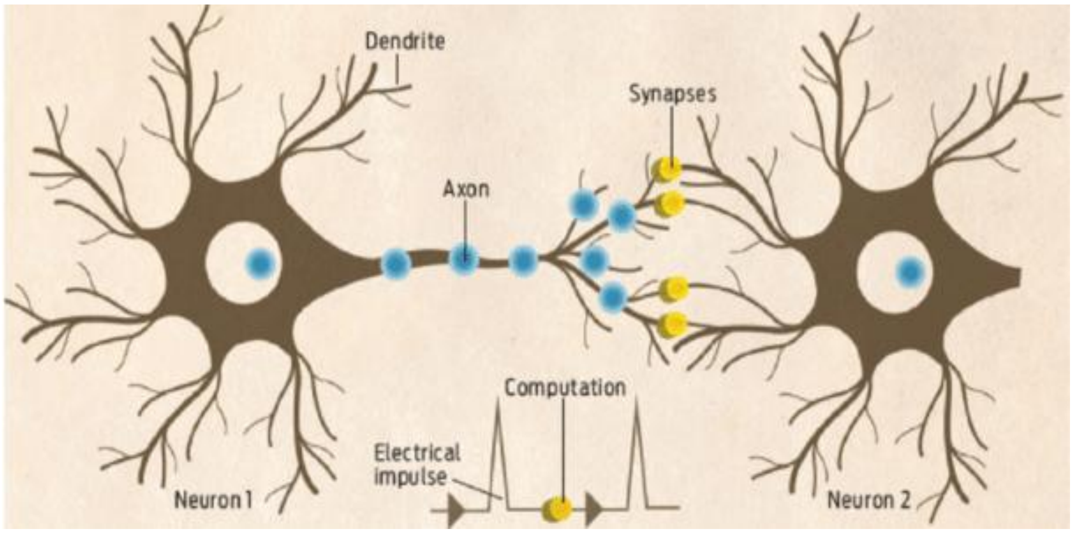


Figure 2.1: Neuronal communication in the human brain, showing how electrical impulses are transmitted between neurons via axons. (Wang, Lu, & Wen, 2017).

Figure 2.1 depicts a feedforward neural network, where it consists of multiple interconnected layers of nodes, or neurons. In the input layer the raw data (e.g., images, text, numerical data) is fed into the network. Each node in this layer represents a feature of the input data. In the hidden layers, the core of the network, they perform computations on the input data and extract increasingly complex features. Each node in a hidden layer receives weighted inputs from the previous layer, applies an activation function (like ReLU, sigmoid, or tanh), and passes the output to the next layer. The number of hidden layers and the number of nodes within each layer determine the network's capacity to learn complex patterns. The output layer produces the final output of the network. The number of nodes in the output layer depends on the task. For example, in image classification, there would be one node for each class (e.g., cat, dog, bird). In regression tasks, there might be a single output node representing the predicted value.

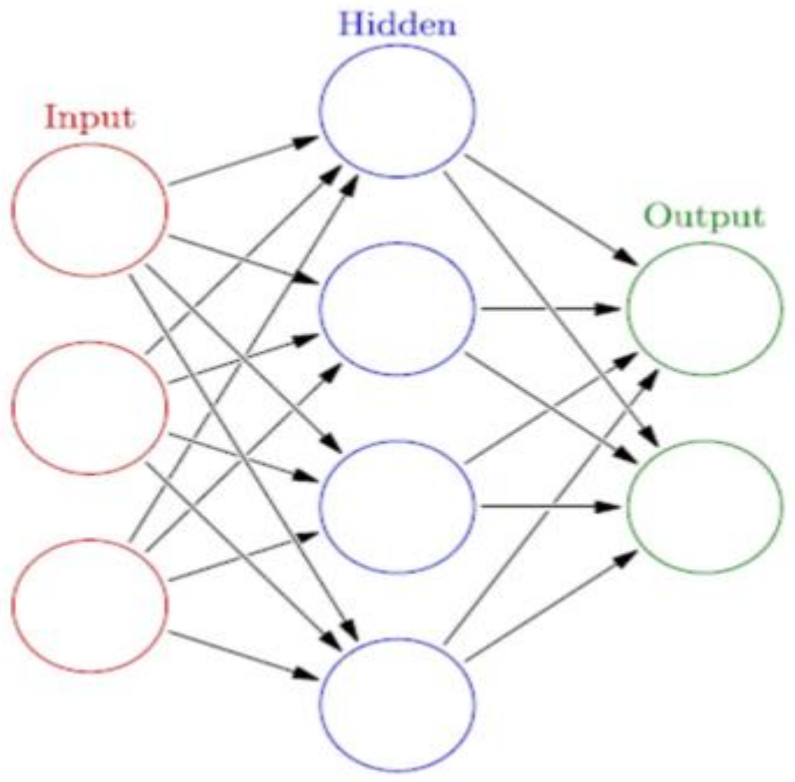


Figure 2.2: A feedforward neural network with input, hidden, and output layers (Wu & Feng, 2017)

In a neural network, input data is presented to the input layer and propagated through the network in a process called forward propagation, which occurs layer by layer. Each node in a hidden layer computes a weighted sum of its inputs and applies an activation function. The output of the final hidden layer is then used to compute the network's prediction. If the network's output does not match the desired output, an error is calculated. This error is subsequently used to adjust the weights of the connections between the nodes through an optimization algorithm such as gradient descent. This iterative process, which continues until the network's performance reaches an acceptable level, is referred to as backpropagation.

The connections between nodes in the network are assigned weights, which are learned during the training process. Activation functions introduce non-linearity into the network, enabling it to learn and represent complex patterns.

Deep learning models with multiple hidden layers have the ability to learn intricate and abstract representations from data. This capability allows them to excel in challenging tasks such as image recognition, natural language processing, and more.

2.2.2 Activation Functions

An activation function is a mathematical function applied to the weighted sum of inputs (plus a bias) in a neuron. It introduces non-linearity into the network, allowing it to learn and model complex relationships in data. Without activation functions, a neural network would simply be a linear regression model, incapable of handling tasks like image recognition, natural language processing, or any other non-linear problem.

Activation functions are characterized by several key properties that determine their effectiveness in neural networks. First and foremost, they introduce non-linearity, enabling the network to model complex, real-world data patterns that linear transformations alone cannot capture. This non-linearity is essential for tasks like image recognition, natural language processing, and other advanced machine learning applications. Additionally, activation functions must be differentiable to support gradient-based optimization methods like backpropagation, which are fundamental to training neural networks. Finally, computational efficiency is a practical consideration, as activation functions should be computationally inexpensive to evaluate, especially in deep networks with millions of parameters. Together, these properties ensure that activation functions not only enhance the learning capability of neural networks but also maintain efficiency and stability during training. There are many activation functions, some common activation functions are:

- **Sigmoid (Logistic Function)**

Equation: $\sigma(x) = \frac{1}{1+e^{-x}}$

Range: (0, 1)

Graph: Smooth S-shaped curve as in figure 2.3.

Pros: Outputs are interpretable as probabilities.

Cons: Suffers from the vanishing gradient problem (gradients approach zero for very high or low inputs) and computationally expensive due to exponential operations.

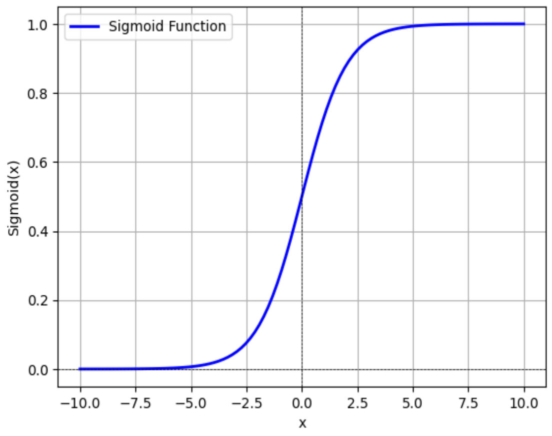


Figure 2.3: Sigmoid function

Use Case: Historically used in binary classification tasks.

- **Hyperbolic Tangent (tanh)**

Equation: tanh(x) = $\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

Range: (-1, 1)

Graph: S-shaped curve centered at zero as in figure 2.4.

Pros: Zero-centered, which helps with convergence during training.

Cons: Still suffers from the vanishing gradient problem.

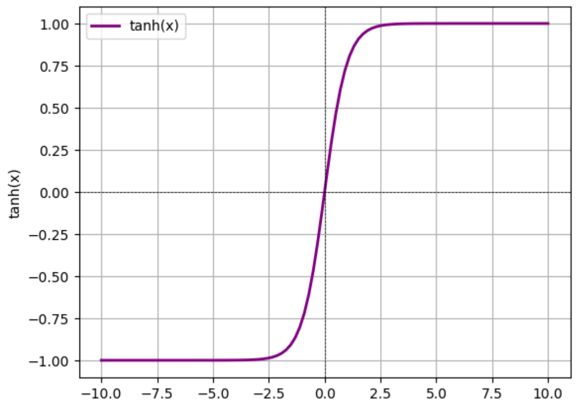


Figure 2.4: Hyperbolic Tangent

Use Case: Often used in hidden layers of neural networks.

- **Rectified Linear Unit (ReLU)**

Equation**:** ReLU(x) = max(0,x)

Range**:** (0, ∞)

Graph: Linear for positive inputs, zero for negative inputs as in figure 2.5.

Pros: Computationally efficient and Mitigates the vanishing gradient problem for positive inputs.

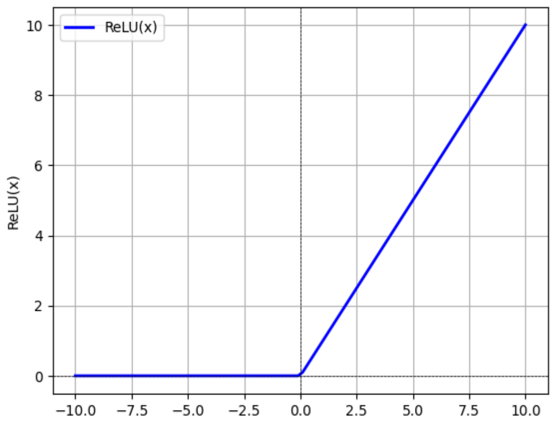


Figure 2.5: ReLU function

Cons: Suffers from the dying ReLU problem (neurons can get stuck outputting zero).

Use Case: Most commonly used in hidden layers of deep neural networks.

- **Leaky ReLU**

Equation: LeakyReLU(x) = max(0.01x,x)

Range: (-∞, ∞)

Graph: Similar to ReLU but with a small slope for negative inputs as in figure 2.6.

Pros: Addresses the dying ReLU problem.

Cons: Requires tuning of the slope parameter.

Use Case: Alternative to ReLU in deep networks.

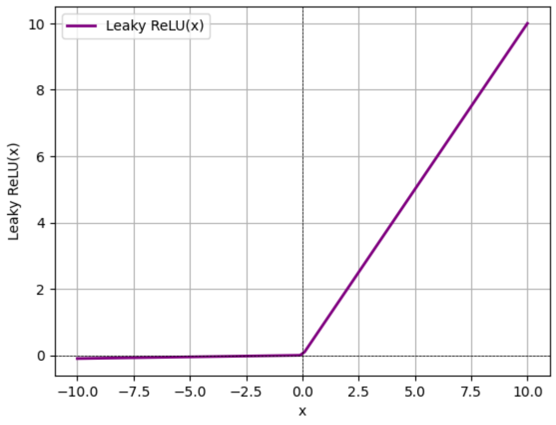


Figure 2.6: Leaky ReLU function

- **Softmax**

Equation: Softmax(x_i) = $\frac{e^{xi}}{\sum_{j=1}^{n} e^{xj}}$

Range: (0, 1)

Graph: Converts logits into probabilities as in figure 2.7.

Pros: Outputs are interpretable as probabilities.

Cons: Computationally expensive for large outputs.

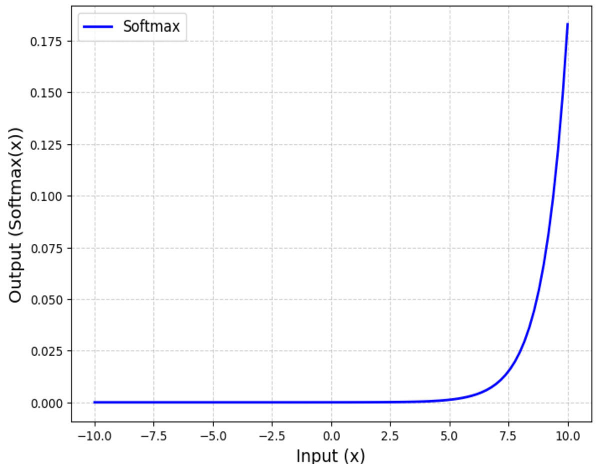


Figure 2.7: Softmax function

Use Case: Used in the output layer for multi-class classification.

2.2.3 Feedforward Neural Networks (FNNs):

Feedforward Neural Networks (FNNs) are the simplest type of artificial neural networks, where information flows in one direction from the input layer, through hidden layers, to the output layer, without any cycles or loops (Goodfellow et al., 2016). They are composed of interconnected neurons, each applying a weighted sum of inputs followed by a non-linear activation function, such as sigmoid or ReLU, to introduce complexity into the model (Haykin, 1999). FNNs are widely used for tasks like regression and classification, where the relationship between inputs and outputs is relatively straightforward (Bishop, 2006). However, they lack the ability to handle sequential or spatial data, which limits their applicability in domains like image processing or time-series analysis (LeCun et al., 2015). Despite their simplicity, FNNs serve as the foundation for more advanced architectures, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), which address their limitations (Schmidhuber, 2015). Their training typically involves backpropagation and gradient descent to minimize the error between predicted and actual outputs (Rumelhart et al., 1986). While FNNs are computationally efficient for small-scale problems, they may struggle with scalability and overfitting in large, complex datasets (Bengio et al., 2013). Overall, FNNs remain a fundamental tool in the machine learning toolkit, particularly for introductory and baseline modeling tasks.

2.2.4 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural networks designed to handle sequential data by maintaining a "memory" of previous inputs through cyclic connections within the network (Goodfellow et al., 2016). Unlike Feedforward Neural Networks, RNNs process inputs sequentially, making them well-suited for tasks such as time-series analysis, speech recognition, and natural language processing (Graves, 2012). Each neuron in an RNN receives input not only from the current time step but also from its own output at the previous time step, allowing it to capture temporal dependencies and patterns in data (Hochreiter & Schmidhuber, 1997). However, standard RNNs often struggle with long-term dependencies due to issues like vanishing or exploding gradients, which hinder their ability to learn relationships over extended sequences (Bengio et al., 1994). To address these limitations, advanced variants such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) were introduced, which incorporate gating mechanisms to better preserve and control information flow (Cho et al., 2014). Despite their challenges, RNNs have been instrumental in advancing applications like machine translation, text generation, and video analysis (Sutskever et al., 2014). Their ability to model sequential data makes them a cornerstone of modern deep learning, particularly in domains where context and order are critical (LeCun et al., 2015). However, their computational complexity and training difficulties have led to the rise of alternative architectures, such as Transformers, which offer improved scalability and performance (Vaswani et al., 2017).

2.2.5 Transformers

Transformers are a revolutionary architecture in deep learning, introduced to address the limitations of Recurrent Neural Networks (RNNs) in handling long-range dependencies and sequential data (Vaswani et al., 2017). Unlike RNNs, Transformers rely entirely on self-attention mechanisms, which allow them to process entire sequences in parallel, significantly improving computational efficiency and scalability. This parallelization enables Transformers to capture relationships between distant elements in a sequence, making them highly effective for tasks like machine translation, text summarization, and natural language understanding (Devlin et al., 2019). The architecture consists of an encoder-decoder structure, where multi-head attention layers weigh the importance of different parts of the input sequence, followed by feedforward layers for further processing. Transformers have become the foundation of state-of-the-art models like BERT, GPT, and T5, which dominate benchmarks in natural language processing (NLP) due to their ability to learn contextual representations of text (Brown et al., 2020) . Beyond NLP, Transformers are also being applied to computer vision, audio processing, and even protein structure prediction, demonstrating their versatility. Despite their success, Transformers require substantial computational resources and large datasets for training, which can be a limitation for some applications. Nevertheless, their ability to model complex relationships in data has made them a cornerstone of modern artificial intelligence.

2.2.6 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a specialized class of neural networks designed to process grid-like data, such as images, by leveraging convolutional layers to extract spatial hierarchies of features (LeCun et al., 2015). Unlike traditional neural networks, CNNs use shared weights and local receptive fields, making them computationally efficient and capable of capturing patterns like edges, textures, and shapes (Goodfellow et al., 2016). These properties have made CNNs the backbone of modern computer vision, powering applications such as image classification, object detection, and facial recognition. Their ability to automatically learn relevant features from raw data, combined with architectural innovations like pooling layers and activation functions, has established CNNs as a cornerstone of deep learning. In this work, particular emphasis is placed on CNNs, as they form the foundation of the two proposed algorithms to be trained. CNNs are chosen for their exceptional ability to capture spatial hierarchies and patterns in data, making them highly suitable for the tasks addressed in this study. Their architecture and performance characteristics are central to the development and effectiveness of the proposed methodologies.

2.3 Applications of Deep Learning in Remote Sensing

Deep learning has revolutionized remote sensing by enabling the automated analysis of large-scale, high-dimensional, and heterogeneous satellite and aerial imagery data. Applications span a wide range of domains, from environmental monitoring and land-use classification to disaster response and precision agriculture. The advancements in deep learning models, particularly CNNs and transformers, have enabled accurate feature extraction, pattern recognition, and real-time decision-making in remote sensing.

This section explores the key applications of deep learning in remote sensing, highlights commonly used models and datasets, and provides examples with case studies from recent literature.

2.3.1 Land Cover Classification and Mapping

Land cover classification involves categorizing satellite images into classes such as water, forest, agriculture, or urban areas. Deep learning, particularly CNNs, has significantly improved classification accuracy by learning spatial and spectral features from multi-spectral or hyperspectral imagery. For example, a study by Shah et al. (2025). implemented a spatio-temporal deep learning model for enhanced atmospheric correction, which was specifically tailored for land cover classification tasks. This model utilized high-resolution Sentinel-2 imagery as input and employed U-Net architectures for semantic segmentation. The integration of spatio-temporal data proved particularly effective in enhancing classification accuracy for complex, heterogeneous landscapes.

Figure 2.8 illustrates the workflow for land cover classification using a Convolutional Neural Network (CNN). The process begins with input satellite imagery, followed by preprocessing steps such as normalization and noise reduction. Features are then extracted using CNN layers, including convolutional and pooling operations. Finally, the classification layer assigns probabilities to different land cover categories, producing the final categorized outputs.

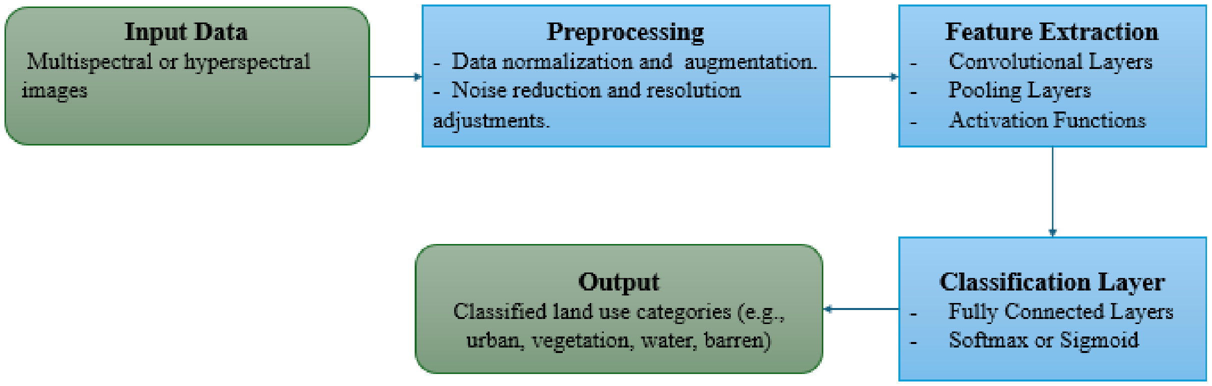

Input Data
Multispectral or hyperspectral images
Preprocessing
- Data normalization and augmentation.
- Noise reduction and resolution adjustments.
Feature Extraction
- Convolutional Layers
- Pooling Layers
- Activation Functions
Classification Layer
- Fully Connected Layers
- Softmax or Sigmoid
Output
Classified land use categories (e.g., urban, vegetation, water, barren)

Figure 2.8: Land cover classification workflow using a CNN, from input imagery to final land use categories.

Yuan et al. (2009) introduced an automated artificial neural network (ANN) system designed for land use and land cover (LU/LC) classification using Landsat Thematic Mapper (TM) imagery. The system comprises two primary modules:

I. Unsupervised Kohonen's Self-Organizing Mapping (SOM) Neural Network: This module ,as illustrated in figure 2.9, clusters input data without prior labeling, effectively identifying inherent patterns within the dataset. Two training algorithms were implemented:

- Standard SOM Algorithm: Employs traditional competitive learning for clustering.
- Refined SOM with Simulated Annealing (SA): Integrates SA to enhance the optimization process, aiming to achieve a global or near-global optimum by mitigating the local minima problem.

II. Supervised Multilayer Perceptron (MLP) Neural Network: Utilizes the Backpropagation (BP) training algorithm to perform classification based on labeled training data.

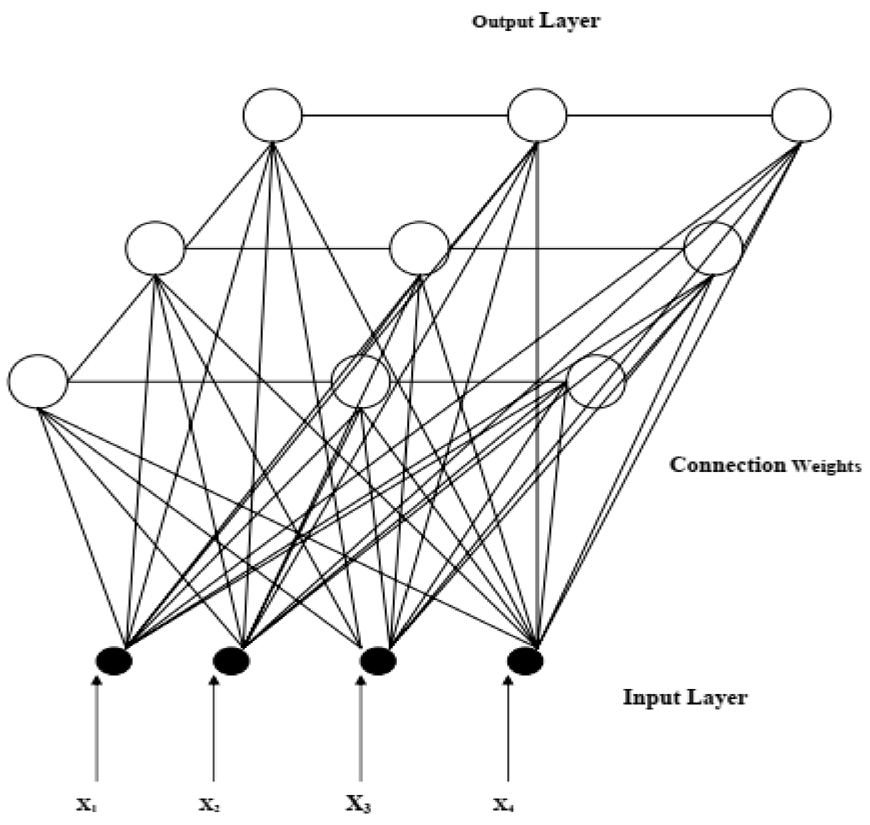


Figure 2.9 The structure of Kohonen’s Self-Organization Mapping neural network *(Yuan et al., 2009)*

The system was tested on Landsat TM imagery, a widely used dataset in remote sensing for LU/LC classification tasks. The ANN system successfully automated the processes of network training pattern creation, training, and generalization. The incorporation of the refined SOM with SA demonstrated improved classification accuracy compared to the standard SOM, highlighting the effectiveness of the SA-enhanced learning algorithm.

Figure 2.10 illustrates the architecture of the ANN system, depicting the flow from input data through the SOM and MLP modules, culminating in the LU/LC classification output.

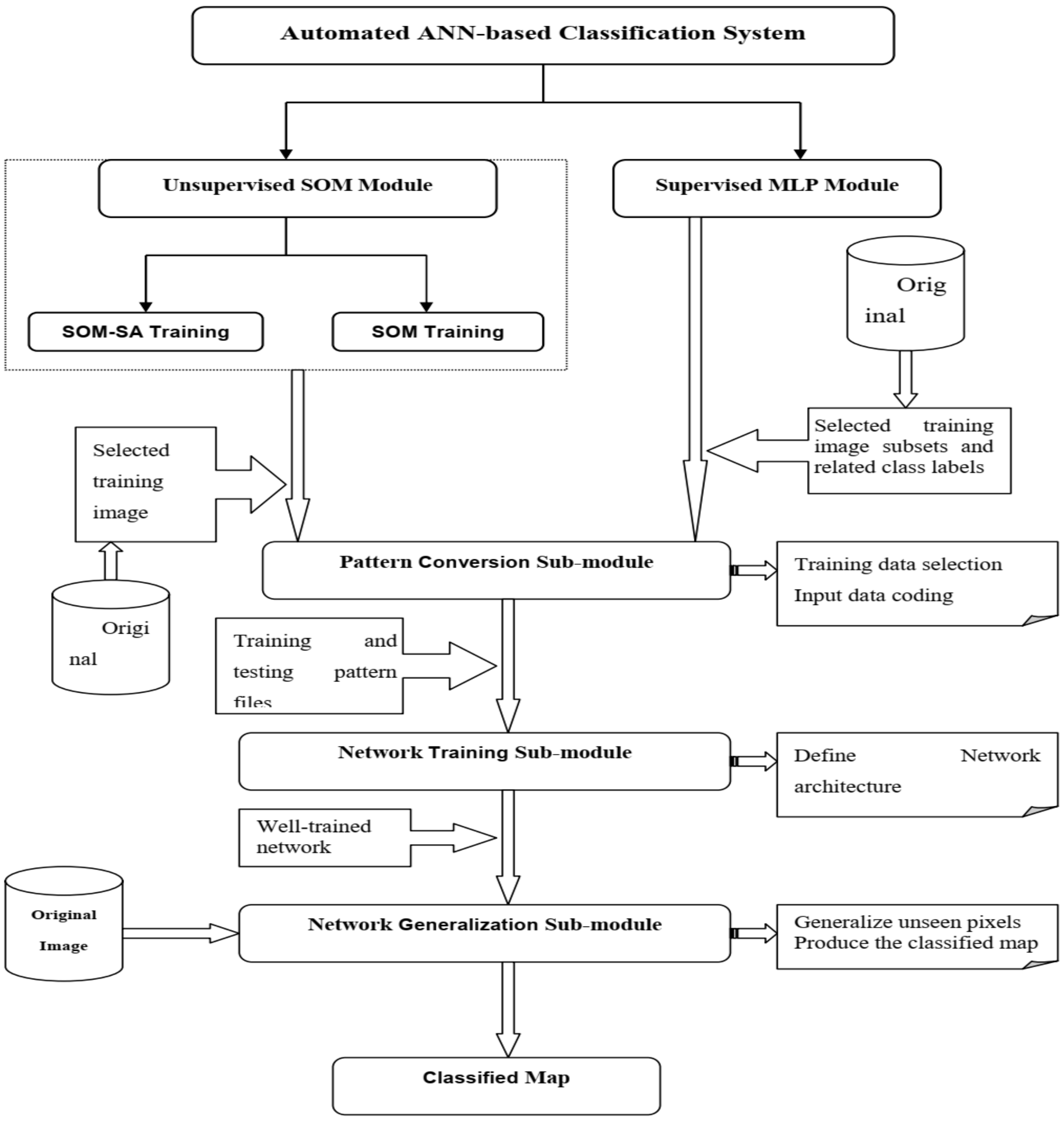

Automated ANN-based Classification System
Unsupervised SOM Module
Supervised MLP Module
SOM-SA Training
SOM Training
Original
Selected training image subsets and related class labels
Selected training image
Original
Pattern Conversion Sub-module
Training data selection
Input data coding
Training and testing pattern files
Network Training Sub-module
Define Network architecture
Well-trained network
Original Image
Network Generalization Sub-module
Generalize unseen pixels
Produce the classified map
Classified Map

Figure 2.10 Flowchart of the ANN-based classification system (Yuan et al., 2009)

2.3.2 Object Detection and Monitoring

Object detection in remote sensing focuses on identifying and localizing specific features, such as buildings, vehicles, or ships, within imagery. Algorithms like YOLO (You Only Look Once) and RetinaNet have been employed to handle the challenges posed by the variability in scale, angle, and resolution of objects in satellite imagery. For instance, Feng et al. (2024) introduced an edge-self-attention mechanism integrated into large convolution kernels for ship detection in Synthetic Aperture Radar (SAR) imagery. This approach addressed the complexities of SAR data, including noise and interference, and achieved high accuracy in maritime surveillance and disaster response scenarios. These models demonstrate the robust potential of deep learning for object detection applications in both structured urban environments and unstructured natural settings.

Table 2.1 presents the performance of four object detection models—YOLOv5L, YOLOv4, YOLOv3, and Faster R-CNN—on the SAR dataset, evaluated using two key metrics, precision: which measures the proportion of correctly detected objects (true positives) out of all detected objects. YOLOv5L achieves the highest precision (97.1%), followed closely by YOLOv4 (96.0%) and YOLOv3 (96.2%). Faster R-CNN exhibits a significantly lower precision (67.0%), indicating a higher rate of false positives compared to YOLO models, and recall that represents the proportion of correctly detected objects (true positives) out of all ground truth objects. Faster R-CNN achieves the highest recall (93.6%), slightly outperforming YOLOv5L (93.2%). YOLOv4 and YOLOv3 perform moderately well, with recall values of 91.0% and 89.2%, respectively.

Table 2.1 Precision-to-recall ratios for different models (Bachir& Memon 2024).

Model	**Precision**	**Recall**
YOLOv5L	0.971	0.932
YOLOv4	0.96	0.91
YOLOv3	0.962	0.892
Faster R–CNN	0.67	0.936

The results indicate that YOLOv5L strikes the best balance between precision and recall, making it highly effective for object detection tasks. While Faster R-CNN demonstrates strong recall, its lower precision suggests potential over-detection and false positives. Conversely, YOLO models maintain a strong trade-off, especially YOLOv5L, which leads in precision and is nearly on par in recall.

2.3.3 Hyperspectral Image (HSI) Analysis

Hyperspectral imaging collects detailed spectral information across hundreds of narrow bands, enabling the differentiation of materials at a pixel level. However, analyzing HSI data is computationally intensive due to its high dimensionality. Deep learning has been pivotal in reducing this complexity while maintaining high accuracy. Tu et al. (2024) implemented an adaptive feature self-attention network in Spiking Neural Networks for hyperspectral image classification. Their method tackled the challenge of computational intensity by introducing an energy-efficient neural network architecture. The study highlights the growing focus on designing deep learning solutions tailored for resource-constrained environments, a critical consideration in the field of hyperspectral remote sensing.

2.3.4 Precision Agriculture

Deep learning models analyze remote sensing data to monitor crop health, predict yields, and detect diseases or pests. UAVs equipped with spectral sensors provide high-resolution imagery that, combined with models like ResNet or LSTM, can yield actionable insights for farmers. In a comprehensive review, Ramalingam et al. (2024) evaluated the use of YOLO for object detection in precision agriculture. This study emphasized the ability of YOLO to classify and detect crops, pests, and diseases in real-time, making it a valuable tool for addressing the challenges of modern agricultural management. The integration of drone technology with real-time deep learning models further enhances the potential of precision agriculture to optimize yields and resource usage.

2.4 previous deep learning methods used for extracting roads from satellite images

Baumgartner et al. (1999) proposed a method for the automatic extraction of roads from digital aerial imagery. Their approach leverages multi-resolution analysis to balance detailed and generalized information. High-resolution images (0.2 to 0.5 meters) are used to detect edges, while lower-resolution images are utilized to identify lines. This multi-scale strategy helps formulate initial road segment hypotheses, which are subsequently combined into larger road segments through the application of explicit road-related knowledge. The method incorporates a hierarchical contextual framework, distinguishing between global and local contexts. Globally, the aerial images are segmented into categories such as rural, forested, or urban areas to target the most relevant regions for road extraction. Locally, context-specific features, such as shadows cast by trees, are used to close gaps in the detected road network. Finally, intersections are identified to create a complete representation of the road network.

To evaluate their method, the researchers tested it on digital aerial images of varying resolutions. The results, validated against manually generated reference data, demonstrated the method's capability to accurately extract road networks.

The proposed approach effectively addresses several challenges in road extraction:

- Multi-Scale Analysis: Using different image resolutions enables the extraction of both fine-grained details and broader road structures.
- Contextual Information: Combining global and local context enhances extraction accuracy and bridges gaps caused by obstructions like shadows.
- Iterative Grouping: The iterative grouping of road segments ensures the construction of a more complete and continuous road network.

Valadan Zoej and Mokhtarzade (2006) presented a study that explores the use of Artificial Neural Networks (ANNs) for detecting roads from high-resolution satellite images. The methodology involves several key steps. Initially, the researchers collected RGB Ikonos and QuickBird images from Kish Island and Bushehr Harbour. These images were then enhanced using linear functions to improve their quality. A training set was created by selecting 500 road pixels and 500 background pixels. The researchers designed a Back-propagation Neural Network (BNN) with one hidden layer to process the images. They experimented with different input parameters, including spectral values, normalized distances to the road mean vector, and spatial information from neighboring pixels. The training process utilized an adaptive strategy to adjust the learning rate and momentum, ensuring optimal network performance. Accuracy was assessed using metrics such as the Road Detection Correctness Coefficient (RCC), Background Detection Correctness Coefficient (BCC), Root Mean Square Error (RMSE), and overall accuracy.

The data used in the study comprised RGB Ikonos and QuickBird images, with the training set consisting of 500 road and 500 background pixels. The input parameters tested included spectral values, normalized distances, and spatial information from neighboring pixels. The study areas were Kish Island and Bushehr Harbour in Iran, with the images from these locations serving as the basis for the analysis.

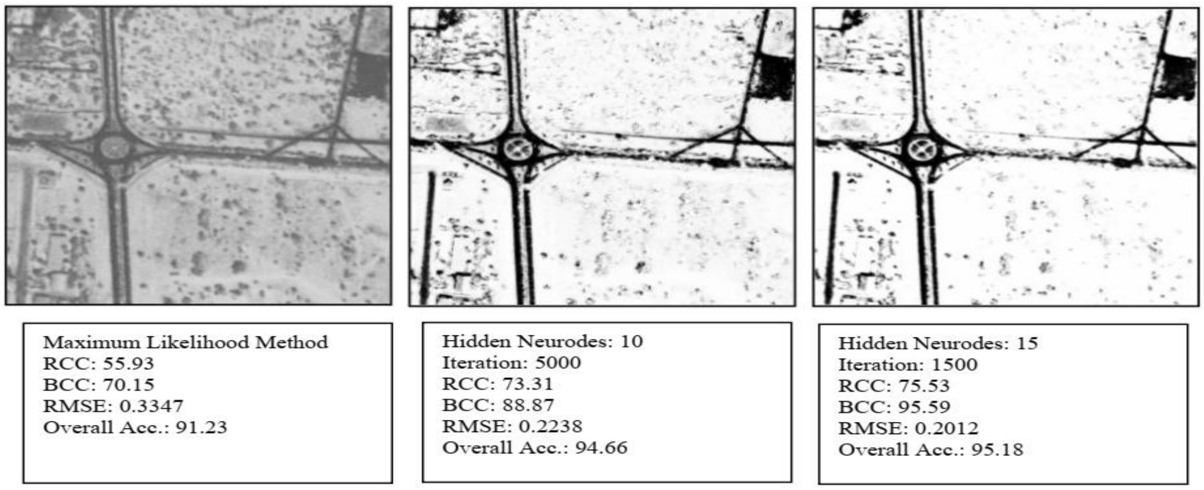


Figure 2.11: Left image: Maximum-Likelihood result, Middle image: Simple BNN with 10 neurods in hidden layer, Right image: Improved BNN with 15 neurods in hidden layer (Valadan & Mokhtarzade 2006).

The results of the study indicated varying levels of performance depending on the input parameters used. When spectral values alone were used as input, the network achieved moderate performance, with RCC values around 73-77% and overall accuracy around 94-95%. Including normalized distances improved the network's ability to detect background pixels, resulting in BCC values around 92-93%. Incorporating spatial information from neighboring pixels enhanced the network's road detection capability, with RCC values increasing to around 80-81%. The best performance was achieved by combining spectral values, normalized distances, and spatial information, resulting in RCC values around 75-76% and overall accuracy around 95%. Figure 2.11 represents the results from different algorithms with its metrics, showing better results for ANN algorithms, the results are better as the number of neurons increase.

Zhang et al. (2017) introduced a novel approach to road extraction from aerial imagery using a Deep Residual U-Net (ResUnet). This method combines the strengths of residual learning and the U-Net architecture to improve semantic segmentation performance. The architecture addresses challenges in training deep networks by integrating residual units, which mitigate issues like vanishing gradients and enable easier optimization. Unlike traditional U-Net, ResUnet removes the need for cropping operations, simplifying the architecture while maintaining accuracy. The network is divided into three components: an encoding path that compresses the input into feature representations, a bridge that connects encoding and decoding paths, and a decoding path that reconstructs feature representations into pixel-wise segmentations.

The study leverages the Massachusetts Roads Dataset, which consists of 1171 high-resolution images (1500 × 1500 pixels). These images capture diverse landscapes, including urban, suburban, and rural areas, along with a variety of structures such as roads, rivers, and buildings. The training dataset includes 1108 images, while 49 are reserved for testing. Preprocessing involves resizing images to 224 × 224 pixels for network input, and evaluation metrics include relaxed precision and recall, as well as the break-even point, which measures the intersection of precision and recall.

The results demonstrate that ResUnet outperforms existing methods, including U-Net and Mnih-CNN, with a break-even point of 0.9187 compared to 0.9053 for U-Net. The model excels in handling complex scenarios, such as occluded roads and distinguishing roads from similar structures like airfield runways. Additionally, ResUnet achieves these results with only 25% of the parameters used by U-Net, significantly reducing computational requirements.

A diagram of the proposed ResUnet architecture is illustrated in figure 2.12. To showcase qualitative results, example images can compare outputs from ResUnet and competing models against ground truth. Table 2.2 shows evaluation of the proposed and three other deep learning-based road extraction techniques on the Massachusetts roads dataset using breakeven point metrics. A higher breakeven point reflects superior performance in balancing precision and recall.

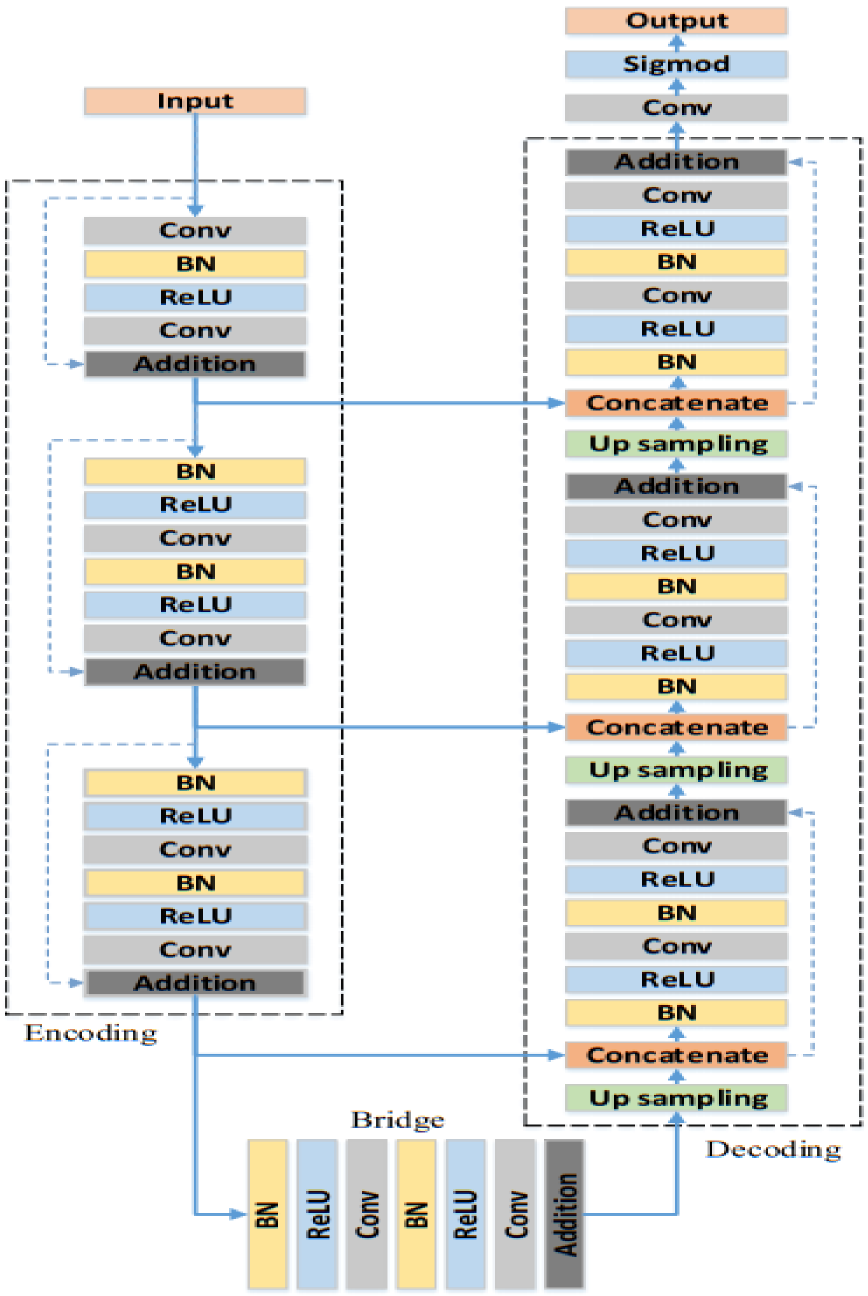


Figure 2.12 The architecture of the proposed deep ResUnet
(Zhang et al 2017)

Table 2.2: comparison of the proposed and three other techniques (Zhang et al 2017)

Model	Breakeven point
Mnih-CNN	0.8873
Mnih-CNN+CRF	0.8904
Mnih-CNN+Post-Processing	0.9006
Saito-CNN	0.9047
U-Net	0.9053
ResUnet	0.9187

Zhang. et al. (2019) introduced a novel method for road extraction using Sentinel-1 Synthetic Aperture Radar (SAR) images based on a Deep Fully Convolutional Neural Network (FCN). Figure 2.13 provides a flowchart of the proposed FCN-based road extraction method, illustrating the overall methodology from data pre-processing to performance evaluation. The methodology begins with the pre-processing of dual-polarimetric (VV and VH) Sentinel-1 SAR images, which includes radiation correction, speckle filtering using the Refined Lee filter, terrain correction, and decibel conversion. Roads were manually labeled using Google Maps and field surveys, and the labeled data was converted into raster format for training. The core of the method is the U-Net-based FCN, which was adapted for pixel-wise road extraction, Figure 2.14 illustrates how the FCN (U-Net) is used to extract roads automatically, showing the architecture and flow of the network. This architecture includes 20 convolutional layers with down-sampling and up-sampling steps, allowing for detailed feature extraction. For comparison, a CNN based on LeNet-5 was also implemented. The models were trained using TensorFlow with mini-batch Stochastic Gradient Descent (SGD) and GPU acceleration. Four optimization algorithms (SGD, Momentum, RMSProp, and Adam) were tested, with Adam emerging as the most effective. Huber loss was used for pixel-wise classification to balance the learning process. Performance was evaluated using Precision, Recall, and F1 score, with confidence intervals calculated for precision.

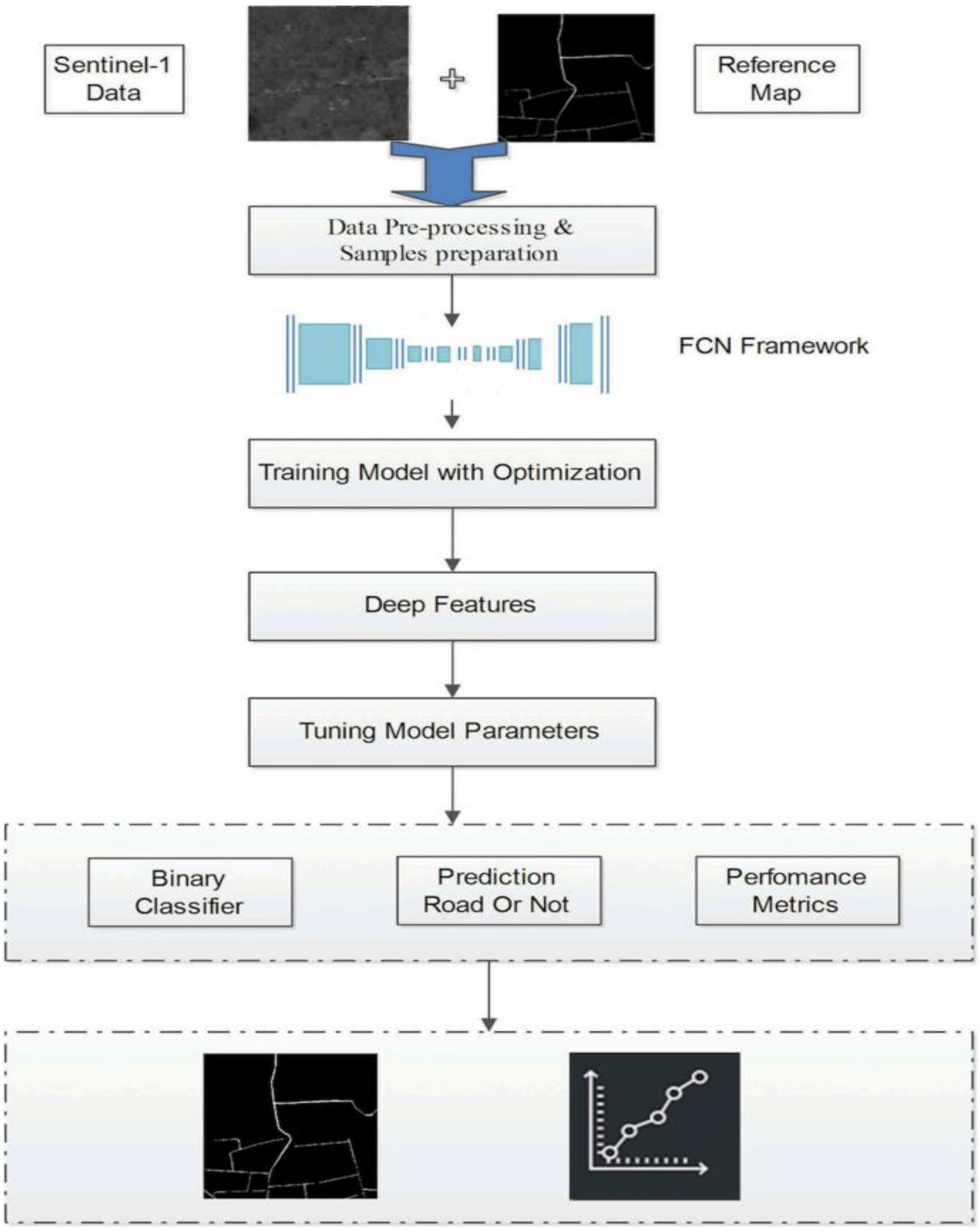


Figure 2.13 flowchart of the proposed FCN-based road extraction method (Zhang et al. 2019)

The study utilized dual-polarimetric (VV and VH) Sentinel-1 SAR images in Ground Range Detected (GRD) format, acquired in Interferometric Wide (IW) swath mode. The study area was a region in Beijing, China which includes both urban and rural areas with complex road networks. The dataset consisted of 1330 labeled tiles, each 256x256 pixels, with 300 tiles used for training and 200 for testing. Ground truth data was generated using Google Maps and field surveys, ensuring accurate labeling for model training and validation.

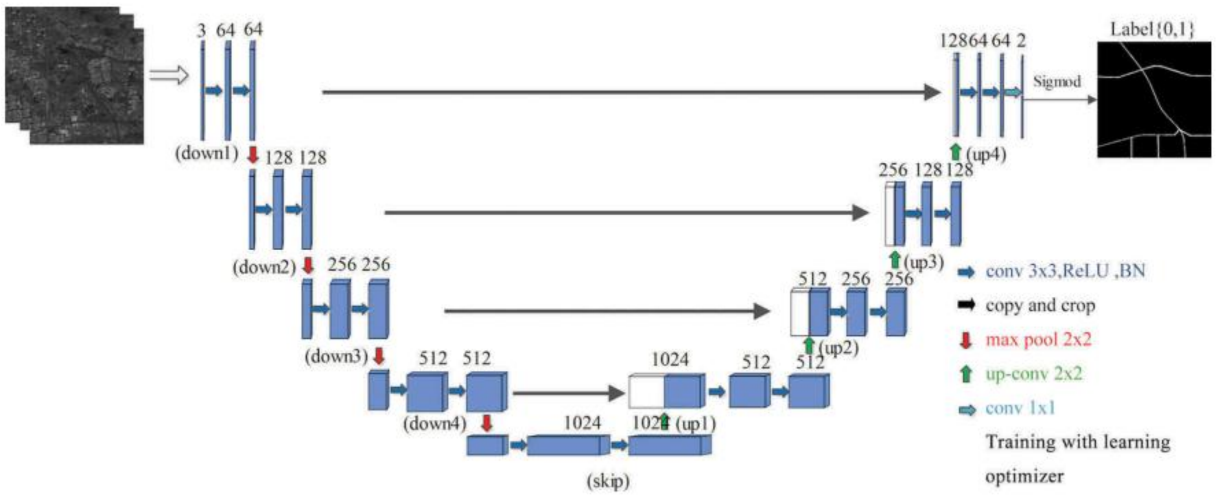


Figure 2.14: flowchart of the proposed FCN-based road extraction method (Zhang et al. 2019)

The results demonstrated the effectiveness of the proposed FCN-based method. In single-polarization tests, VV polarization outperformed VH polarization due to the higher energy in co-polarization. The FCN achieved an F1 score of 90.78% for VV polarization, compared to 88.73% for VH polarization. When using dual-polarization (VV/VH), the FCN achieved even better results, with an F1 score of 93.63%. The FCN also outperformed the CNN in both accuracy and computational efficiency, reducing training time by 34.4% for a full dataset. Additionally, the FCN surpassed traditional machine learning methods (SVM, KNN, DT, RF) in accuracy, achieving an F1 score of 93.63% compared to 89.23% for the best machine learning method (Random Forest). These results highlight the robustness of the FCN-based method for road extraction in complex environments.

Mahara. Et al. (2025) presented an advanced approach for automated road extraction from high-resolution satellite imagery using a modified DeepLabV3+ model. The authors proposed a novel Dense Depthwise Dilated Separable Spatial Pyramid Pooling (DenseDDSSPP) module, which replaces the conventional Atrous Spatial Pyramid Pooling (ASPP) module in DeepLabV3+. The DenseDDSSPP module is designed to enhance the extraction of complex road structures by leveraging depthwise dilated separable convolutions, which improve computational efficiency and feature extraction capabilities. The model also integrates a Squeeze-and-Excitation (SE) block in the decoder to focus on relevant feature channels, further improving road extraction accuracy.

The study uses two publicly available datasets:

1. **Massachusetts Road Dataset**: Contains 1171 satellite images with corresponding road masks, each of size 1500x1500 pixels. The dataset was cropped into 512x512 tiles for training and testing.
2. **DeepGlobe Road Dataset**: Consists of 6226 images with 1024x1024 resolution, also cropped into 512x512 tiles. A subset of 9000 images was used due to computational constraints.

The proposed model integrates the following key components:

1. **DenseDDSSPP Module**: Replaces the ASPP module in DeepLabV3+ and uses depthwise dilated separable convolutions to generate dense feature maps with multi-scale contextual information.
2. **Squeeze-and-Excitation Block**: Enhances the decoder by focusing on relevant feature channels, improving the model's ability to handle occlusions and shadows.
3. **Xception Backbone**: Selected as the backbone network for feature extraction due to its efficiency and performance.

The model was evaluated using metrics such as Intersection over Union (IoU), Precision, and F1 Score. The results, as shown in Table 2.3 and Table 2.4, demonstrate that the proposed model outperforms several state-of-the-art models, including U-Net, DeepLabV3+ with ASPP, and RFE-LinkNet, on both datasets. For example, on the Massachusetts dataset, the proposed model achieved an IoU of 67.21% and an F1 Score of 79.29%, while on the DeepGlobe dataset, it achieved an IoU of 71.61% and an F1 Score of 81.75%.

Table 2.3 Quantitative observation of results obtained from all models in Massachusetts road dataset.

Model	IOU (%)	Precision (%)	F1 Score (%)
U-Net	64.19	80.23	74.78
Original DeepLabV3+	65.92	80.04	75.6
SegNet	58.67	78.56	73.73
DCSFEP	62.48		76.59
D-LinkNet	63.74	75.89	77.86
RFE-LinkNet	66.77	80.88	80.07
Proposed Model	67.21	81.38	79.29

Table 2.4 Quantitative observation of results obtained from all models in DeepGlobe road dataset.

Model	IOU (%)	Precision (%)	F1 Score (%)
U-Net	62.82	80.36	71.83
Original DeepLabV3+	69.05	83.16	77.96
SegNet	57.66	80.67	72.09
Attention-Unet	67.42	79.95	80.54
D-LinkNet	63.94	78.54	76.59
RFE-LinkNet	70.72	83.09	**82.85**
Proposed Model	**71.61**	**83.19**	81.75

The authors highlight the computational efficiency of the proposed model, which reduces the number of parameters and FLOPs compared to the original DeepLabV3+ with ASPP. The DenseDDSSPP module is shown to be approximately 62% more efficient than standard convolution in terms of operations, as illustrated in the mathematical comparison provided in the paper. Visual comparisons in figure 2.15 and figure 2.16 further demonstrate the model's ability to accurately extract and connect road segments, even in challenging scenarios such as occlusions by trees.

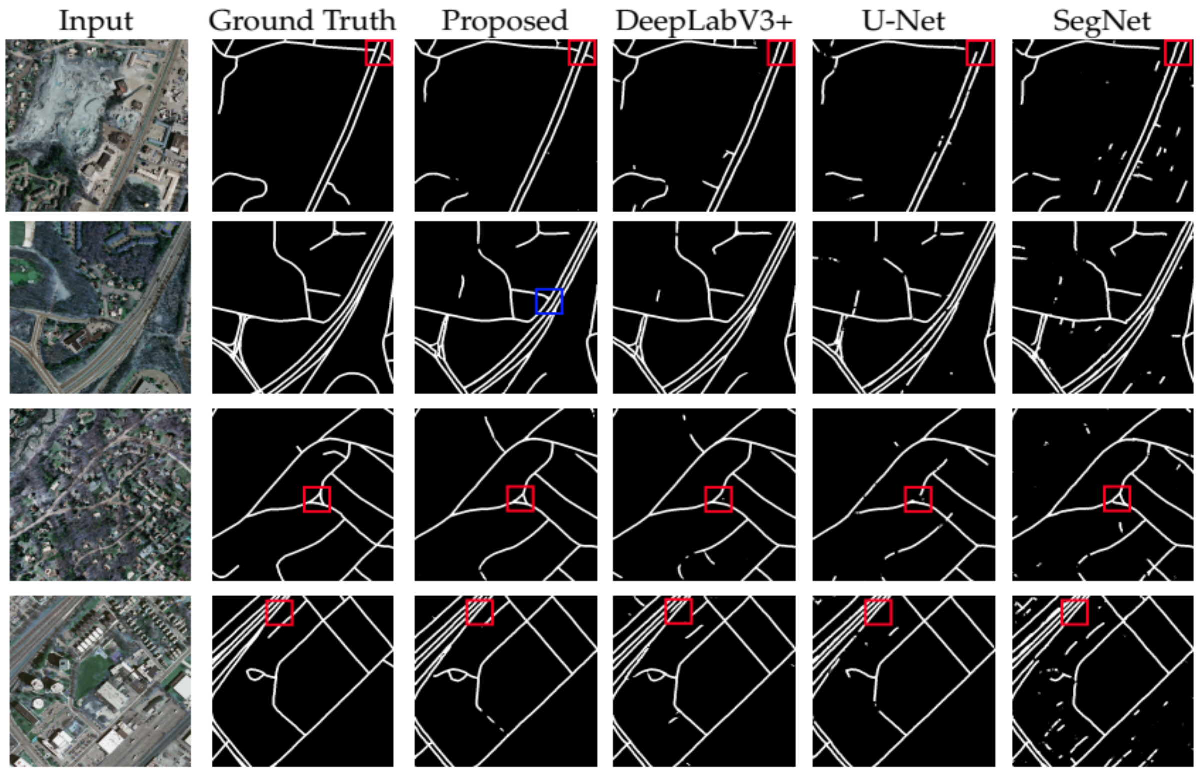


Figure 2.15 Comparative results of road extraction from the Massachusetts road dataset

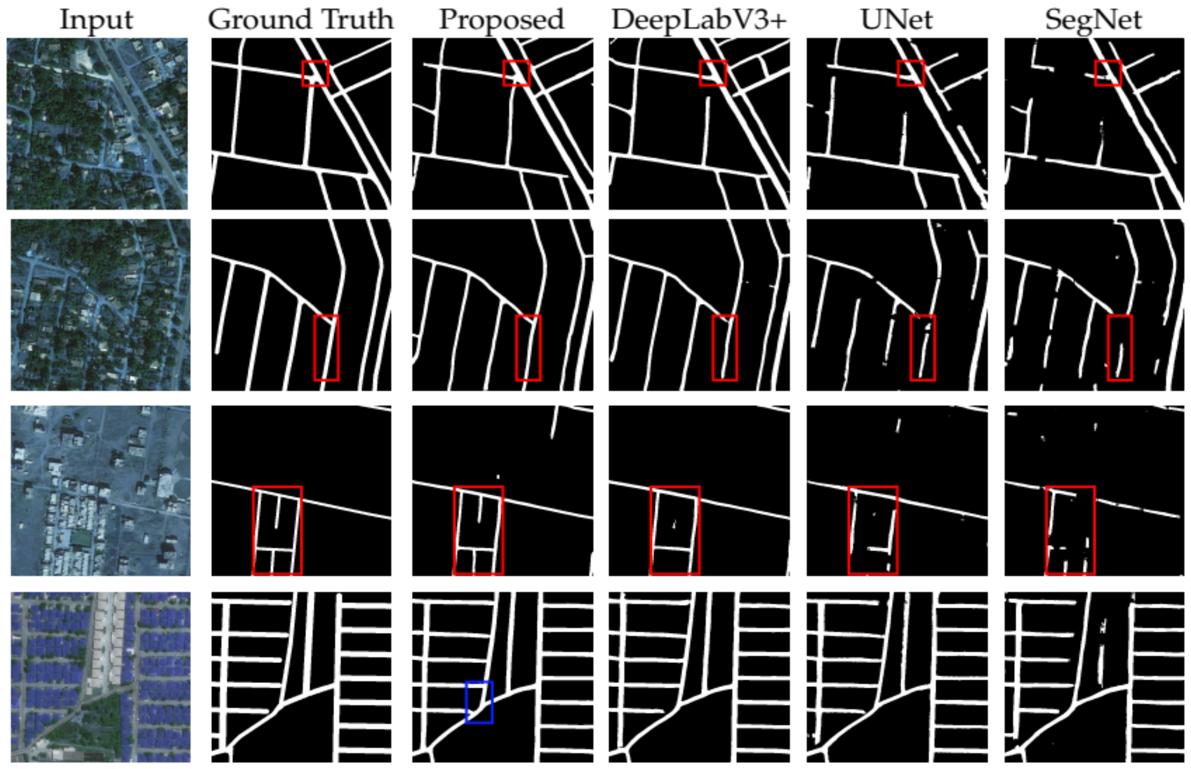


Figure 2.16 Comparative results of road extraction from the DeepGlobe road dataset

2.5 Gaps in Deep Learning Approaches for Road Detection from Satellite Images

Deep learning has emerged as a promising approach, leveraging convolutional neural networks (CNNs) and related architectures to automatically extract road features. However, despite significant advancements, several gaps remain that limit the accuracy and practicality of these methods. These gaps are particularly evident in edge detection, road connectivity, and the ability to distinguish roads from similar features, each of which is explored in detail below.

2.5.1 **Inaccurate Edge Detection**: Current deep learning models, such as U-net or DeepLab, often rely on standard segmentation architectures that may not optimally capture the fine details required for precise edge detection. This is due to the downsampling and upsampling operations in these models, which can lead to a loss of spatial information at the boundaries.
The impact can be seen in the consequence is blurred or imprecise road boundaries, which are unsatisfactory for applications needing accurate road outlines, such as navigation systems and urban planning. For instance, imprecise edges can lead to errors in route planning or infrastructure mapping, affecting the reliability of the detected road network. Mohd and Pankaj (2023) provided a study on advanced road extraction using a CNN-based U-net model (Advanced road extraction using CNN-based U-net model and satellite imagery) highlighted that many techniques struggle to maintain proper edges and boundaries, particularly in the presence of complex backgrounds.

2.5.2 **Lack of Road Connectivity**: Most deep learning-based road detection methods treat the problem as a standard semantic segmentation task, where each pixel is classified independently. This approach does not inherently ensure that the detected road segments are connected, leading to fragmented road networks.
The Impact is disconnected road segments which are problematic for applications that rely on continuous road networks, such as route planning, traffic management, and map updating. For example, a fragmented road network can disrupt navigation algorithms, making them less effective in real-world scenarios.
Cheng. et.al (2019) proposed a structured deep neural network to obtain smooth and continuous road skeletons, indicating that previous methods lacked this focus. Additionally, Emrah and

Yıldırım (2025) in their work “Detection of road extraction from satellite images with deep learning method” achieved high accuracy but did not explicitly address connectivity, suggesting a need for improvement in this area.

2.5.3 **Difficulty in Distinguishing Roads from Similar Features**: Satellite images often contain a variety of linear features that can be mistaken for roads, such as rivers, railways, and building shadows. This visual similarity poses a significant challenge for accurate classification, especially in complex urban or natural environments.

Misclassifications can degrade the accuracy of road detection, leading to false positives and reduced reliability of the detected road network. This is particularly critical in areas with dense infrastructure or varied landscapes, where distinguishing roads from other features is more challenging.

This gap is highlighted by Mohd and Pankaj (2023) Advanced road extraction using CNN-based U-net model and satellite imagery, which noted the presence of complex background elements like shadows of buildings and trees as a challenge. Similarly, A review of road extraction from remote sensing images emphasized the difficulty of using only one kind of road feature, suggesting the need for methods that can handle multiple feature types.

2.5.4 Additional Considerations

Beyond the three primary gaps, there are other areas worth noting

- **Generalization Across Different Geographies and Resolutions:** Many deep learning models perform well on the specific dataset they are trained on but may not generalize effectively to new, unseen data, such as images from different geographical areas or with varying resolutions. This is evident in “Predicting road quality using high resolution satellite imagery: A transfer learning approach”, which used transfer learning to adapt models across regions, suggesting a gap in generalization that your paper could address through domain adaptation or robust model design.
- **Dependence on Labeled Data:** The reliance on large amounts of labeled data is a common challenge in deep learning for remote sensing. While some studies, like “Recognizing road from satellite images by structured neural network”, collected large datasets, the general scarcity of labeled data, especially for specific regions, remains a gap.

2.6 Summary

This chapter has provided a comprehensive foundation for understanding the integration of deep learning with satellite imagery for road detection, a critical task for urban planning, environmental sustainability, and smart cities. Section 2.2 introduced neural networks and deep learning, detailing architectures like CNNs, which are central to this study due to their ability to extract spatial features from imagery. Section 2.3 explored deep learning's transformative applications in remote sensing, including land cover classification, object detection, and environmental monitoring, showcasing its adaptability to diverse geospatial challenges. Section 2.4 reviewed prior methods for road extraction, from Baumgartner et al.'s (1999) multi-resolution approach to Mahara et al.'s (2025) advanced DeepLabV3+ modification, demonstrating significant progress in accuracy and efficiency.

However, Section 3 identified persistent gaps in these approaches that limit their practicality. Inaccurate edge detection, due to spatial information loss in models like U-Net, results in blurred boundaries, undermining applications requiring precision (Mohd & Pankaj, 2023). The lack of road connectivity, stemming from pixel-independent segmentation, produces fragmented networks, challenging navigation systems (Cheng et al., 2019; Emrah & Yıldırım, 2025). Difficulty distinguishing roads from similar features like rivers or shadows leads to misclassifications, especially in complex environments. Additional concerns include poor generalization across geographies and resolutions and heavy reliance on labeled data. These gaps—in edge precision, connectivity, feature differentiation, adaptability, and data efficiency—present opportunities for innovation. By addressing these challenges, this research aims to enhance the robustness and applicability of deep learning for road detection, advancing the field toward more reliable geospatial solutions.

CHAPTER 3

CONVOLUTIONAL NEURAL NETWORKS AND ADVANCED ARCHITECTURES FOR ROAD EXTRACTION

CHAPTER 3

CONVOLUTIONAL NEURAL NETWORKS AND ADVANCED ARCHITECTURES FOR ROAD EXTRACTION

The previous literature shows the new direction of object detection which is leveraging the use of neural networks especially the Convolutional Neural Networks (CNNs) in the operations whether feature extraction or feature detection or even any other Computer Vision (CV) task. According to Liu et al. (2020), CNNs have revolutionized generic object detection by enabling high-accuracy segmentation and efficient feature extraction through their hierarchical architectures. Agarwal et al. (2018) reviewed advancements in object detection, emphasizing CNNs' ability to process raw image pixels into meaningful features, leading to improved detection accuracy.

3.1 Convolutional Neural Networks (CNNs)

3.1.1 Introduction to CNNs

Convolutional Neural Networks (CNNs) are a specialized class of deep learning models designed to process grid-like data, such as images, by automatically learning spatial hierarchies of features (LeCun et al., 1998). Inspired by the visual cortex, CNNs use convolutional layers to extract local patterns like edges, textures, and shapes, making them highly effective for tasks such as image classification, object detection, and segmentation (Goodfellow et al., 2016). Unlike traditional neural networks, CNNs leverage parameter sharing and pooling layers to reduce computational complexity and improve translation invariance. Their ability to learn relevant features directly from raw data has revolutionized fields like computer vision, medical imaging, and video analysis. Since their introduction, CNNs have evolved significantly, with architectures like AlexNet, ResNet, and EfficientNet pushing the boundaries of performance and scalability. Today, CNNs are a cornerstone of modern artificial intelligence, enabling breakthroughs in both research and industry.

3.1.2 Core Components of CNNs

The core components of CNNs are the building blocks that enable them to effectively process and analyze grid-like data, such as images. The first key component is the convolutional layer, which applies a set of learnable filters (kernels) to the input data to extract local features like edges, textures, and patterns (LeCun et al., 1998). Each filter slides (convolves) over the input, performing element-wise multiplication and summation to produce a feature map, highlighting specific characteristics of the data. Figure 3.1 gives an example of the convolution operation using a 3*3 filter.

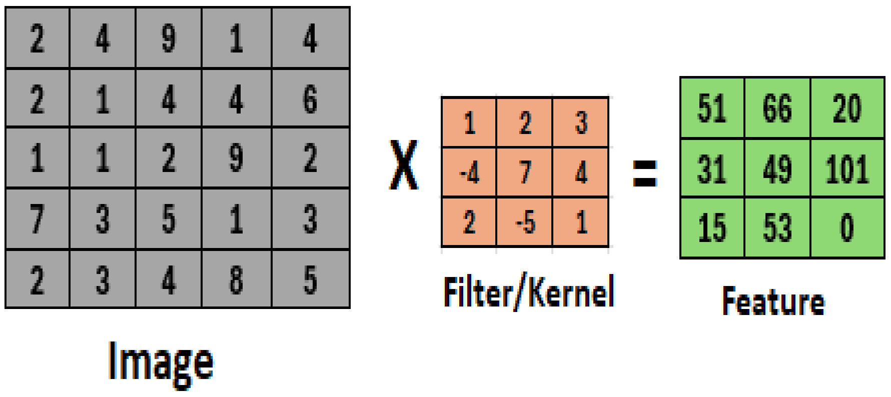
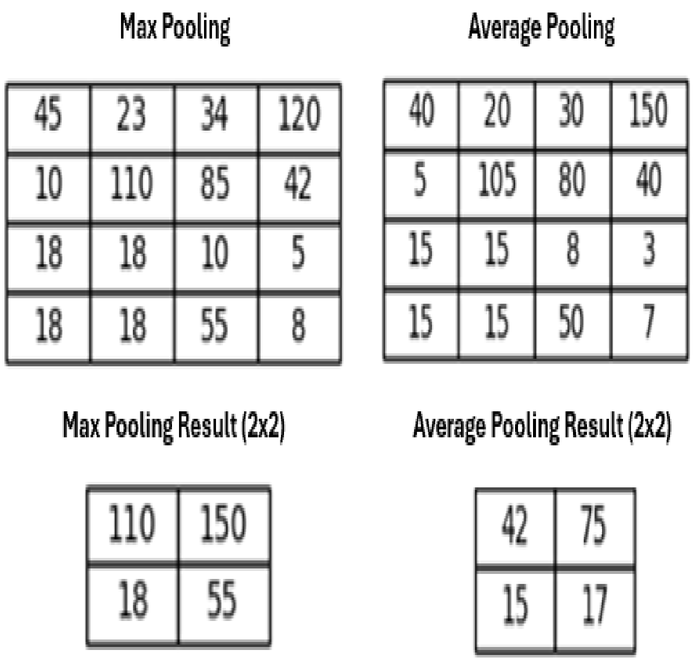


Figure 3.1: Convolving an image using 3×3 filter

Parameters such as kernel size, stride, and padding control the spatial dimensions and granularity of the extracted features. The second component is the pooling layer, which downsamples the feature maps to reduce computational complexity and prevent overfitting. Common pooling operations include max pooling and average pooling, as shown in figure 3.2, which retain the most salient information while reducing spatial dimensions (Goodfellow et al., 2016).

The third component is the activation function, which introduces non-linearity into the network, enabling it to learn complex patterns. The Rectified Linear Unit (ReLU) is the most widely used activation function due to its simplicity and effectiveness in mitigating the vanishing gradient problem.



Max Pooling

45	23	34	120
10	110	85	42
18	18	10	5
18	18	55	8

Average Pooling

40	20	30	150
5	105	80	40
15	15	8	3
15	15	50	7

Max Pooling Result (2x2)

110	150
18	55

Average Pooling Result (2x2)

42	75
15	17

Figure 3.2: Max and Average pooling

The fourth component is the fully connected (FC) layer as shown in figure 3.3, which combines the high-level features extracted by the convolutional and pooling layers to produce the final output, such as class probabilities in classification tasks. Additionally, dropout layers are often used to regularize the network by randomly deactivating neurons during training, reducing the risk of overfitting. Finally, batch normalization is frequently employed to stabilize and accelerate training by normalizing the inputs of each layer. These components form the foundation of CNNs, enabling them to achieve state-of-the-art performance.

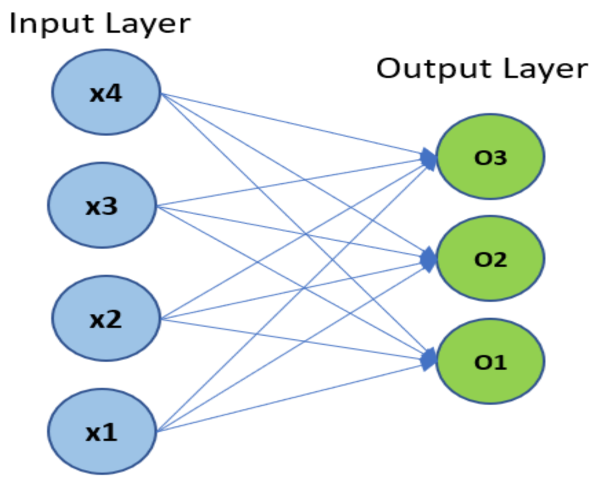


Figure 3.3: Fully Connected Network

To better visualize the components of the CNN, figure 3.2 illustrates the flow of data through a CNN, showing the input image, convolutional layers with feature maps, pooling layers, fully connected layers, and the final output.

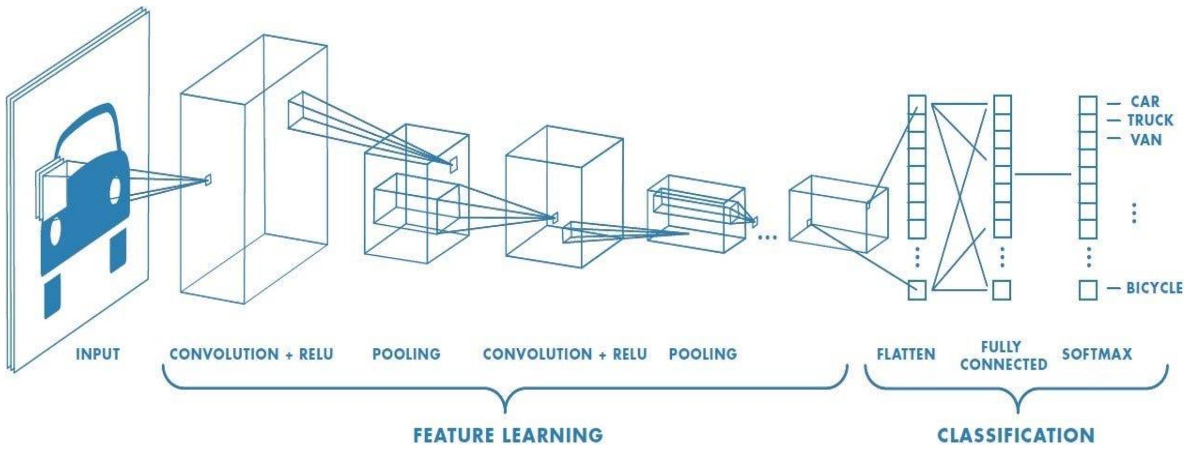


Figure 3.4: A visual representation of a CNN architecture, showing the input image, convolutional layers, pooling layers, fully connected layers, and the final output.

3.1.3 CNN Architecture

- Input Layer:
    - The input layer receives raw data, typically in the form of images, which are represented as 3D tensors (height × width × channels). For example, an RGB image has three channels (red, green, blue), while grayscale images have one channel.
    - The input is normalized (e.g., pixel values scaled to [0, 1] or [-1, 1]) to improve training stability.

- Convolutional Layers:
    - Filters (Kernels): Convolutional layers apply a set of learnable filters to the input. Each filter is a small matrix (e.g., 3×3 or 5×5) that slides over the input, performing element-wise multiplication and summation to produce a feature map.

  - Feature Maps: Each filter extracts specific features, such as edges, textures, or patterns. Multiple filters are used to capture diverse features, resulting in multiple feature maps.
  - Stride: Controls the step size of the filter as it moves across the input. A stride of 1 moves the filter one pixel at a time, while a stride of 2 skips every other pixel.
  - Padding: Adds extra pixels around the input to control the spatial dimensions of the output. "Same" padding preserves the input size, while "valid" padding reduces it.

- **Activation Functions**
  - After each convolutional operation, an activation function is applied to introduce non-linearity. The most common activation function is the Rectified Linear Unit (ReLU), defined as ReLU(x)=max(0,x).
  - ReLU helps mitigate the vanishing gradient problem and accelerates convergence. Variants like Leaky ReLU and Parametric ReLU address the "dying ReLU" problem by allowing small negative values.

- **Pooling Layers**
  - Pooling layers downsample the feature maps, reducing their spatial dimensions while retaining the most important information.
  - Max Pooling: Selects the maximum value in each pooling window (e.g., 2×2), preserving the most salient features.
  - Average Pooling: Computes the average value in each pooling window, providing smoother downsampling.
  - Pooling reduces computational complexity, controls overfitting, and improves translation invariance.

- **Fully Connected Layers**
  - After several convolutional and pooling layers, the high-level features are flattened into a 1D vector and passed to fully connected (FC) layers.
  - FC layers combine the features to produce the final output, such as class probabilities in classification tasks.

  - Each neuron in an FC layer is connected to every neuron in the previous layer, making it computationally expensive for large inputs.

- **Output Layer**
  - The output layer produces the final predictions. For classification tasks, a softmax activation function is often used to convert raw scores into probabilities.
  - For regression tasks, a linear activation function is used to produce continuous values.

3.1.4 Training CNNs

3.1.4.1 The problem of overfitting

One of the most common challenges faced in training deep learning models is the issue of high bias or high variance, which directly impacts the model's ability to generalize to new, unseen data. High bias, often referred to as underfitting, occurs when the model is too simplistic and fails to capture the underlying patterns in the training data. In mathematical terms, this can be understood as the polynomial function mapping the relationship between the input (x) and output (y) being of a lower order, which limits its ability to represent complex relationships. As a result, the model performs poorly not only on the training data but also on new data, as it lacks the necessary complexity to make accurate predictions.

On the other hand, high variance, commonly known as overfitting, arises when the model is excessively complex and memorizes the training data, including its noise and outliers. In this case, the polynomial function mapping the input-output relationship is of a higher order, allowing the model to fit the training data almost perfectly. However, this excessive complexity hinders the model's ability to generalize to new data, leading to poor performance on unseen datasets. Essentially, the model becomes too specialized to the training data, losing its predictive power for real-world applications.

Figure 3.5 provides a visual representation of the bias-variance tradeoff, a fundamental concept in machine learning that highlights the relationship between model complexity and error. The x-axis of the graph represents model complexity, ranging from simple models on the left to highly complex models on the right. The y-axis represents error, which includes both training error (blue line) and validation error (red line).

On the right side of the graph, where model complexity is high, the model exhibits high variance (overfitting). Here, the training error is very low, indicating that the model fits the training data exceptionally well. However, the validation error is significantly higher, demonstrating that the model fails to generalize to new data. This region is highlighted by the purple shaded area, where the model captures not only the underlying patterns but also the noise and irregularities in the training data, resulting in poor performance on unseen datasets.

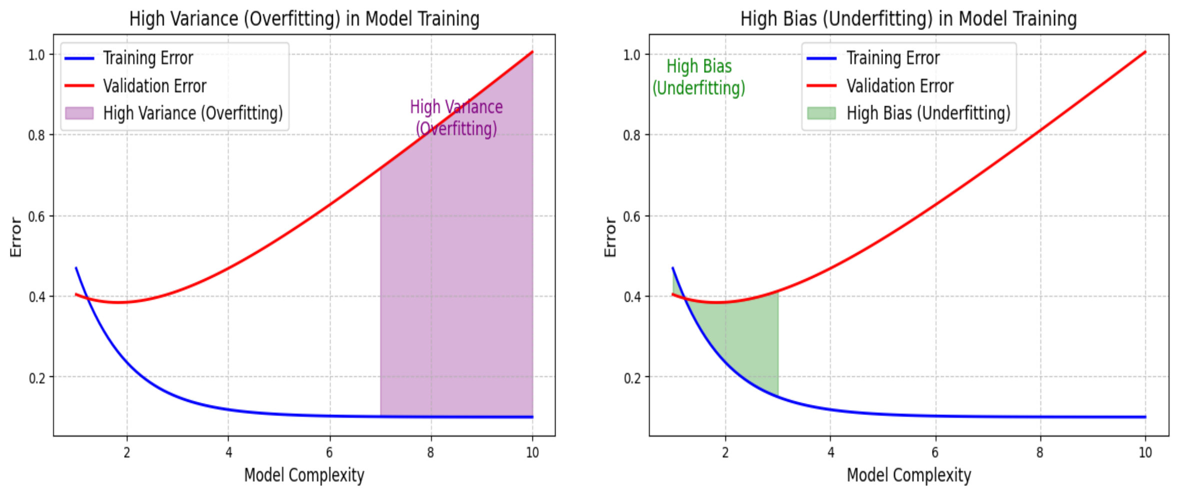


Figure 3.5: High variance and high bias

Conversely, on the left side of the graph, where model complexity is low, the model suffers from high bias (underfitting). In this scenario, both the training error and validation error are high, indicating that the model is too simplistic to capture the essential patterns in the data. This region is marked by the green shaded area, where the model lacks the necessary complexity to learn the data's structure, leading to suboptimal performance on both the training and validation datasets.

The optimal point lies in the middle of the graph, where the validation error is minimized. At this point, the model achieves a balance between bias and variance, performing well on both the training and validation datasets. This balance ensures that the model is complex enough to capture the underlying patterns in the data without overfitting to the noise, thereby maximizing its generalization capability.

Understanding and addressing the bias-variance tradeoff is crucial for developing robust and reliable deep learning models. Techniques such as regularization, cross-validation, and hyperparameter tuning can help mitigate these issues, ensuring that the model achieves optimal performance on both seen and unseen data. By carefully managing model complexity, practitioners can build models that not only perform well during training but also generalize effectively to real-world applications.

3.1.4.2 Training phases

Training Convolutional Neural Networks (CNNs) is a complex process that involves optimizing the network's parameters to minimize a loss function. Below is a detailed explanation of the training process, including key concepts, equations, and visualizations where applicable. First of all, forward propagation, during forward propagation, input data passes through the network's layers to produce an output. For a convolutional layer, the operation can be expressed as in equation (3.1) (Goodfellow et al., 2016):

$$z_{i,j,k} = \sum_{m} \sum_{p} \sum_{q} x_{i+p-1,j+q-1,m} \cdot w_{p,q,m,k} + b_k \qquad (3.1)$$

Where x is the input feature map, *w* is the filter (kernel), b is the bias., and z is the output feature map.

The output is then passed through an activation function, such as ReLU.

The loss function here plays a vital role in the propagation function to update the weight of the NN. Different loss functions can be used according to the application of the NN.

For classification tasks, cross-entropy loss is commonly used as in equation (3.2) (Bishop, 2006):

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}) \qquad (3.2)$$

Where *N* is the number of samples, *C* is the number of classes, and $y_{i,c}$ is the true label (one-hot encoded). While For regression tasks as the two proposed models in this study, mean squared error (MSE) is used as in equation (3.3) (Hastie et al., 2009):

$$L = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \qquad (3.3)$$

After the forward propagation, the loss error is calculated and the backpropagations step starts, Backpropagation computes the gradients of the loss function with respect to the network's parameters. The gradient for a weight in a convolutional layer is calculated as in equation (3.4) (Goodfellow et al., 2016):

$$\frac{\partial L}{\partial w_{p,q,m,k}} = \sum_i \sum_j \frac{\partial L}{\partial z_{i,j,k}} \cdot x_{i+p-1,j+q-1,m} \qquad (3.4)$$

Equation (3.5), which is derived from equation 3.4 shows that the gradient for the bias is:

$$\frac{\partial L}{\partial b_k} = \sum_i \sum_j \frac{\partial L}{\partial z_{i,j,k}} \qquad (3.5)$$

These gradients are used to update the weights and biases using an optimization algorithm.

3.1.4.3 Optimization Algorithms

Optimization algorithms are critical for training CNNs as they determine how the model's parameters are updated to minimize the loss function. The choice of optimization algorithm can significantly impact the speed of convergence, the quality of the final model, and the stability of the training process. The most common optimization algorithms used with CNN are:

1. Stochastic Gradient Descent (SGD)

SGD is one of the most fundamental optimization algorithms used in training neural networks. It works by iteratively updating the model's parameters in the direction of the negative gradient of the loss function. The update rule for SGD is given by equation (3.6) (Bottou, 2010):

$$w_{t+1} = w_t - \eta \nabla L(w_t) \qquad (3.6)$$

where w_t represents the model's parameters at time step t, η is the learning rate, and $\nabla L(w_t)$ is the gradient of the loss function with respect to the parameters. The learning rate controls the size of the steps taken during optimization. A small learning rate can lead to slow convergence, while a large learning rate can cause the training process to diverge.

One of the main advantages of SGD is its simplicity and efficiency, especially when dealing with large datasets. However, SGD can suffer from oscillations and slow convergence, particularly when the loss surface is highly non-convex. To address these issues, variants of SGD, such as SGD with momentum and Nesterov Accelerated Gradient (NAG), have been developed.

2. SGD with Momentum

SGD with momentum introduces a momentum term as in equation (3.7) (Qian, 1999) to smooth the update process and accelerate convergence. The update rule for SGD with momentum is:

$$v_t+1 = \gamma\, v_t + \eta \nabla L(w_t) \quad (3.7)$$

$$w_{t+1} = w_t - v_{t+1}$$

where v_t is the velocity vector at time step t, and γ is the momentum coefficient, typically set to a value between 0.5 and 0.9. The momentum term helps to accumulate the gradient over time, allowing the optimizer to build up speed in directions with consistent gradients and dampen oscillations in directions with fluctuating gradients.

3. Nesterov Accelerated Gradient (NAG)

NAG is a variant of SGD with momentum that improves convergence by taking into account the future position of the parameters. The update rule for NAG as shown in equation (3.8) (Nesterov, 1983) is:

$$v_t+1 = \gamma v_t + \eta \nabla L(w_t - \gamma v_t)$$

$$w_{t+1} = w_t - v_{t+1} \qquad (3.8)$$

By evaluating the gradient at the estimated future position $w_t - \gamma v_t$, NAG can make more informed updates, leading to faster convergence and better performance, especially in the presence of high curvature in the loss surface.

3.1.4.4 Regularization Techniques

Regularization techniques are essential for preventing overfitting, which occurs when a model learns to perform well on the training data but fails to generalize to unseen data. Regularization methods introduce additional constraints or penalties to the model's training process to encourage simpler and more generalizable solutions. There are many regularization techniques that can be applied.

1. Dropout

Dropout is a regularization technique that randomly deactivates a fraction of neurons during training. At each training step, each neuron has a probability p*p* of being temporarily "dropped out," meaning its output is set to zero. This prevents the network from becoming overly reliant on specific neurons and encourages the development of redundant representations. The dropout rate p*p* is typically set between 0.2 and 0.5. During inference, all neurons are active, but their outputs are scaled by $1-p$ to account for the dropout during training.

Dropout effectively acts as a form of ensemble learning, where multiple sub-networks are trained simultaneously, and their predictions are averaged during inference. This leads to improved generalization and robustness.

2. Batch Normalization

Batch Normalization (BN) is a technique that normalizes the inputs of each layer to have zero mean and unit variance. This is done by computing the mean and variance of the inputs over the current mini-batch and applying the following transformation, as shown in equations (3.9) (Ioffe & Szegedy,2015):

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad (3.9)$$

$$y_i = \gamma \hat{x}_i + \beta$$

where μB and $\sigma_B{}^2$ are the mean and variance of the mini-batch, ϵ is a small constant to prevent division by zero, and γ and β are learnable parameters that allow the network to scale and shift the normalized values. Batch Normalization stabilizes and accelerates training by reducing internal covariate shift, which refers to the change in the distribution of layer inputs during training. It also acts as a regularizer by adding noise to the network's activations.

3. Data Augmentation

Data Augmentation is a technique that increases the diversity of the training data by applying random transformations to the input data. For image data, common transformations include rotation, scaling, flipping, cropping, and color jittering. By exposing the model to a wider variety of data during training, data augmentation helps the model learn more robust and generalizable features. This is particularly useful when the training dataset is small, as it effectively increases the amount of training data without requiring additional labeled examples.

4. Weight Decay (L2 Regularization)

Weight Decay, also known as L2 regularization, adds a penalty term to the loss function that discourages large weights. The modified loss function is given by equation (3.10) (Krogh & Hertz, 1992):

$$L_{\text{reg}} = L + \frac{\lambda}{2} \sum_{i} w_i^2 \qquad (3.10)$$

where L is the original loss function, λ is the regularization strength, and w_i are the model's weights. The penalty term encourages the model to learn smaller weights, which typically leads to simpler and more generalizable solutions. Weight decay is particularly effective in preventing overfitting in models with a large number of parameters.

5. Early Stopping

Early Stopping is a simple yet effective regularization technique that monitors the model's performance on a validation set during training. If the validation loss does not improve for a specified number of epochs (known as the patience), training is stopped early. This prevents the model from overfitting to the training data by halting the training process before it begins to memorize the training set. Early stopping is often used in conjunction with other regularization techniques to further improve generalization.

3.2 Faster R-CNN

Faster R-CNN (Region-based Convolutional Neural Network) is a state-of-the-art object detection algorithm proposed by Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun in their seminal 2015 paper, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks" (Ren et al., 2015) . It builds upon the earlier R-CNN (Girshick et al., 2014) and Fast R-CNN (Girshick, 2015) frameworks, addressing their computational inefficiencies and limitations in generating region proposals. Faster R-CNN introduces a novel Region

Proposal Network (RPN), which shares convolutional features with the detection network, enabling end-to-end training and significantly improving both speed and accuracy.

3.2.1 Architecture Overview

The Faster R-CNN architecture consists of two main components:

- **Region Proposal Network (RPN):** Generates region proposals (candidate object bounding boxes) directly from the feature maps.

- **Fast R-CNN Detection Network:** Classifies the proposed regions and refines their bounding box coordinates.

These components share a common set of convolutional layers, making the system computationally efficient and capable of real-time performance. As shown in Figure 3.6, Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network, generating proposals that are then processed by the detection network.

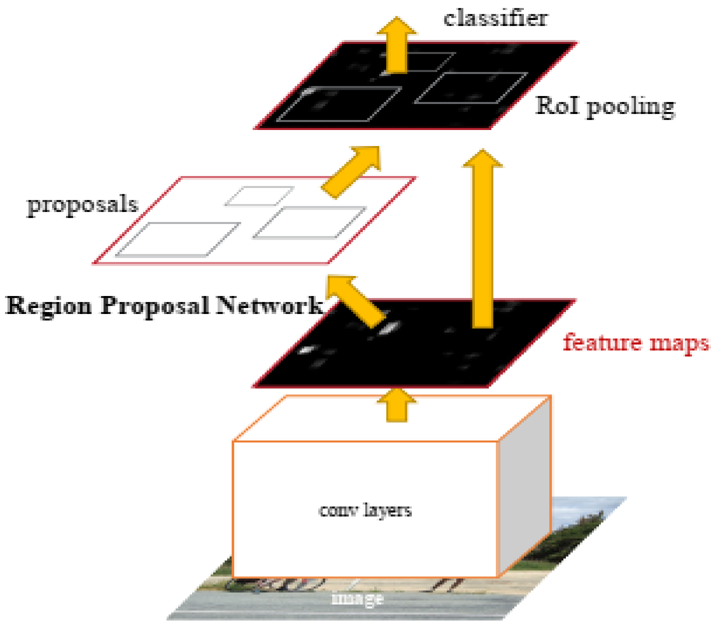


Figure 3.6: Faster R-CNN operates as a unified framework for object detection, where the Region Proposal Network (RPN) acts as the system's attention mechanism. (Ren et al., 2015).

3.2.2 Region Proposal Network (RPN)

The RPN is the core innovation of Faster R-CNN. It replaces the traditional region proposal methods (e.g., Selective Search used in R-CNN and Fast R-CNN) with a fully convolutional network that predicts object bounds and objectness scores at each spatial location.

The RPN uses a set of predefined anchor boxes of varying scales and aspect ratios at each sliding window location. These anchors serve as reference boxes for predicting region proposals. For each anchor, the RPN predicts two outputs: (1) a binary classification score (object vs. background) and (2) bounding box regression offsets to refine the anchor's position.

The RPN is trained using a multi-task loss function that combines classification loss (log loss) and regression loss (smooth L1 loss) for the bounding box coordinates (Ren et al., 2015). The loss function is given by:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \quad (3.11)$$

where L_{cls} is the classification loss, L_{reg} is the regression loss, p_i is the predicted probability of the anchor being an object, p_i^* is the ground-truth label (1 for object, 0 for background), t_i is the predicted bounding box coordinates, and t_i^* is the ground-truth bounding box coordinates.

The RPN generates high-quality region proposals efficiently, sharing convolutional features with the detection network and eliminating the need for external proposal methods. Figure 3.7 illustrates the architecture of the RPN, showing how it generates proposals from feature maps.

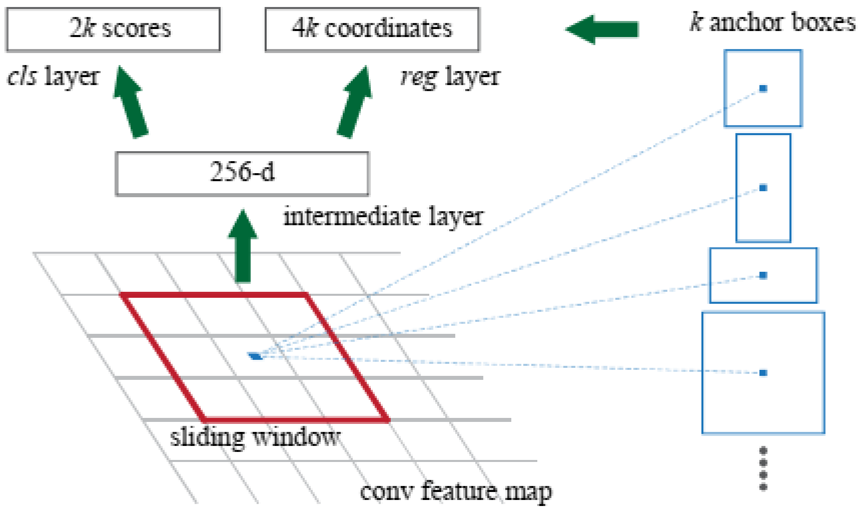


Figure 3.7: Region Proposal Network (RPN). (Ren et al., 2015).

3.2.3 Fast R-CNN Detection Network

Once the RPN generates region proposals, they are fed into the Fast R-CNN detection network for classification and bounding box refinement.

- **RoI Pooling:** Region of Interest (RoI) Pooling extracts fixed-size feature maps from the convolutional feature maps for each proposal, regardless of the proposal's size. This ensures that the detection network can process regions of varying sizes.
- **Classification and Regression:** The Fast R-CNN network consists of fully connected layers that classify the proposals into object categories (e.g., car, person) and refine the bounding box coordinates using regression.

3.2.4 Training Procedure

The training procedure of Faster R-CNN employs a 4-step alternating training strategy to optimize both the RPN and the Fast R-CNN detection network efficiently (Ren et al., 2015). In the first step, the RPN is trained using pre-trained convolutional layers (e.g., from a model like VGG16) to generate region proposals. These proposals are then used in the second step to train the Fast R-CNN detection network, which classifies the proposals and refines their bounding box coordinates. Once the detection network is trained, the third step involves fine-tuning the RPN using the updated detection network, ensuring that the RPN generates higher-quality proposals that align with the detection network's improved understanding of object boundaries. Finally, in the fourth step, the Fast R-CNN detection network is fine-tuned again using the refined proposals from the updated RPN. This alternating training process allows the RPN and detection network to iteratively improve each other, leading to a more accurate and robust object detection system. Importantly, the convolutional layers are shared between the RPN and detection network throughout this process, enabling end-to-end training and significantly reducing computational overhead compared to earlier methods like R-CNN and Fast R-CNN.

3.2.5 Implementation Details

The implementation of Faster R-CNN requires attention to both hardware and software aspects, as well as data preparation and training pipeline specifics. On the hardware side, GPUs are essential for training Faster R-CNN due to the heavy computational demands of convolution operations. GPUs such as NVIDIA's RTX series or Tesla GPUs are widely used, with at least 8GB of GPU memory recommended for handling the large intermediate feature maps and anchors. Additionally, 16GB or more of system RAM is often required for efficient data loading and preprocessing, while SSDs are preferred for managing large datasets like Common Objects in Context dataset (COCO) or ImageNet, which can exceed 100GB in size.

On the software side, Faster R-CNN is typically implemented using frameworks like PyTorch, TensorFlow, or Keras. PyTorch, with its torchvision library, offers pretrained Faster R-CNN models that are easy to fine-tune. TensorFlow provides a model zoo with trained weights, while Detectron2 by Facebook AI Research is a specialized library optimized for object detection, including Faster R-CNN. These frameworks simplify the implementation process while allowing customization of key components such as backbones, anchors, and optimization strategies.

Data preparation plays a critical role. Input images are typically resized to a fixed size (e.g., 600–1024 pixels on the shorter side) while maintaining the aspect ratio. Annotations, including bounding boxes and class labels, are converted into formats like Pascal VOC or COCO, which are widely supported. Data augmentation is used to improve generalization, employing techniques like flipping, rotation, and cropping.

For model configuration, the backbone network is usually a pretrained CNN like ResNet or VGG-16. These networks extract features from the input images, which are then passed to the Region Proposal Network (RPN). Anchors are configured with predefined sizes (e.g., 128×128, 256×256) and aspect ratios (e.g., 1:1, 1:2, 2:1). During training, hyperparameters such as the learning rate (commonly starting at 0.001) and batch size (limited to 1 or 2 images per GPU) are optimized using methods like stochastic gradient descent (SGD) or Adam. Training can be performed in two stages: first training the RPN and detection head separately, then fine-tuning them jointly. Alternatively, modern implementations allow end-to-end training, reducing complexity.

3.2.6 Performance Metrics

Evaluating Faster R-CNN’s performance involves using metrics that measure its accuracy, speed, and robustness. The most important metric is Mean Average Precision (mAP), which evaluates how well the model detects objects across different classes and thresholds. mAP combines precision, which measures the proportion of correct detections among all predictions, and recall, which measures the proportion of detected objects among all ground truth objects.

$$\text{Precision} = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)}$$

$$\text{Recall} = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Negatives\ (FN)}$$

A key aspect of mAP is the Intersection over Union (IoU), which calculates the overlap between predicted and ground truth bounding boxes. For a detection to be considered correct, the IoU must exceed a certain threshold, commonly set to 0.5.

$$\text{IoU} = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

Different IoU thresholds (e.g., 0.5, 0.75) affect the evaluation. For example, a higher IoU threshold (e.g., 0.75) requires more precise bounding box predictions, making the evaluation stricter.

mAP can be calculated for individual object classes by plotting precision-recall curves and computing the area under the curve (AUC).

The Precision-Recall (PR) curve is a graphical representation of a model's performance, particularly in tasks like object detection where the goal is to balance precision and recall. In object detection, the model outputs bounding boxes along with confidence scores (probabilities) for each prediction. By varying the confidence threshold (e.g., from 0 to 1), you can control which predictions are considered "positive." A higher threshold results in fewer but more confident predictions, while a lower threshold results in more predictions, including less confident ones. For each confidence threshold, compute precision and recall. As the confidence threshold decreases, recall typically increases (more objects are detected), but precision may decrease (more false positives are included). The PR curve is created by plotting precision (y-axis) against recall (x-axis) for all confidence thresholds. The curve typically starts at high precision and low recall (high confidence threshold) and ends at low precision and high recall (low confidence threshold).

The overall mAP is the mean of the AP values across all classes. The COCO dataset introduces a more comprehensive metric, mAP @[0.5:0.95], which averages AP values over IoU thresholds ranging from 0.5 to 0.95 in steps of 0.05. This provides a detailed evaluation of the model's performance across different levels of bounding box accuracy. COCO also includes metrics for small, medium, and large objects, highlighting the model's ability to detect objects of varying sizes.

Another important consideration is the speed of Faster R-CNN, measured in frames per second (FPS). While Faster R-CNN achieves high accuracy, its FPS is lower compared to models like YOLO, making it less suitable for real-time applications. Specifically, Faster R-CNN typically processes 5 to 7 FPS, whereas YOLOv4 achieves 50 to 60 FPS, and SSD handles 20 to 30 FPS. Despite its slower speed, Faster R-CNN is well-suited for tasks that demand high precision where accuracy is more critical than real-time performance.

3.3 Multi-Task Road Extractor (MTRE)

Traditional methods for road extraction often rely on manual interpretation or single-task deep learning models, which struggle with challenges such as occlusions, complex backgrounds, and variability in road appearance.

To address these limitations, Multi-Task Road Extractors have emerged as a powerful solution. These models simultaneously learn multiple related tasks, such as road segmentation, centerline detection, and boundary detection, leveraging shared features to improve accuracy and robustness.

The multi-task learning paradigm allows the model to generalize better by learning complementary information from different tasks. For example, while road segmentation provides pixel-level classification, centerline detection ensures topological correctness, and boundary detection refines the edges of the roads. This holistic approach has been shown to outperform single-task models in various benchmarks, such as the DeepGlobe Road Extraction Challenge (Demir et al., 2018).

3.3.1 Architecture Overview

The architecture of a Multi-Task Road Extractor is designed to simultaneously address multiple related tasks, such as road segmentation, centerline detection, and boundary detection, by leveraging a shared encoder and multiple task-specific decoders. This framework is illustrated in figure 3.8, which depicts a common multi-task learning architecture with a single encoder and multiple decoders. The shared encoder extracts high-level features from the input image, while the task-specific decoders process these features to produce outputs for each task.

The shared encoder typically consists of a deep convolutional neural network (CNN) backbone, such as U-Net (Ronneberger et al., 2015), or ResNet (He et al., 2016), which is responsible for capturing hierarchical features from the input image. These features are then passed to multiple decoders, each tailored to a specific task. For example, Decoder 1 might focus on road segmentation, producing a pixel-wise classification map, while Decoder 2 could handle centerline detection, predicting the topological structure of the road network.

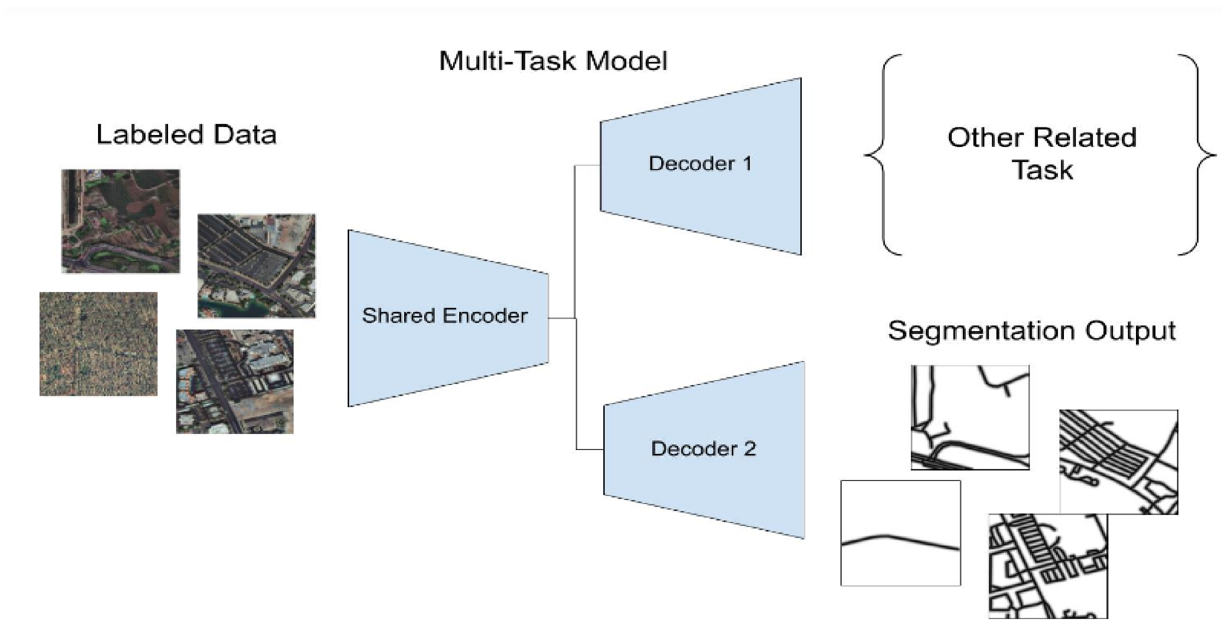


Figure 3.8: Shared encoder with multiple decoders for tasks like segmentation and centerline detection. Adapted from How Multi-Task Road Extractor works (Esri, n.d.)

This modular design allows the model to share feature representations across tasks, improving efficiency and performance.

A more detailed architecture for a Multi-Task Road Extractor is shown in figure 3.9, which presents a stacked encoder-decoder framework with additional components for feature fusion and multi-branch processing. The shared encoder incorporates skip connections, which help preserve spatial information by combining low-level and high-level features. This is particularly important for tasks like road segmentation, where fine-grained details are crucial. The encoder also includes modules for feature stacking and fusion, which integrate information from different branches to enhance the model's ability to capture complex patterns.

The multi-branch architecture consists of several specialized modules, such as the Segmentation Branch and the Orientation Branch, each designed to address a specific aspect of road extraction. The Segmentation Branch processes the shared features to produce pixel-wise road classifications, while the Orientation Branch focuses on detecting road centerlines and boundaries. These branches are interconnected through fusion modules, which combine their outputs to ensure consistency and coherence across tasks.

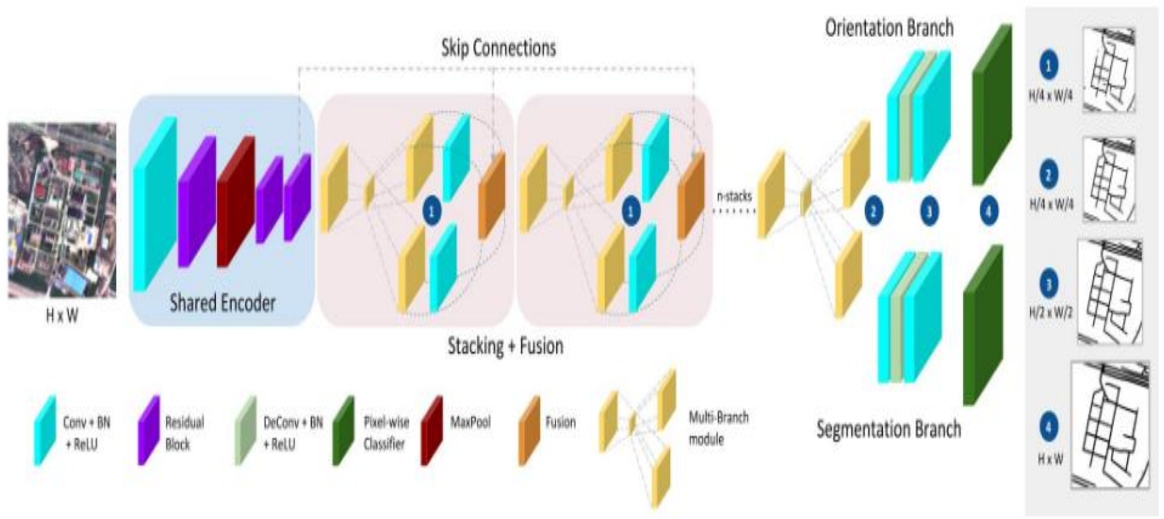


Figure 3.9: Skip connections, feature fusion, and task-specific branches for road extraction tasks. Adapted from How Multi-Task Road Extractor works (Esri, n.d.).

The use of convolutional layers with batch normalization (Conv+BN) and rectified linear unit (ReLU) activations ensures efficient feature extraction and non-linear transformation. Additionally, residual connections are employed to facilitate gradient flow and enable the training of deeper networks. The final outputs from each branch are passed through classifiers to generate the task-specific predictions, such as segmentation masks, centerline maps, and boundary delineations.

By combining a shared encoder with task-specific decoders and incorporating advanced techniques like skip connections, feature fusion, and residual learning, the Multi-Task Road Extractor achieves state-of-the-art performance in road extraction tasks. This architecture not only improves accuracy but also enhances the model's ability to generalize across diverse datasets and challenging conditions.

After the shared backbone, separate branches (or heads) are added for each task. These heads are typically lightweight convolutional layers that process the shared features to produce task-specific outputs. For example, the segmentation head produces a pixel-wise classification map indicating road regions, the centerline head predicts the centerlines of roads, which are crucial for ensuring topological correctness, and the boundary head detects the edges of roads, refining the segmentation results. The model is trained using a weighted combination of losses for each task. The total loss can be expressed as:

$$L_{total}=\lambda_1 L_{seg}+\lambda_2 L_{center} +\lambda_3 L_{boundary} \qquad (3.12)$$

where L_{seg} is the segmentation loss, L_{center} is the centerline detection loss, $L_{boundary}$ is the boundary detection loss, and λ_1, λ_2, and λ_3 are task-specific weights that balance the contributions of each loss.

3.3.2 Training Procedure

The training procedure for a Multi-Task Road Extractor involves several steps. First, the input data, which consists of high-resolution satellite or aerial images, is prepared along with ground truth labels for road segmentation, centerlines, and boundaries. Data augmentation techniques like rotation, flipping, and scaling are used to increase the diversity of the training data. The shared backbone is initialized with pre-trained weights, often from ImageNet, to leverage transfer learning, while the task-specific heads are initialized randomly.

The model is then trained end-to-end using a multi-task loss function. The optimizer, such as Adam or SGD, updates the model parameters to minimize the total loss. Learning rate scheduling and early stopping are employed to prevent overfitting. After initial training, the model can be fine-tuned on specific datasets to improve performance in particular regions or conditions. This fine-tuning process ensures that the model adapts well to the unique characteristics of the target dataset.

3.3.3 Implementation Details

Implementing a Multi-Task Road Extractor requires careful consideration of several factors. Popular deep learning frameworks like TensorFlow or PyTorch are typically used for implementation, with libraries such as MMSegmentation or Detectron2 providing pre-built modules for multi-task learning. Training is usually performed on GPUs, such as the NVIDIA Tesla V100, due to the computational complexity of deep learning models.

Key hyperparameters, including the learning rate, batch size, and loss weights (λ_1 , λ_2 ,and λ_3), are tuned using cross-validation or grid search. The model is evaluated on a validation set during training to monitor performance and avoid overfitting. This iterative process of training, validation, and fine-tuning ensures that the model achieves optimal performance on the target task.

3.3.4 Performance Metrics

The performance of a Multi-Task Road Extractor is evaluated using several metrics.

- **For road segmentation,**

  This involves classifying each pixel in an image as either "road" or "non-road. The performance metrics for this task are similar to those used in semantic segmentation:

  1. Pixel Accuracy: Measures the proportion of correctly classified pixels.

  $$\text{Pixel Accuracy}=\frac{\textbf{True Positives (TP)+True Negatives (TN)}}{\mathit{Total\ Pixels}}$$

  While simple, pixel accuracy can be misleading in imbalanced datasets where the "non-road" class dominates.

  2. The Intersection over Union (IoU) or Jaccard Index measures the overlap between predicted and ground truth road regions. IoU is more robust than pixel accuracy, especially for imbalanced datasets.

  $$\text{IoU} = \frac{\textbf{Area of Overlap}}{\boldsymbol{Area\ of\ Union}}$$

  Another important metric is dice coefficient (F1 Score) which is a harmonic mean of precision and recall, commonly used in segmentation tasks.

Dice Coefficient= $\frac{2\ x\ \mathrm{TP}}{2\ x\ TP+FP+FN}$

Precision and recall metrics are calculated at the pixel level. A high precision indicates fewer false positives, while high recall indicates fewer false negatives.

- **For Road Centerline Detection**

  Road centerline detection involves identifying the centerlines of roads, which are often represented as thin lines. The performance metrics for this task include:

1. **F1 Score:** Measures the balance between precision and recall for detected centerlines.

F1 Score= $2\ \boldsymbol{x}\ \frac{\text{Precision}\ x\ \text{Recall}}{\text{Precision}\ +Recall}$

2. **There is also Hausdorff Distance metric which** measures the maximum distance between the predicted and ground truth centerlines. A lower Hausdorff distance indicates better alignment.
3. **Topological Accuracy:** Evaluates how well the predicted centerlines preserve the topological structure of the road network (e.g., connectivity and intersections).

- **For Road Boundary Detection**

  Road boundary detection involves identifying the edges of roads. The performance metrics for this task are similar to those for road segmentation but focus on boundary pixels:

Boundary IoU: Measures the overlap between predicted and ground truth road boundaries.

Boundary IoU = $\frac{\text{Area of Overlap (Boundary)}}{Area\ of\ Union\ (Boundary)}$

Precision and Recall metrics are calculated specifically for boundary pixels.

- **For Multi-Task Evaluation Metrics**

Since the Multi-Task Road Extractor performs multiple tasks simultaneously, it is important to evaluate its overall performance across all tasks. This can be done using:

1. **Weighted Average of Task-Specific Metrics:** Combine the metrics for each task (e.g., IoU for segmentation, F1 score for centerline detection) using task-specific weights that reflect their relative importance.

   Overall Score=w_1×IoU+w_2×F1 Score+w_3×Boundary IoU where w_1, w_2, w_3 are the weights for segmentation, centerline detection, and boundary detection, respectively.

2. Task Balancing: Evaluate how well the model balances performance across all tasks. For example, a model that performs well on segmentation but poorly on centerline detection may not be suitable for applications requiring accurate centerlines.

CHAPTER 4

DATA ACQUISITION AND PRE-PROCESSING

CHAPTER 4

DATA ACQUISITION AND PRE-PROCESSING

The availability of high-quality, large-scale datasets is a critical challenge in training deep learning models, particularly for complex tasks like road detection from satellite imagery. Traditional methods of data preparation, such as manual digitization, are time-consuming, labor-intensive, and often impractical given the large scale of satellite data. These methods do not align with the rapid advancements in deep learning, which require massive amounts of accurately labeled data to achieve state-of-the-art performance. Satellite images add another layer of complexity due to their high resolution, variability in lighting and terrain, and the presence of occlusions like clouds or shadows.

4.1 Ready-made datasets

To address these challenges, researchers have increasingly turned to ready-made datasets such as ImageNet, COCO, SpaceNet, and others which provide large volumes of pre-processed and annotated data. These datasets have become foundational resources for training deep learning models, enabling researchers to bypass the labor-intensive process of manual annotation. For example, ImageNet (Deng et al., 2009), with its millions of labeled images across thousands of categories, has been instrumental in advancing computer vision tasks through transfer learning. Similarly, the COCO dataset (Lin et al., 2014) offers a rich collection of images with detailed annotations for object detection, segmentation, and captioning, making it a valuable resource for training models on complex tasks.

In the context of satellite imagery, SpaceNet (Van Etten et al., 2018) has emerged as a critical dataset for geospatial applications. It provides high-resolution satellite images with annotations for buildings, roads, and other urban features, enabling the development of models for tasks like road extraction and urban planning. By leveraging these datasets, researchers can focus on fine-tuning pre-trained models rather than starting from scratch, significantly reducing the time and resources required for training.

The use of ready datasets also aligns with the concept of transfer learning, where models pre-trained on large datasets like ImageNet or COCO are adapted to specific tasks using smaller, domain-specific datasets like part of SpaceNet as in this study. This approach has been shown to improve model performance, especially when labeled data is scarce (Pan & Yang, 2010). Additionally, the availability of these datasets has spurred the development of benchmarking frameworks, allowing researchers to compare their models against state-of-the-art methods and identify areas for improvement.

Despite their advantages, ready datasets are not without limitations. For instance, the annotations in datasets like COCO or SpaceNet may not always align perfectly with the specific requirements of a given task, necessitating additional fine-tuning or data augmentation. Moreover, the diversity of satellite imagery—ranging from rural to urban environments—requires datasets that are representative of the target domain. To address this, researchers often combine multiple datasets or use techniques like data augmentation and synthetic data generation to enhance the diversity and robustness of their training data.

4.2 Study Area

In this research, the SpaceNet 3: Road Network Detection dataset was utilized, a cornerstone resource for advancing geospatial machine learning applications. Among the available datasets within SpaceNet 3, the Paris dataset was specifically chosen for training the model. This selection was motivated by several key factors that align with the objectives of this research.

Satellite imagery, combined with co-registered map features, has significantly advanced geospatial analysis. Prior to the launch of SpaceNet (Van Etten et al., 2018), computer vision researchers had limited access to openly available satellite imagery with precise annotations. SpaceNet now hosts datasets generated by its own team, as well as contributions from initiatives such as IARPA's Functional Map of the World (fMoW). The commercialization of the geospatial industry has led to a substantial increase in available data for monitoring global changes, creating opportunities for innovation in applying computer vision and deep learning techniques to extract large-scale information from satellite imagery.

In this regard, CosmiQ Works, Radiant Solutions, and NVIDIA collaborated to make the SpaceNet dataset publicly available, providing a valuable resource for developers and data scientists. The Paris dataset, a subset of SpaceNet 3, encompasses over 400 kilometers of roads, categorized into various types such as motorways, primary, and tertiary roads. Table 4.1 presents the distribution of road types and their respective lengths.

Table 4.1 Breakdown of Road Types and Lengths in Paris

Road Type	**Length (Km)**
Motorway	9
Primary	14
Secondary	58
Tertiary	11
Residential	232
Unclassified	95
Cart track	6
Total	**425**

The dataset provides high-resolution satellite images accompanied by road masks (ground truth), where each image has a corresponding GeoJSON file representing the roads in GeoJSON format. The geographic extent of the study area as shown in figure 4.1 is approximately bounded by the coordinates (2.20°E, 48.80°N) in the southwest and (2.46°E, 48.91°N) in the northeast, covering a significant portion of the Paris metropolitan area. The imagery was captured by DigitalGlobe's WorldView-3 satellite, and includes multiple types of imagery: 8-band Multi-Spectral (MS) at 1.24m resolution, Panchromatic (PAN) at 0.3m resolution, Pan-sharpened Multi-Spectral (PS-MS) at 0.3m resolution, and Pan-sharpened RGB (PS-RGB) at 0.3m resolution, this study utilizes the PS-RGB imagery. The dataset contains 425 km of road centerline vectors, labeled according to OpenStreetMap guidelines, with attributes such as road type, surface type, and lane number. For this study, 314 PS-RGB images, each with dimensions of 1300 × 1300 pixels and stored in 16-bit unsigned integer (uint16) format, were used. The total size of the dataset is about 7.3 GB. The geographic coordinate system (GCS) used is WGS 84, ensuring global compatibility and accurate georeferencing.

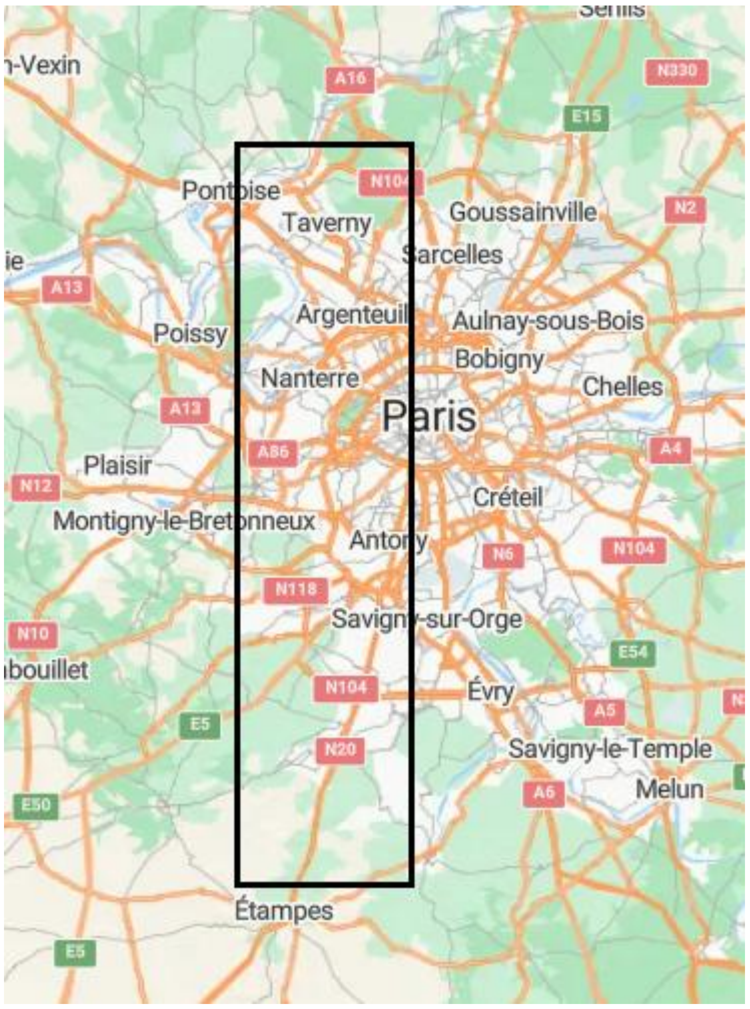


Figure 4.1: The study area bounded by the black rectangle

The Paris dataset was chosen for this study due to its diversity in road types, which includes highways, residential streets, and pedestrian pathways. This diversity ensures that the model can learn to detect roads in various contexts, improving its generalizability to different regions or environments. Additionally, the dataset is distributed across approximately 300 high-resolution satellite images, providing a substantial yet manageable volume of data. This balance is crucial for ensuring computational efficiency during training while maintaining model performance. A dataset of this size allows for sufficient training iterations without incurring excessively long processing times, which is particularly important given the computational demands of deep learning algorithms. Furthermore, the Paris dataset is known for its high-quality annotations, which are critical for supervised learning tasks like road detection. The availability of accurate and detailed road network labels reduces the need for manual correction or additional preprocessing, streamlining the data preparation phase.

4.3 Downloading Data

As discussed in the previous section, the dataset used in this study is hosted on Amazon Web Services (AWS), a comprehensive and widely adopted cloud computing platform provided by Amazon.com. AWS offers a vast network of data centers and servers that provide a wide range of services over the internet, including storage, computing power, and databases. Instead of setting up and managing physical infrastructure, users can rent these resources on a pay-as-you-go basis, making it an efficient and scalable solution for handling large datasets like the SpaceNet 3: Road Network Detection dataset.

To download the data, the AWS Command Line Interface (CLI) must be installed on the local computer. The AWS CLI is a unified tool that allows users to interact with AWS services directly from the command line. Once installed, users can write commands in the Command Prompt (CMD) or terminal to access and download the required data from AWS servers.

Steps to Download Data:

1. Install AWS CLI : The first step is to install the AWS CLI v2 on the local machine. This can be done by downloading the installer from the official AWS website and following the installation instructions for the operating system in use (Windows, macOS, or Linux).

2. Configure AWS CLI: After installation, the AWS CLI must be configured with the user's credentials (Access Key ID and Secret Access Key) to authenticate access to the AWS services. This is done using the command:

"aws configure"

The user will be prompted to enter their AWS credentials, default region, and output format.

3. Download Data: Once the AWS CLI is configured, users can download the dataset using specific commands. For example, to download the satellite images for the training data from the Paris dataset, the following command is used: *"aws s3 cp s3://spacenet-dataset/spacenet/SN3_roads/tarballs/SN3_roads_train_AOI_3_Paris.tar.gz ."*. This command downloads the compressed file containing the satellite images to the current directory.

4. Download GeoJSON Files: To download the corresponding GeoJSON files, which contain the road network annotations in GeoJSON format, the following command is used: *"aws s3 cp* *s3://spacenetdataset/spacenet/SN3_roads/tarballs/SN3_roads_train_AOI_3_Paris_geojson_roads_speed.tar.gz .”.* This command downloads the compressed file containing the GeoJSON annotations to the same directory.

The downloaded data consists of two main components: Satellite Images: High-resolution images of the study area (Paris) in GeoTIFF format and GeoJSON Files: Annotations of the road network, including road types (e.g., motorways, primary, and tertiary roads) and their geometries, stored in GeoJSON format.

4.4 Data pre-processing

In this research, two primary models were employed for road detection: Faster R-CNN and Multi-Task Road Extractor. Each model has unique input format requirements, necessitating a meticulous and multi-step data preparation process. While the overall workflow was consistent across both models, specific adjustments were made to ensure compatibility with their respective architectures.

After downloading the data, the first step was to convert the GeoJSON files into Shapefiles for easier manipulation and processing. This conversion was achieved using a Python script (provided in the appendix) that leveraged the arcpy library, a powerful tool for geospatial data processing. The Shapefile format was chosen due to its compatibility with GIS software and its ability to store geometric and attribute data in a structured manner.

Satellite images often have a high radiometric resolution, typically stored in uint16 format, which provides a wide dynamic range for pixel values. However, most deep learning models, including Faster R-CNN and U-Net, are optimized for uint8 images, which have a pixel value range of 0 to 255. To align the data with the input requirements of these models, the radiometric resolution of the satellite images was converted from uint16 to uint8. This conversion was performed using a MATLAB script provided in the appendix, which normalized the pixel values and scaled them to the 8-bit range. This step was critical to ensure that the models could process the images efficiently without losing essential information.

Faster R-CNN, being an object detection model, requires data to be formatted in a specific way to facilitate training and inference. The model expects input data in the Pascal VOC format, a widely used standard for object detection tasks. The Pascal VOC format includes image files and annotation files in XML format, which contain bounding box coordinates and class labels for each object in the image. To prepare the data for Faster R-CNN, the Shapefiles containing the road annotations were processed to extract bounding boxes for each road segment. These bounding boxes were then converted into Pascal VOC-compatible XML files using the ia module in arcpy library, this is provided in the appendix. The script also ensured that the annotations were correctly aligned with the corresponding satellite images.

The Multi-Task Road Extractor, based on the U-Net architecture, is designed for pixel-wise segmentation tasks. Unlike Faster R-CNN, which relies on bounding boxes, this model requires mask annotations that delineate the exact boundaries of the roads. To prepare the data for the Multi-Task Road Extractor, the Shapefiles were converted into binary masks, where each pixel was labeled as either "road" or "non-road." This conversion was performed using the rasterio library in Python, which allowed for efficient manipulation of geospatial raster data. The resulting masks were saved as grayscale images, with pixel values of 255 representing roads and 0 representing the background.

To enhance the robustness and generalizability of the models, data augmentation techniques were applied to the training dataset. These techniques included random rotations, flips, and brightness adjustments, which helped simulate variations in lighting, orientation, and scale that the models might encounter in real-world scenarios. Additionally, the dataset was split into training, and validation using an 80-20 ratio. This split ensured that the models were evaluated on unseen data, providing a reliable measure of their performance. The code of data augmentation and dividing data is included in the training code which will be discussed in the next chapter.

Once the data was processed and formatted, it was organized into a structured directory layout to facilitate easy access during training. For Faster R-CNN, the directory contained subfolders for images and annotations, while for the Multi-Task Road Extractor, separate folders were created for images and their corresponding masks. This organization streamlined the data loading process and ensured that the models could access the required files efficiently.

The data preparation process was not without challenges. One significant issue was the misalignment between the satellite images and the GeoJSON annotations, which occasionally occurred due to differences in coordinate systems or projection errors. To address this, a geometric transformation was applied to align the annotations with the images accurately. Additionally, the large file sizes of the satellite images posed computational challenges, which were mitigated by tiling the images into smaller patches and processing them in batches.

Figure 4.2 illustrates an example of these images alongside their corresponding road ground truth.

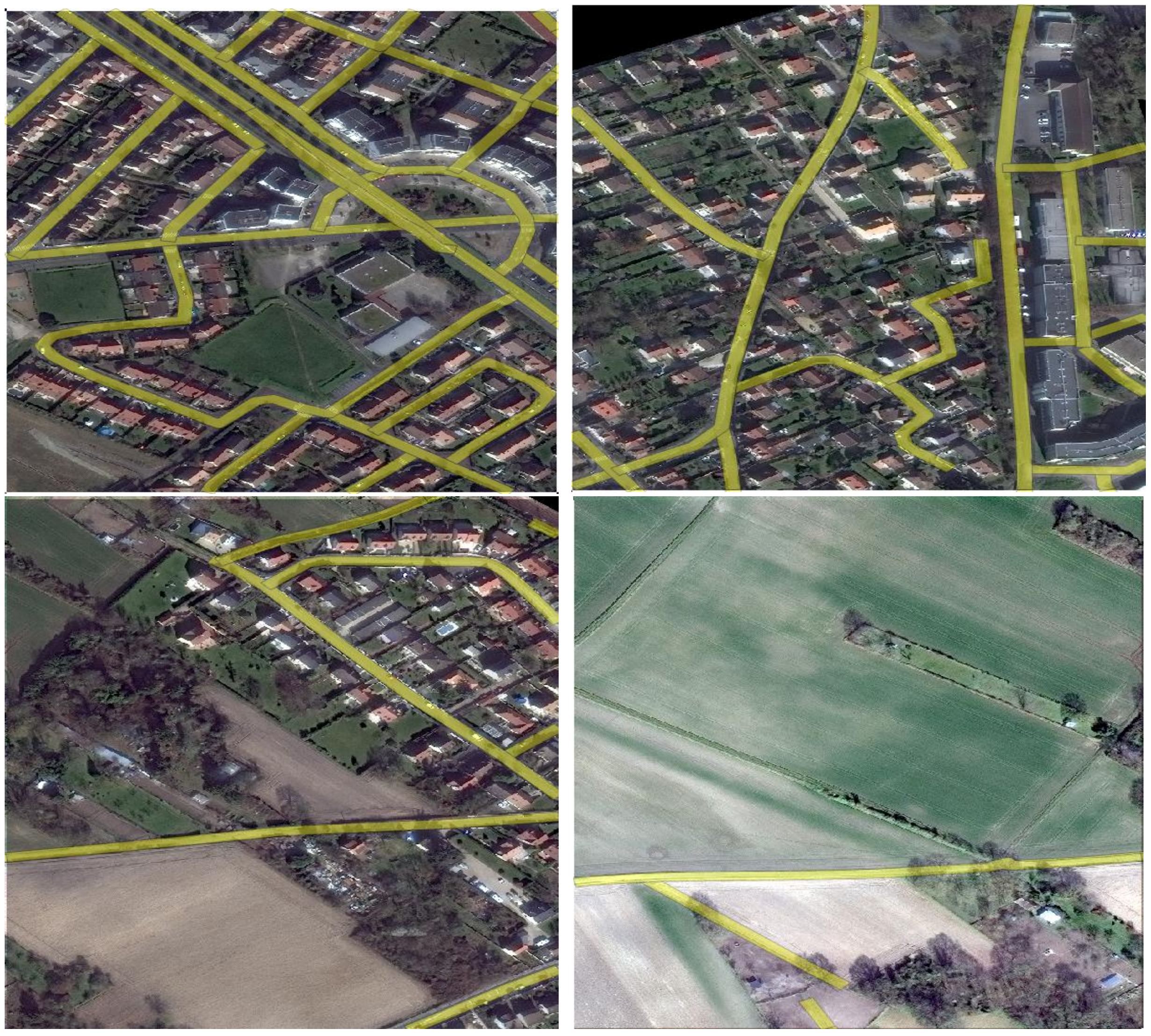


Figure 4.2. Sample of the satellite images for the study area with road ground truth

CHAPTER 5

MODEL TRAINING AND EXPERIMENTAL RESULTS

CHAPTER 5

MODEL TRAINING AND EXPERIMENTAL RESULTS

Training deep learning models is a complex and iterative process that involves several key steps: preparing data, selecting architectures, configuring hyperparameters, and optimizing the model. While advancements in algorithms and computing power have significantly improved the efficiency of training, several challenges remain, particularly when applying these models to real-world scenarios. Once the algorithms have been selected, the next critical step is choosing the appropriate parameters and hyperparameters that align with the available computational resources.

Deep learning algorithms rely heavily on parallel processing rather than serial processing to perform computations efficiently. This is why GPUs (Graphics Processing Units) are preferred over CPUs (Central Processing Units) for training deep learning models. GPUs are designed to handle thousands of computations simultaneously, making them ideal for the matrix and vector operations that are fundamental to deep learning. In contrast, CPUs process tasks sequentially, which is significantly slower for large-scale deep learning workloads. Figure 5.1 illustrates the difference between parallel and serial processing.

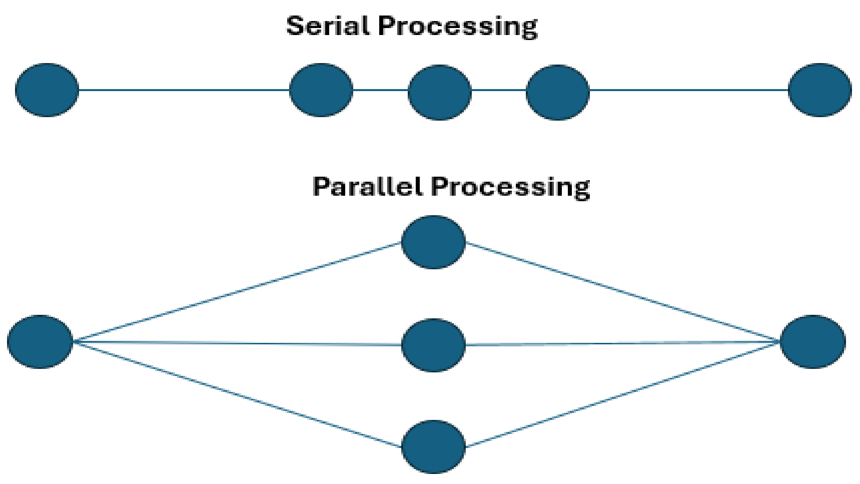


Figure 5.1: Serial and parallel processing, where each circle represents a process.

The emergence of deep learning libraries such as PyTorch and TensorFlow is largely due to the capabilities of GPUs. While libraries like NumPy provide essential functions and attributes for numerical computations, these operations are performed on CPUs, limiting their scalability for deep learning tasks. In contrast, PyTorch and TensorFlow are specifically designed to leverage the parallel processing power of GPUs, enabling faster and more efficient training of deep learning models. As a result, most of the training code in this study utilizes PyTorch and TensorFlow to take full advantage of GPU acceleration.

5.1 Training Environment

The most critical aspect of the training process is the hardware used to execute the computations. In this study, the training was performed on a machine equipped with an NVIDIA GeForce GTX 1060 Max-Q GPU and 16 GB of RAM. For training, coding, and data manipulation, ArcGIS Pro 2.8 was utilized. ArcGIS Pro was selected due to its user-friendly interface for handling and visualizing geospatial images, as well as its seamless integration with Jupyter Notebook, a widely used interface for writing and executing Python code. This integration allows for efficient coding, visualization of results, and the ability to install and import essential Python libraries directly within the environment.

The primary libraries used in this work include NumPy, PyTorch, TensorFlow, and Matplotlib.

- **NumPy (Numerical Python)**: A foundational library for scientific computing in Pytho n, NumPy provides support for large, multi-dimensional arrays and matrices, along with a comprehensive collection of mathematical functions to operate on these structur es efficiently. It is optimized for CPU-based computations.
- **PyTorch**: An open-source deep learning framework developed by Facebook's AI Rese arch lab (FAIR), PyTorch is designed for GPU-accelerated tensor computations. It offe rs dynamic computation graphs, making it highly flexible and well-suited for research a nd development in deep learning.
- **TensorFlow**: An open-source deep learning framework developed by Google, TensorF low is designed for both research and production environments. It supports static comp utation graphs and provides extensive capabilities for distributed computing. TensorFlo w excels in leveraging GPU and TPU (Tensor Processing Unit) acceleration, making it ideal for large-scale machine learning tasks.

- **Matplotlib**: A comprehensive plotting library for Python, Matplotlib is widely used for creating static, animated, and interactive visualizations. It is an essential tool for data a nalysis and model evaluation, enabling clear and insightful representation of results.
- **fastai.vision**: A high-level module within the FastAI library, built on PyTorch, designed to simplify and accelerate computer vision tasks. It provides tools like vision_learner and ImageDataLoaders for easy data handling, model training, and transfer learning with pre-trained models (e.g., ResNet) for tasks like image classification and segmentation.
- **arcgis.learn**: A module in the ArcGIS API for Python, tailored for geospatial deep learning. It integrates with PyTorch and FastAI to offer models (e.g., UnetClassifier, FasterRCNN) for tasks like object detection and pixel classification on satellite imagery, streamlining workflows within the ArcGIS ecosystem.

To leverage the computational power of the NVIDIA GeForce GTX 1060 Max-Q GPU, the CUDA Toolkit was installed and configured. CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) developed by NVIDIA, enabling developers to use NVIDIA GPUs for general-purpose processing (NVIDIA, 2023). By utilizing CUDA, the training process was significantly accelerated, as the GPU's parallel processing capabilities were harnessed to perform computationally intensive operations, such as matrix multiplications and convolutions, more efficiently than a CPU. The CUDA Toolkit was integrated with the deep learning frameworks PyTorch and TensorFlow, ensuring seamless compatibility and optimal utilization of the GPU during training. This setup not only reduced the training time but also enabled the handling of larger batch sizes and more complex models, which would have been infeasible with CPU-only computation.

5.2 Training Details and Results of the proposed Faster R-CNN

The training of the model is final step to get the result and start analysing the results. Usually, the well-known complex deep learning models are trained on a specific environment where all components in this environment are dedicated to server for the heavy processing of the model. This leads to the need for staff as a software engineer, devops engineer, network engineer, and others. This is due to the big scale of these projects and the big data that the models learn from. Thanks to the era of AI and Internet, we could mimic as all the previous roles and train the proposed models.

5.2.1 Hyperparameter Tuning of the proposed Faster R-CNN model

In order to avoid the previous common issues in NNs training, a carful understanding of deep learning model parameters and each hyper parameter has to be understood to carefully choose the most appropriate values. These hyperparameters will be addressed

1. **Learning Rate (α):** The learning rate determines the step size at which the optimization algorithm updates the model's weights during training. A high learning rate can cause the model to converge quickly but may overshoot the optimal solution, while a low learning rate results in slow convergence but ensures more precise updates. Techniques like learning rate decay or adaptive learning rates (e.g., with Adam or RMSProp optimizers) help adjust the learning rate dynamically. In practice, there are many approaches to use an apropriate learning rate, in this training, the built-in function in ArcGIS API for Python 'lr_find' is used to find the optimal learning rate. The function begins by initializing the model with a very low learning rate. The learning rate is increased exponentially over a number of iterations. As the learning rate increases, the loss is tracked at each step. After the process is complete, the function typically plots the loss vs. learning rate, showing where the loss starts decreasing and where it begins to diverge as in figure 5.3, it shows a value which is approximately 0.00009.

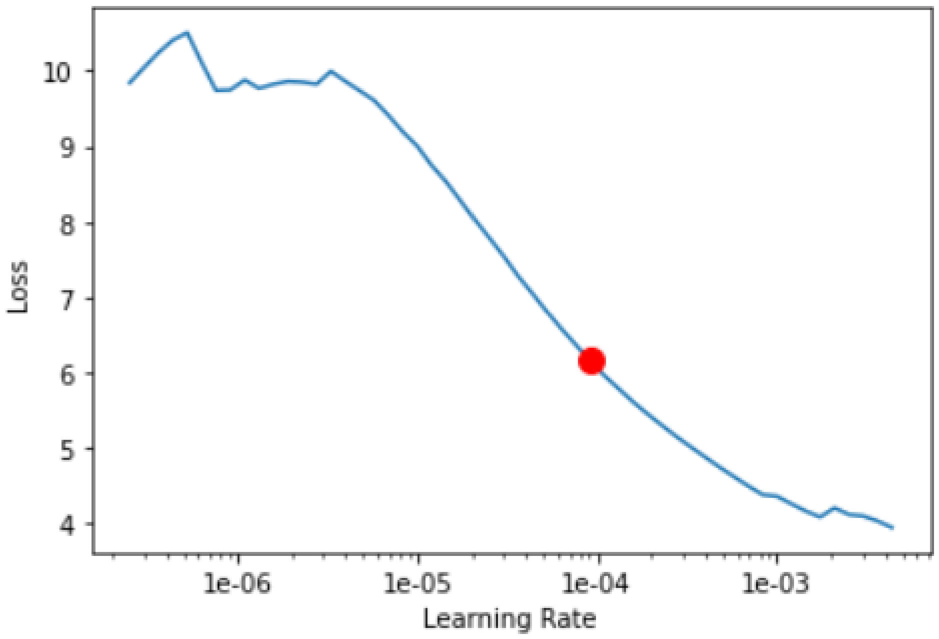


Figure 5.2: Learning rate versus loss

2. **Batch Size:** Batch size specifies the number of training samples processed before the model updates its parameters. Smaller batch sizes provide noisy but frequent updates, potentially improving generalization, while larger batch sizes offer more stable updates but require more memory and may converge slower in some cases. Selecting the right batch size depends on hardware limitations (e.g., GPU memory), model complexity, dataset size, and training stability. Common batch sizes range from 16 to 512, with smaller sizes often providing better generalization and larger sizes enabling faster convergence. According to this, a batch size of 4 was used. The code used to estimate the batch size is provided in the appendix .

3. **Number of Epochs:** Epochs represent the number of complete passes through the training dataset during training. Too few epochs can lead to underfitting, while too many may result in overfitting. Early stopping is a technique used to halt training when performance on a validation set stops improving. Using this technique has led to 25 epochs in the training of the Faster R-CNN.

4. **Optimizer:** The optimizer determines how the model updates its weights during training. Common choices include Stochastic Gradient Descent (SGD), Adam, RMSProp, and AdaGrad. Each optimizer has its strengths and weaknesses, and their performance often depends on the specific problem and dataset. SGD with momentum was used in the original paper of Faster R-CNN, so it has been applied as it is in this study

5. **Weight Initialization:** The method for initializing the weights of a deep learning model can impact its convergence. Popular initialization techniques include random initialization, Xavier initialization (Glorot), and He initialization, which are chosen based on the type of activation functions used in the network. Since, transfer learning is applied in this study. The final weights of the original model has been used as the initial weights, these weights will be updated continuously through training epochs.

6. **Momentum:** Momentum is used in optimization algorithms like SGD with momentum. It accelerates gradient descent by using the direction of previous gradients to smooth updates

and avoid oscillations, particularly in regions with high curvature. In literature the value of the momentum is always 0.9, and this is what used in this study.

7. **Regularization Parameters:** These include L1 and L2 penalties, as well as other techniques like weight decay and dropout. Regularization parameters control the strength of these techniques to prevent overfitting. In the original Faster R-CNN paper A weight decay (L2 regularization) of **0.0005** was applied to prevent overfitting. It has been kept as it is in this work.

8. **Data Augmentation Parameters:** Data augmentation introduces variations into the training data to improve generalization. Parameters may include rotations, flips, cropping, scaling, and color adjustments. In this study, the applied techniques include horizontal flips, rotation, zoom, lighting adjustments, warping, and affine transformations.

9. **Backbone Network:** Backbone convolutional neural network model used for feature extraction, while the used version in the paper is VGG16, in this study, resnet 50 has been used, this is for some reasons as:
   - **Deeper Architecture with Residual Connections**: ResNet-50 with its 50 layers, uses residual connections (skip connections) to address the vanishing gradient problem. These connections allow the network to learn more complex features while maintaining stable training, even with a large number of layers. In contrast, VGG-16 has only 16 layers and lacks residual connections, making it less effective for very deep architectures.

   - **Improved Feature Extraction:** ResNet-50's deeper architecture enables it to capture hierarchical features more effectively than VGG-16. This is particularly important for object detection tasks, where fine-grained details (e.g., small objects or complex textures) must be detected. The residual blocks in ResNet-50 allow the network to learn identity mappings, which preserve information and improve gradient flow during backpropagation.

- **Better Performance on Complex Datasets:** ResNet-50 has been shown to outperform VGG-16 on large-scale datasets like ImageNet and COCO. Its ability to handle complex patterns and variations in data makes it a better choice for object detection tasks, especially when dealing with diverse or challenging datasets.

- **Efficiency in Training and Inference:** ResNet-50 is more computationally efficient than VGG-16 due to its use of bottleneck layers (1x1 convolutions) in residual blocks. These layers reduce the number of parameters and computations, making ResNet-50 faster to train and deploy. VGG-16, on the other hand, has a large number of parameters due to its fully connected layers, making it slower and more resource-intensive.

Table 5.1 summarizes the hyper-parameter settings used for training the Faster R-CNN model. These parameters were selected based on common practices in deep learning for object detection tasks.

Table 5.1: Hyper-parameters of the proposed Faster R-CNN model

Hyper parameter	**Value**
Learning Rate	0.00009
Batch Size	4
Number of Epochs	25
Optimizer	SGD
Momentum	0.9
L2	0.0005
Augmentation	Horizontal Flips, Rotation, Zoom, Lighting Adjustment, Warping, Affine Transformations
Backbone Network	resnet50

5.2.2 Results and Analysis of the training of the proposed Faster R-CNN

The training process of the proposed Faster R-CNN model over 25 epochs is summarized in Table 5.2 and visualized in Figure 5.3. The table and graph show the training loss, validation loss, and time per epoch, providing insights into the model's learning dynamics, generalization ability, and computational efficiency.

- **Training Loss:**

  The training loss starts at 0.879766 in the first epoch and steadily decreases to 0.663775 by the 24th epoch. This consistent reduction indicates that the model is effectively learning and optimizing its parameters to fit the training data. The most significant drop in training loss occurs between epochs 0 and 6, where the loss decreases from 0.879766 to 0.741171. This rapid improvement suggests that the model quickly learns the basic features of the dataset during the initial epochs. After epoch 6, the training loss continues to decrease but at a slower rate, indicating that the model is refining its understanding of more complex patterns in the data.

- **Validation Loss:**

  The validation loss starts at 0.885979 and decreases to 0.755161 by the 24th epoch. This parallel reduction in validation loss demonstrates that the model is generalizing well to unseen data and not overfitting to the training set. The validation loss reaches its lowest value at epoch 20 (0.753947) and remains relatively stable afterward, with minor fluctuations. This suggests that the model has likely converged and is not significantly improving further.

  The small gap between training and validation losses throughout the training process is a positive sign, indicating that the model is not overfitting. However, the slight fluctuations in validation loss after epoch 14 could be an early sign of overfitting, which may need to be addressed through techniques like early stopping or increased regularization.

- **Performance Plateau:**

  After epoch 14, the validation loss fluctuates slightly but remains stable, suggesting that the model has reached a performance plateau. This is common in deep learning models, where further training yields diminishing returns. The best validation loss occurs at epoch 20 (0.753947), after which the loss stabilizes with minor variations. This stability indicates that the model has likely achieved its optimal performance on the validation set.

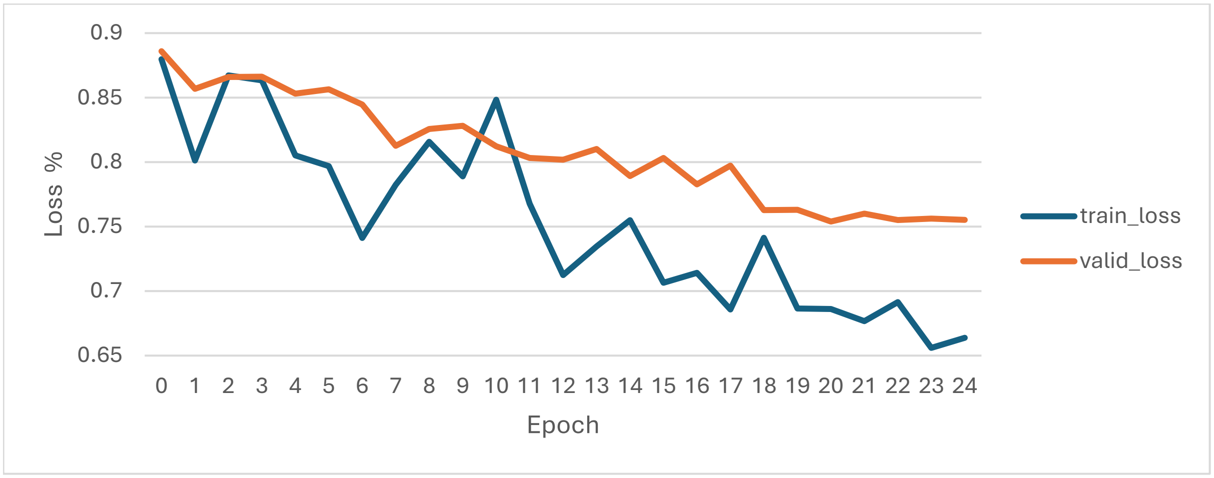


FIGURE 5.3. Training and Validation Loss Over Epochs

Table 5.2: Training metrics of the proposed Faster R-CNN model

Epoch	train_loss	valid_loss	time
0	0.879766	0.885979	42:38:00
1	0.801228	0.856816	40:09:00
2	0.86727	0.865976	40:42:00
3	0.863367	0.866189	45:47:00
4	0.805265	0.853134	52:32:00
5	0.796847	0.85642	42:07:00
6	0.741171	0.844714	40:41:00
7	0.782396	0.812617	40:42:00
8	0.815711	0.825644	40:12:00
9	0.789016	0.828216	39:52:00
10	0.848433	0.812234	39:41:00
11	0.767724	0.803284	40:05:00
12	0.712454	0.801812	40:18:00
13	0.734677	0.810155	40:15:00
14	0.754991	0.789264	40:41:00
15	0.706424	0.803184	40:39:00
16	0.714153	0.782772	40:41:00
17	0.685764	0.797231	40:42:00
18	0.741384	0.762835	40:11:00
19	0.686538	0.763030	40:10:00
20	0.68617	0.753947	40:31:00
21	0.676753	0.760063	40:33:00
22	0.691418	0.75508	40:32:00
23	0.655983	0.756223	40:32:00
24	0.663775	0.755161	40:31:00

Analysis of Training Time

The training time per epoch is generally consistent, ranging from approximately **39 to 45 minutes**, with the exception of epoch 4, which took **52 minutes**. This consistency suggests that the training process is well-optimized and not significantly affected by external factors in most epochs.

The longer training time for epoch 4 (**52 minutes**) can be attributed to several factors:

- **Data Loading and Preprocessing**: The dataset is large and requires significant preproce ssing, such as data augmentation, which may cause delays in specific epochs.
- **Hardware or System Overhead**: Background processes or system resource contention (e.g., CPU/GPU usage by other applications) could cause delays in specific epochs.
    - **Checkpointing or Logging**: If the model saves checkpoints or logs metrics at t he end of each epoch, this process might take longer in some cases, especially if the system is under heavy load.

5.3 Training Details and Results of the proposed MTRE model

Although the Multi-Task Road Extractor (MTRE) is a different algorithm from Faster R-CNN, it is still based on a convolutional neural network (CNN) architecture. Therefore, the training steps, code implementation, and overall workflow are similar to those used for training the Faster R-CNN model.

The model was trained twice in order to validate the impact of training duration on performance. The first attempt was limited to 25 epochs, where early stopping did not take effect, indicating that the model had not yet converged. Therefore, a second training run with an increased number of epochs was conducted to allow the model sufficient opportunities to learn and to assess whether extended training could yield improvements in accuracy and other performance metrics.

5.3.1 Hyperparameter Tuning of the proposed MTRE model

1. **Learning Rate**: Using the same approach as in Faster R-CNN, the learning rate was set to 0.001, as determined by the built-in learning rate scheduler. The corresponding learning rate curve is shown in Figure 5.5.

2. **Batch size:** Since the MTRE model shares similarities with Faster R-CNN in terms of architecture and complexity, the same batch size of 4 was used.

3. **Number of Epochs:** Similar to Faster R-CNN, the early stopping technique was employed in both training trials to halt the process once the validation performance stopped improving. However, in both cases, early stopping did not trigger, which indicated that the model was still learning and had not yet reached a plateau in performance. This justified our decision to continue training with more epochs in the second trial.

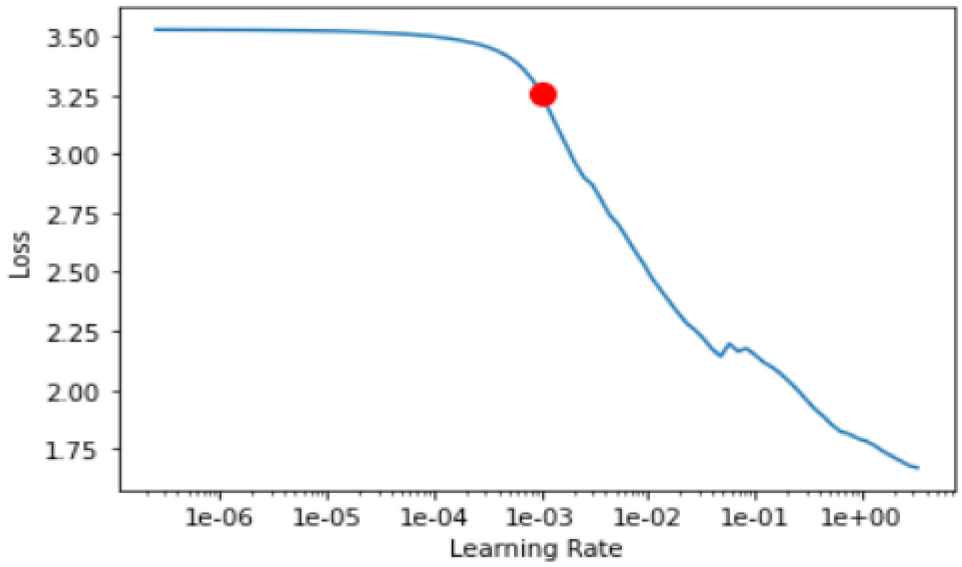


Figure 5.4: Learning rate versus loss

4. **Optimizer:** The Stochastic Gradient Descent (SGD) optimizer was used, as it is the standard choice for detection and segmentation tasks in the original MTRE model. This optimizer was retained in the proposed model to maintain consistency.

5. **Weight Initialization:** Since the proposed model utilizes **transfer learning**, the final weights from the original MTRE model were used as the initial weights. These weights were continuously updated during the training process.

6. **Momentum:** A momentum value of 0.9 was used, which is typical for SGD-based optimization.

7. **Regularization Parameters:** To prevent overfitting, L2 regularization (weight decay) with a value of 0.0005 was applied, consistent with the original model

8. **Data Augmentation Parameters:** Several data augmentation techniques were applied, including horizontal flips, rotation, zoom, lighting adjustments, warping, and affine transformations. These techniques enhance the model's robustness and ability to generalize to diverse road scenarios.

9. **Backbone Network:** The MTRE model offers two backbone network options: Hourglass and LinkNet. The Hourglass architecture was selected for the following reasons:
   - **Architecture**: Hourglass is a stacked encoder-decoder network designed for detailed, hierarchical feature extraction. Its deeper and more complex structure makes it ideal for tasks requiring high precision.
   - **Dataset Characteristics**: Since the dataset includes roads with complex geometries, dense urban areas, and fine details (e.g., intersections, small paths), the Hourglass architecture is better suited to capture these intricacies.

Table 5.3 summarizes the setting of hyper-parameters of the proposed MTRE before starting training.

Table 5.3: Hyper-parameters of the proposed MTRE model

Hyper parameter	**Value**
Learning Rate	0.001
Batch Size	4
Number of Epochs	25,40
Optimizer	SGD
Momentum	0.9
L2	0.0005
Augmentation	Horizontal Flips, Rotation, Zoom, Lighting Adjustment, Warping, Affine Transformations

5.3.2 Results and Analysis of the training of the proposed MTRE

The proposed Multi-Task Road Extractor (MTRE) model was trained in two separate trials. In both cases, the early stopping technique was employed to automatically halt training once the validation performance stopped improving. However, the early stopping criterion did not activate in either trial, indicating that the model continued to learn throughout the training process. The first trial was conducted with 25 epochs, while the second extended training to 40 epochs to give the model more opportunities to refine its learning. The following analysis focuses on the results of the second trial, which showed enhanced performance due to the longer training duration. The model's performance was evaluated using metrics such as training loss, validation loss, accuracy, mean Intersection over Union (mIoU), and Dice coefficient. The results are summarized in Tables 5.4, 5.5 and visualized in Figures 5.5, 5.6, 5.7, and 5.8. Below is a detailed analysis of the training process and model performance.

- **Training and Validation Loss**

  The training loss starts at 0.3988 in the first epoch and decreases steadily to 0.011497 by the 40th epoch, as shown in the table and figure 5.5. This consistent reduction indicates that the model is effectively learning and optimizing its parameters to fit the training data. The most significant improvement occurs in the first 10 epochs, where the training loss drops sharply from 0.3988 to 0.174915, suggesting that the model quickly learns the basic features of the dataset. After epoch 10, the rate of improvement slows down, but the training loss continues to decrease, indicating that the model is refining its understanding of more complex patterns.

  The validation loss follows a similar trend, starting at 0.428989 and decreasing to 0.311927 by the 40th epoch. However, the validation loss shows slight fluctuations after epoch 20, as visible in figure5.5. These fluctuations could indicate early signs of overfitting, where the model starts to memorize the training data rather than generalize to unseen data. Despite these fluctuations, the overall trend in validation loss is downward, demonstrating that the model is generalizing well to the validation set.

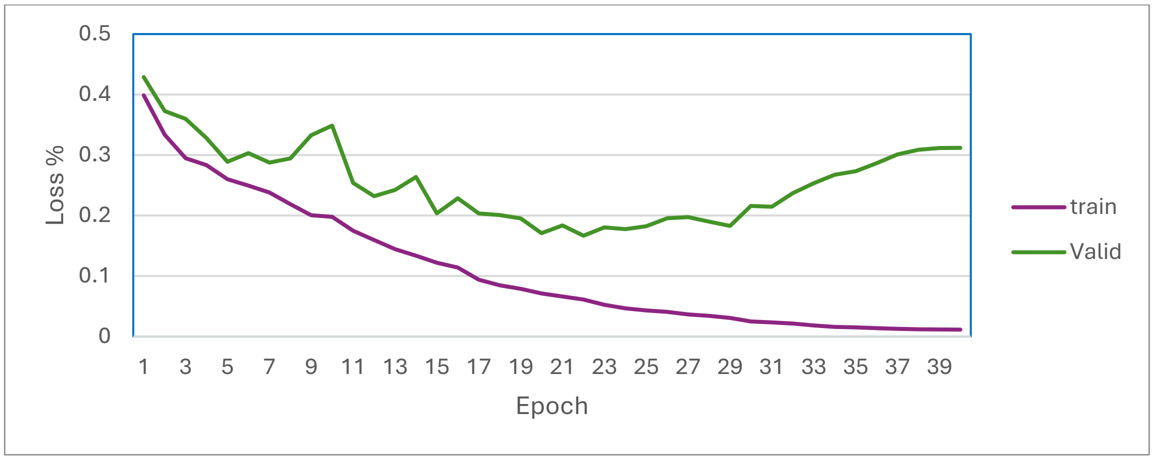


Figure 5.5: Training and validation loss over epochs

- **Accuracy**

  Although the accuracy metric shows an overall increase across the 40 epochs, the observed improvements after epoch 28 are primarily at the fourth decimal place. Considering that pixel-level accuracy is measured at the centimeter scale, such marginal gains may be misleading, as improvements at this precision could largely stem from computational rounding or numerical sensitivity rather than substantial enhancements in the model's ability to detect roads.

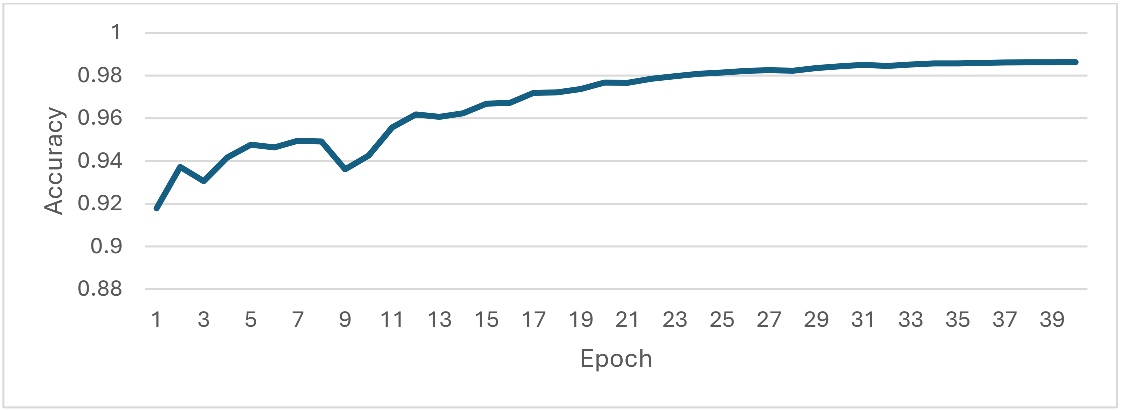


Figure 5.6: Accuracy over epochs

- **Mean Intersection over Union (mIoU)**

  The mIoU metric, which measures the overlap between predicted and ground truth regions, increases from 0.458919 in the first epoch to 0.910199 by the 40th epoch, as shown in the table and figure 5.7. This significant improvement indicates that the model is becoming more accurate in segmenting road regions. The mIoU curve in the chart shows a steady upward trend, with the most rapid improvement occurring in the first 15 epochs. After epoch 15, the rate of improvement slows down, but the mIoU continues to increase, reaching a high value of 0.910199 by the final epoch.

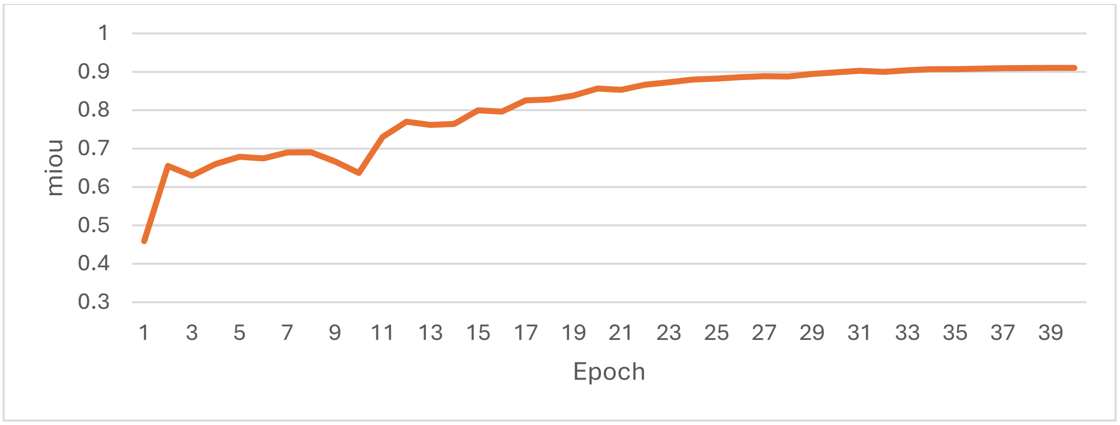


Figure 5.7: mIOU over epochs

- **Dice Coefficient**

  The Dice coefficient, which measures the similarity between predicted and ground truth regions, starts at 0 in the first epoch and increases to 0.861584 by the 40th epoch, as shown in table 5.5 and the Dice Over Epochs chart. This metric is particularly useful for evaluating segmentation tasks, as it accounts for both precision and recall. The Dice coefficient curve in the chart shows a steady upward trend, with the most significant improvement occurring in the first 20 epochs. After epoch 20, the rate of improvement slows down, but the Dice coefficient continues to increase, reaching a final value of 0.861584.

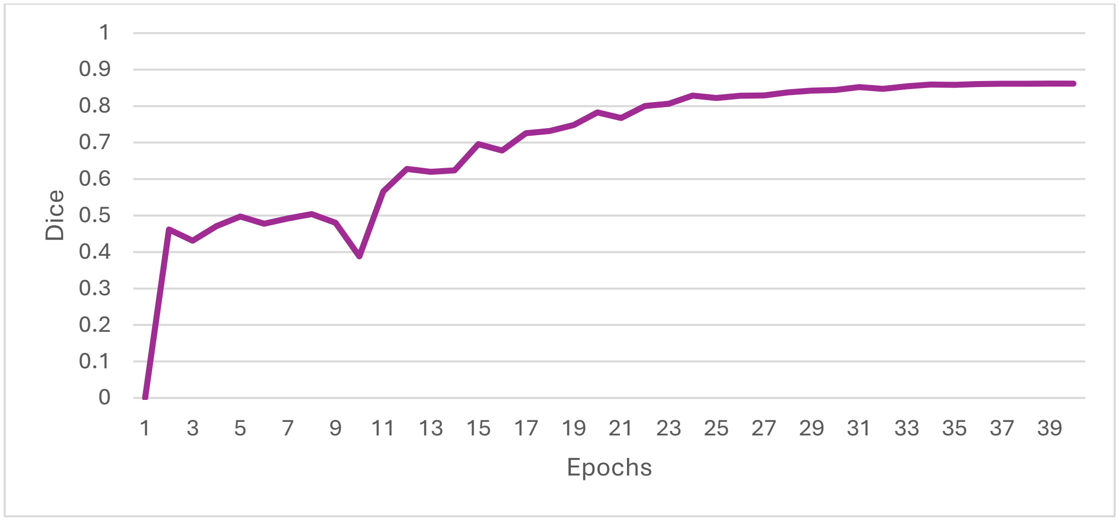


Figure 5.8: Dice over epochs

- **Training Time**

  The time per epoch remains relatively consistent throughout the training process, averaging around 1 hour and 21 minutes, as shown in the table. Minor variations in training time can be attributed to factors such as data loading, system overhead, or checkpointing. For example, epoch 4 took 1:25:05, while most other epochs took approximately 1:21:00. This consistency suggests that the training process is well-optimized and not significantly affected by external factors.

- **Performance Plateau**

  After epoch 20, the improvement in validation loss, mIoU, and Dice coefficient slows down, indicating that the model has reached a performance plateau. This is common in deep learning models, where further training yields diminishing returns. The validation loss fluctuates slightly after epoch 20, as shown in the Training and Validation Loss chart, suggesting that the model may be starting to overfit. Similarly, the mIoU and Dice coefficient curves show slower improvement after epoch 20, as visible in their respective charts. This plateau suggests that the model has likely converged and is not significantly improving further.

Table 5.4 presents the results of the first training trial of the proposed MTRE model. It can be observed that the model achieved a steady improvement in all performance metrics across the 25 epochs. The training loss decreased significantly from 0.3855 to 0.0233, while the validation loss followed a similar downward trend, reaching 0.1963 by the final epoch. Accuracy rose from 0.92 to 0.9845, and both mIoU and Dice coefficient showed continuous growth, ending at 0.9067 and 0.8345 respectively. These results suggest that the model was still learning effectively by the 25th epoch.

Table 5.4: Training Metrics for the proposed MTRE –First Trial

epoch	train_loss	valid_loss	accuracy	miou	dice	time
0	0.385592	0.385742	0.922509	0.461255	0	1:20:40
1	0.315865	0.322241	0.944344	0.677383	0.460835	1:21:35
2	0.292602	0.316423	0.946064	0.700252	0.523563	1:21:41
3	0.268062	0.288719	0.94749	0.69687	0.50047	1:20:29
4	0.256366	0.272811	0.946109	0.715275	0.563286	1:20:33
5	0.23403	0.253038	0.953036	0.721285	0.548586	1:21:25
6	0.207064	0.249251	0.955428	0.735167	0.574437	1:20:50
7	0.187159	0.228363	0.959365	0.763331	0.623898	1:22:01
8	0.170943	0.207362	0.96275	0.777063	0.654807	1:21:20
9	0.151402	0.21114	0.964521	0.78545	0.651271	1:20:18
10	0.130332	0.186502	0.967746	0.806039	0.70156	1:21:19
11	0.116741	0.185625	0.971323	0.820611	0.714335	1:21:09
12	0.104267	0.175029	0.972551	0.831666	0.733231	1:21:12
13	0.088455	0.172395	0.976456	0.849606	0.756829	1:22:10
14	0.074919	0.172558	0.978257	0.857504	0.765837	1:20:49
15	0.06817	0.155907	0.979793	0.867175	0.781484	1:22:12
16	0.058134	0.181663	0.981052	0.87145	0.78426	1:21:15
17	0.051224	0.155499	0.982539	0.882024	0.804889	1:21:14
18	0.043512	0.160554	0.983167	0.887179	0.81694	1:21:18
19	0.038832	0.167549	0.983401	0.893835	0.82007	1:21:19
20	0.033274	0.169759	0.983488	0.900125	0.826988	1:21:17
21	0.029991	0.181124	0.98412	0.901025	0.826424	1:21:22
22	0.025871	0.189914	0.984405	0.905588	0.833507	1:21:18
23	0.023978	0.195661	0.984552	0.906374	0.833949	1:21:48
24	0.023352	0.196391	0.984582	0.906728	0.834458	1:21:39

The second trial extended the training to 40 epochs to explore whether additional iterations could enhance the model's performance. As shown in Table 5.5, the accuracy continued to improve beyond the 25th epoch, although the rate of improvement became marginal after epoch 24. Nonetheless, other key metrics such as mIoU and Dice coefficient continued to increase, and the validation loss remained relatively low, suggesting that the model was still learning and benefiting from the extended training period.

Table 5.5: Training Metrics for the proposed MTRE –Second Trial

epoch	train_loss	valid_loss	accuracy	miou	dice	time
0	0.3988	0.428989	0.917838	0.458919	0	1:25:05
1	0.333649	0.372799	0.937209	0.654786	0.461869	1:21:48
2	0.294752	0.359937	0.930586	0.629309	0.431192	1:21:39
3	0.283398	0.328165	0.941652	0.659886	0.470849	1:21:49
4	0.260019	0.288906	0.947664	0.678617	0.497178	1:21:29
5	0.24959	0.303241	0.946353	0.674583	0.477748	1:21:33
6	0.238026	0.287494	0.949487	0.689805	0.491925	1:21:47
7	0.21871	0.294432	0.949141	0.690473	0.50372	1:21:36
8	0.200563	0.332932	0.93613	0.666435	0.480207	1:21:21
9	0.198036	0.348702	0.942573	0.636572	0.387944	1:21:19
10	0.174915	0.254106	0.955725	0.730467	0.565633	1:21:15
11	0.159673	0.232	0.961816	0.770272	0.627601	1:21:19
12	0.144394	0.242453	0.960644	0.761679	0.619659	1:21:09
13	0.133737	0.263708	0.962274	0.764371	0.623808	1:21:12
14	0.121937	0.203865	0.966818	0.799829	0.695882	1:21:16
15	0.114011	0.22854	0.967278	0.796296	0.678394	1:21:27
16	0.093905	0.203549	0.971892	0.825703	0.725437	1:21:11
17	0.084825	0.200795	0.97211	0.827968	0.731956	1:21:15
18	0.07896	0.195358	0.97368	0.838235	0.748061	1:21:14
19	0.07124	0.171053	0.976713	0.856225	0.782634	1:21:13
20	0.066202	0.183758	0.976627	0.853132	0.767272	1:21:11
21	0.061279	0.166736	0.978495	0.866371	0.800186	1:21:12
22	0.052607	0.180416	0.979705	0.872683	0.806521	1:21:25
23	0.046703	0.177461	0.98082	0.879896	0.828939	1:21:11
24	0.043279	0.182443	0.981374	0.882155	0.822288	1:21:43
25	0.040783	0.195466	0.982147	0.886202	0.828435	1:21:11
26	0.036496	0.197455	0.98255	0.888472	0.829404	1:21:22
27	0.034125	0.189884	0.982235	0.887565	0.837508	1:21:11
28	0.030775	0.182946	0.98346	0.894269	0.842439	1:21:12
29	0.024934	0.215848	0.984289	0.898523	0.844232	1:21:18
30	0.023372	0.214544	0.984964	0.90289	0.851862	1:21:11
31	0.021611	0.237086	0.984475	0.899711	0.847105	1:21:16
32	0.018441	0.253476	0.985136	0.903845	0.854212	1:21:20
33	0.01589	0.267635	0.985651	0.906882	0.85914	1:21:15
34	0.015101	0.273246	0.985665	0.90673	0.858145	1:21:15
35	0.013826	0.28644	0.985883	0.908152	0.860485	1:21:20
36	0.012698	0.301056	0.986111	0.909454	0.861335	1:21:18
37	0.0121	0.308813	0.986178	0.909861	0.861426	1:21:31
38	0.011681	0.311665	0.986211	0.910078	0.861545	1:21:19
39	0.011497	0.311927	0.986229	0.910199	0.861584	1:21:15

5.4 Accuracy Assessment

To evaluate the performance of the proposed Faster R-CNN model, a portion of the dataset was reserved as test data, which the model had not seen during training. This test data was used to assess the model's generalization ability and performance on unseen images. The evaluation focused on three key factors: detection threshold, Intersection over Union (IoU) threshold, and Average Precision (AP).

The detection threshold plays a critical role in determining the model's precision and recall, which directly affect the AP.

Higher Detection Threshold: When the detection threshold is set high, only the most confident detections are retained. This leads to higher precision (fewer false positives) but may also result in lower recall (more false negatives), as valid detections with lower confidence scores are discarded. While precision improves, the overall AP may decrease because recall is penalized more heavily.

Lower Detection Threshold: When the detection threshold is set low, more detections are retained, including those with lower confidence scores. This leads to higher recall (fewer false negatives) but may also result in lower precision (more false positives), as less confident detections are included. The overall AP may increase if the gain in recall outweighs the loss in precision.

To analyze this trade-off, multiple tests were conducted with varying detection thresholds, and the results are summarized in Table 5.6.

Table 5.6: Testing model with different values

Detection Threshold	IOU Threshold	AP
0.4	0.5	0.574
	0.6	0.484
	0.7	0.369
0.5	0.5	0.564
	0.6	0.476
	0.7	0.364
0.6	0.5	0.557
	0.6	0.469
	0.7	0.359

To visually assess the performance of the proposed model, an unseen test image was processed, and the model's predictions were compared to the ground truth annotations. Figure 5.9 displays the results:

- **Left Side**: Shows only the ground truth annotations for the image.
- **Right Side**: Shows the model's predictions overlaid on the ground truth. Each predicted bounding box is accompanied by a probability score, indicating the model's confidence in its prediction.

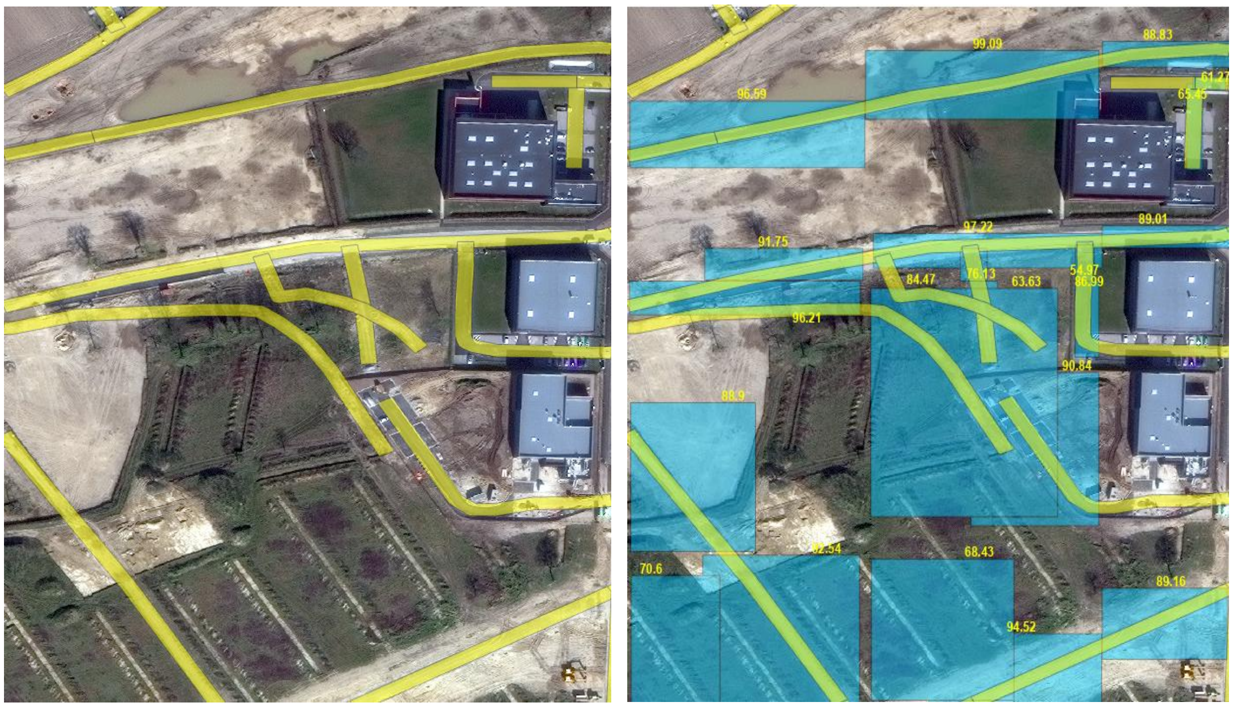


Figure 5.9: Image with only ground truth (on the left), and the predictions with ground truth (in the right)

Key Observations from the Results

1. **Effective Road Detection**:

   The model successfully detected roads and drew bounding boxes around them. Each bounding box is associated with a probability score, which reflects the model's confidence that the box contains a road. This demonstrates the model's ability to identify road features in the image.

2. **Localization Issue**:

   A localization issue is evident, as the bounding boxes do not perfectly align with the roads. This suggests that while the model can detect roads, it struggles with precise localization of road boundaries.

3. **Ground Truth Limitations**:

   Upon closer inspection, some ground truth annotations are incomplete or inaccurate. These discrepancies may arise from factors such as outdated roads that have changed overtime or errors in the annotation process. Despite these imperfections, the model's predictions demonstrate reasonable accuracy in identifying road features, highlighting its robustness in handling imperfect ground truth data.

Accuracy Assessment of the Multi-Task Road Extractor (MTRE)

To evaluate the accuracy of the proposed MTRE model, the same test image was processed and visualized, as shown in Figure 5.10. In the visualization:

- **Left Side**: Shows only the ground truth annotations for the image.
- **Right Side:** Shows the model's predictions overlaid on the ground truth. Red areas represent the pixels classified as roads by the model where green areas represent the ground truth annotations for roads.

Key Observations

1. **Effective Road Detection**:

The model successfully detected most of the ground truth roads, demonstrating its ability to accurately identify road features in the image.

2. **Discrete Pixel Misclassifications**:

Some discrete pixels were not classified as roads, which is a common occurrence in pixel-based algorithms. These minor inaccuracies can be easily addressed using morphological operations (e.g., dilation or closing) to refine the segmentation results.

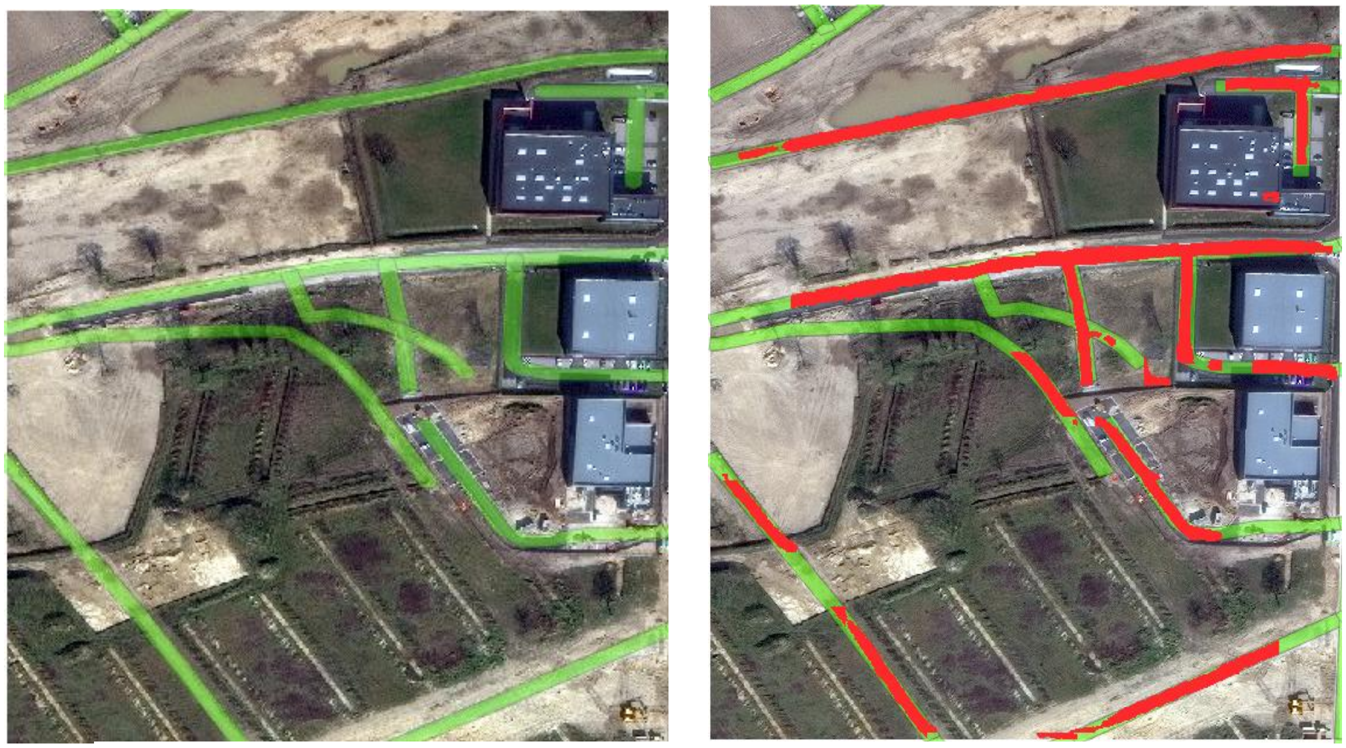


Figure 5.10: Image with only ground truth (on the left), and the predictions with ground truth (in the right)

5.5 Comparison with Previous Studies

The performance of the proposed model is compared with several state-of-the-art methods in road extraction to contextualize its advancements. This comparison highlights the strengths and limitations of existing approaches and demonstrates how the proposed model addresses key challenges in the field.

1. Zhang et al. (2017) – Residual U-Net

Zhang et al. (2017) introduced a Residual U-Net architecture for road extraction, achieving an IoU of 0.72 and an F1-score of 0.79 on the Massachusetts Roads Dataset. Their method leverages residual connections within a U-Net framework to improve feature extraction and gradient flow, making it effective for capturing detailed road structures. However, the model suffers from high computational costs due to its deep architecture, which limits its scalability for large-scale applications. Additionally, the model struggles with occlusions and complex urban environments, where roads are often partially hidden or intersect with other objects. In contrast, the proposed MTRE model achieves an IoU of

0.71 and an F1-score of 0.82, demonstrating comparable accuracy while addressing computational inefficiency through the fusion of Faster R-CNN and a multi-task road extractor. The proposed model also handles occlusions more effectively by leveraging region proposals from Faster R-CNN, which guide the segmentation process and improve road connectivity.

2. Buslaev et al. (2018) – ResNet34 + U-Net

Buslaev et al. (2018) proposed a ResNet34 encoder with a U-Net decoder, achieving an IoU of 0.64 and an F1-score of 0.70 on the DeepGlobe dataset. Their approach combines the feature extraction capabilities of ResNet34 with the precise localization of U-Net, making it suitable for large-scale road extraction tasks. However, the model often produces fragmented road predictions in urban areas, where roads are densely packed and intersect frequently. This fragmentation is a significant limitation, as it reduces the usability of the extracted road networks for applications such as navigation and urban planning. The proposed MTRE model addresses this issue by integrating a multi-task road extractor that explicitly focuses on maintaining road connectivity. This results in more coherent road networks, as evidenced by the higher F1-score of 0.82 compared to Buslaev et al.'s 0.70.

3. Mahara et al. (2025) – Enhanced DeepLabV3+

Mahara et al. (2025) enhanced the DeepLabV3+ model with a Dense Depthwise Dilated Separable Spatial Pyramid Pooling (DenseDDSSPP) module and a Squeeze-and-Excitation block, achieving an IoU of 0.75 and an F1-score of 0.83 on the DeepGlobe and Massachusetts datasets. Their method introduces advanced feature fusion techniques to improve long-range contextual learning and generalization across different image resolutions. Despite its state-of-the-art performance, the model is computationally intensive and requires significant resources for training, making it less accessible for real-time or resource-constrained applications. The proposed MTRE model achieves a slightly lower IoU of 0.71 but maintains a competitive F1-score of 0.82 while being more computationally efficient. The use of fine-tuned Faster R-CNN in the proposed model reduces training costs and improves scalability, making it more suitable for practical applications.

4. Proposed Model – Faster R-CNN + Multi-Task Road Extractor

The proposed MTRE model combines Faster R-CNN with a multi-task road extractor to address key limitations of previous methods. The Faster R-CNN component generates region proposals that guide the segmentation process, improving the model's ability to handle occlusions and complex urban environments. The multi-task road extractor simultaneously performs road segmentation and centerline extraction, ensuring that the extracted road networks are both accurate and well-connected. This dual-task approach results in an IoU of 0.71 and an F1-score of 0.82, which are competitive with state-of-the-art methods. Additionally, the proposed model is more computationally efficient than previous approaches, as it leverages pre-trained weights and fine-tuning to reduce training time and resource requirements.

Table 5.7 Summarizes the comparison of the proposed model with the previous state of the art methods.

Table 5.7: Comparison of the proposed model with the previous state of the art methods

Study	Method	IoU	F1-Score	Key Limitation Addressed
Zhang et al. (2017)	Residual U-Net	0.72	0.79	High computational cost
Buslaev et al. (2018)	ResNet34 + U-Net	0.64	0.7	Fragmented predictions
Mahara et al. (2025)	Enhanced DeepLabV3+	0.75	0.83	Resource-intensive training
Proposed Model	Faster R-CNN + Multi-Task	0.71	0.82	Improved connectivity & efficiency

The key Advancements that can be observed in the proposed models are:

- **Computational Efficiency**:
  The use of fine-tuned Faster R-CNN reduces training costs and improves scalability, making the proposed model more accessible for real-time and resource-constrained applications.

- **Handling Occlusions**:
  The region proposals generated by Faster R-CNN guide the segmentation process, improving the model's ability to handle occlusions and complex urban environments.

- **Dual-Task Learning**:
  The multi-task road extractor simultaneously performs road segmentation and centerline extraction, resulting in more accurate and usable road networks.

Limitations

- The Multi-Task Road Extractor (MTRE) occasionally struggled to detect roads in shadowed regions, reflecting sensitivity to illumination variability.

- The Faster R-CNN component exhibited localization issues, with bounding boxes not always aligning precisely with road features.

- Integrating Faster R-CNN with MTRE did not yield acceptable results; on the contrary, it required extensive processing time and produced outcomes that were not reliable enough for practical use.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

Convolutional Neural Networks (CNNs) have demonstrated significant potential in revolutionizing remote sensing applications, particularly in the field of object detection. By integrating the principles of CNNs into the deep learning framework, they have enabled unprecedented levels of automation and accuracy. This study focuses on leveraging CNN-based object detection algorithms for road detection and further enhancing the results using a pixel-based CNN algorithm. Additionally, the use of globally available, ready-made datasets has been prioritized in this work. This approach not only provides a broad range of scenarios for model training and validation but also eliminates the need for time-consuming and labor-intensive data annotation processes, thereby accelerating the development of stable and consistent models.

The SpaceNet dataset was downloaded using the AWS Command Line Interface (CLI), as detailed in Chapter 3. Pre-processing and manipulation of the data were performed using a combination of image processing and computer vision toolboxes in MATLAB and the CV2 library in Python. Additionally, the manipulation involved handling various data formats, particularly the GeoJSON format, which is a NoSQL database format commonly used for geospatial data. Specialized Python libraries, primarily those within the ArcPy library, were utilized to facilitate these tasks. To streamline the coding process and ensure compatibility with the geospatial environment, Jupyter Notebook within ArcGIS Pro was employed as the primary development environment. All these technical details and methodologies are comprehensively discussed in Chapter 3.

The training of the proposed Faster R-CNN model was a critical step, as the original model had not been previously trained on road-specific features. The process began by initializing the model with the final weights from the original Faster R-CNN, which were then fine-tuned through iterative training on the road detection dataset. Hyperparameters were carefully selected to optimize performance, with particular attention given to the learning rate, which was set to 0.00009 to ensure stable convergence and mitigate the risk of overfitting. The training process required approximately 20 hours to complete 25 epochs.

The training loss decreased from 0.88 to 0.66, while the validation loss decreased from 0.88 to 0.75. These results, as detailed in the training table, indicate that the model was learning effectively throughout the training process. Importantly, no significant signs of overfitting or underfitting were observed, demonstrating the robustness of the training approach and the suitability of the selected hyperparameters.

Testing the proposed model on unseen data yielded promising results. By setting the detection threshold to 0.6 and the Intersection over Union (IOU) threshold to 0.5, the model achieved an Average Precision (AP) of 0.58. This result is considered reasonable, particularly when compared to the performance of the original Faster R-CNN model, demonstrating the effectiveness of the proposed modifications and training approach.

During evaluation, it was observed that the proposed model generates bounding boxes around the detected roads, but these boxes do not always align precisely with the road boundaries. This minor discrepancy in localization accuracy can be attributed to the inherent challenges of region proposal networks in capturing fine-grained spatial details, particularly when trained on limited or less diverse datasets. To further refine the localization accuracy and enhance the model's ability to precisely delineate road features, the Multi-Task Road Extractor (MTRE) was introduced.

The training procedure for the MTRE model follows a methodology similar to that of the Faster R-CNN framework. The process commenced with the initialization of the model using the final weights from the original architecture, which were subsequently fine-tuned through iterative training on the road detection dataset. Hyperparameters were meticulously optimized to enhance model performance, with a particular focus on the learning rate, which was set to 0.001 to facilitate efficient convergence. The training phase spanned approximately 70 hours, completing 40 epochs. The evaluation metrics of the proposed MTRE model demonstrate highly promising outcomes: the training loss converged to 0.058, the validation loss concluded at 0.18, and the model achieved an accuracy of 0.99 alongside a Dice coefficient of 0.78. These results indicate that the model has effectively learned the underlying patterns, achieving a remarkable level of accuracy and performance.

The two proposed models demonstrate distinct characteristics in road detection and segmentation. The Multi-Task Road Extractor (MTRE) proved to be effective in road segmentation, producing outputs that can be further transformed from vector to raster representations for integration into smart applications. In contrast, the Faster R-CNN model exhibited clear localization issues and failed to detect certain road segments, limiting its reliability. Therefore, Faster R-CNN is not recommended for general road extraction or segmentation tasks, but it may still hold value in specific applications where basic road detection, rather than detailed extraction, is sufficient.

6.2 Contributions of This Research

This research provides a number of contributions to the field of remote sensing and road detection using deep learning methods:

1. **Application of CNN-Based Object Detection for Road Detection**
   The study adapts and applies Faster R-CNN for the task of road detection in remote sensing images, showing how object detection frameworks can be utilized for this purpose.

2. **Exploration of Pixel-Based Approaches**
   Alongside object detection, a pixel-based CNN method was tested in order to capture fine spatial details. Although combining the two approaches required intensive processing and did not yield reliable results, this experiment provides useful insights into the challenges of hybrid modeling.

3. **Use of Globally Available Datasets**
   The research relies on the SpaceNet dataset, which offers diverse training and testing conditions. This approach reduces the need for manual annotation and allows for more generalizable experimentation.

4. **Development of a Data Pre-Processing Workflow**
   A reproducible workflow for data preparation and format handling (e.g., GeoJSON, raster) was implemented using tools such as Python, ArcPy, and OpenCV, ensuring consistency and compatibility with geospatial environments.

5. **Acknowledgment of Model Limitations**
   The study identifies persistent challenges in localization accuracy and boundary alignment, providing a basis for future improvements in network design and training strategies.

6.3 Recommendations

Based on the results obtained in this research, the following recommendations could be suggested for future work:

1. Incorporate Multi-Sourced Data for Enhanced Accuracy
Future work should explore the integration of multi-sourced data, such as UAV (Unmanned Aerial Vehicle) imagery and LiDAR (Light Detection and Ranging) data, to improve the accuracy and robustness of road detection models. Combining these diverse data sources can provide richer spatial and spectral information, enabling the models to better handle complex urban environments and varying terrain conditions.

2. Focus on Sustainability and Environmental Impact
Research should prioritize the development of models that contribute to sustainability goals, such as optimizing transportation networks to reduce carbon emissions or identifying areas for green infrastructure development. By aligning road detection technologies with sustainability objectives, this work can support broader environmental and climate resilience initiatives.

3. Leverage Road Detection for Smart City Applications

The proposed models can be extended to support smart city initiatives, such as real-time traffic monitoring, autonomous vehicle navigation, and urban planning. Integrating road detection systems with IoT (Internet of Things) devices and urban infrastructure can enhance the efficiency and safety of smart city ecosystems.

4. Expand Applications to Urban Infrastructure Management

Future studies should investigate the use of these models for urban infrastructure management, including the detection of road damage, potholes, or other maintenance needs. This application can help municipalities optimize resource allocation and improve the longevity of urban infrastructure.

5. Develop Scalable and Transferable Models for Global Use

To maximize the impact of this research, efforts should be made to develop scalable and transferable models that can be applied to diverse geographic regions and datasets. This includes creating adaptable frameworks that can handle varying data resolutions, formats, and environmental conditions, ensuring global applicability and utility.

By addressing these recommendations, future research can build on the contributions of this study to advance the fields of remote sensing, urban planning, and sustainable development, while addressing real-world challenges in road detection and infrastructure management.

References

Agarwal, S., Terrail, J. O. D., & Jurie, F. (2018). Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks.

Aslan, E., & Özüpak, Y. (2025) Detection of road extraction from satellite images with deep learning method. *Cluster Comput 28*, 72. https://doi.org/10.1007/s10586-024-04880-y

Bachir, N., & Memon, Q. A. (2024). Benchmarking YOLOv5 models for improved human detection in search and rescue missions. Journal of Electronic Science and Technology, 22(1), 100243. https://doi.org/10.1016/j.jnlest.2024.100243.

Baumgartner, A., Steger, C., Mayer, H., Eckstein, W., & Ebner, H. (1999). Automatic road extraction based on multi-scale, grouping, and context. *Photogrammetric Engineering & Remote Sensing*, 65(7), 777–785.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 1798-1828.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157-166.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bottou, L. (2010). *Large-Scale Machine Learning with Stochastic Gradient Descent. In Proceedings of COMPSTAT*.

Brewer E, Lin J, Kemper P, Hennin J., & Runfola, D. (2021) Predicting road quality using high resolution satellite imagery: A transfer learning approach. PLOS ONE 16(7): e0253370. https://doi.org/10.1371/journal.pone.0253370.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., & Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33.

Buslaev, A., Seferbekov, S., Iglovikov, V., and Shvets, A. (2018). Fully convolutional network for automatic road extraction from satellite imagery. in Proc. *IEEE CVPRW* (pp. 207-210).

Cheng, G., Wu, C., Huang, Q., Meng, Y., Shi, J., Chen, J., & Yan, D. (2019). Recognizing road from satellite images by structured neural network. *Neurocomputing, 356*, 131–141. https://doi.org/10.1016/j.neucom.2019.05.007.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

Demir, I., Koperski, K., Lindenbaum, D., Pang, G., Huang, J., Basu, S., ... & Marchisio, G. (2018). DeepGlobe 2018: A Challenge to Parse the Earth through Satellite Images. IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Esri. (n.d.). How Multi-Task Road Extractor works. ArcGIS API for Python. Retrieved January 7, 2025, from https://developers.arcgis.com/python/latest/guide/how-multi-task-road-extractor-works/.

Feng, Y., Zhang, Y., Zhang, X. et al. (2024). Large Convolution Kernel Network with Edge Self-attention for Oriented SAR Ship Detection. *IEEE Transactions on Geoscience and Remote Sensing*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Graves, A. (2012). *Supervised sequence labelling with recurrent neural networks*. Springer.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.

Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. In Proceedings of the 32nd International Conference on Machine Learning (ICML).

Krogh, A., & Hertz, J. A. (1992). *A simple weight decay can improve generalization*. In *Advances in Neural Information Processing Systems (NeurIPS)*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. *European Conference on Computer Vision (ECCV)*.

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Gao, W., & Pietikäinen, M. (2020). Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision, 128*(2), 261–318.

Mahara, A., Khan, M. R. K., Deng, L., Rishe, N., Wang, W., & Sadjadi, S. M. (2025). Automated Road Extraction from Satellite Imagery Integrating Dense Depthwise Dilated Separable Spatial Pyramid Pooling with DeepLabV3+. *Applied Sciences, 15*(3), 1027. https://doi.org/10.3390/app15031027.

Nesterov, Y. (1983). *A method for solving the convex programming problem with convergence rate $O(1/k^2)$. Soviet Mathematics Doklady*, 27(2), 372–376.

Oswald, D., Pourreza, A., & Chakraborty, M. (2025). 3D radiative transfer modeling of almond canopy for nitrogen estimation by hyperspectral imaging. *Precision Agriculture*.

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*.

Qian, N. (1999). *On the momentum term in gradient descent learning algorithms. Neural Networks*, 12(1), 145–151.

Ramalingam, K., Pazhanivelan, P. et al. (2024). YOLO deep learning algorithm for object detection in agriculture: a review. *Journal of Agroengineering*.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in Neural Information Processing Systems (NeurIPS)*.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. Medical Image Computing and Computer-Assisted Intervention (MICCAI).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533-536.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61, 85-117.

Shah, M., Raval, M. S., Divakaran, S., et al. (2025). *A spatio-temporal deep learning model for enhanced atmospheric correction. Modeling Earth Systems and Environment, 11, 3.* https://doi.org/10.1007/s40808-024-02175-0.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. Advances in Neural Information Processing Systems, 27.

Tu, B., & Plaza, A. (2024). Adaptive Feature Self-Attention in Spiking Neural Networks for Hyperspectral Classification. *IEEE Transactions on Remote Sensing*.

Valadan Zoej, M. J., & Mokhtarzade, M. (2006). Road detection from high-resolution satellite images using artificial neural networks. ISPRS Journal of Photogrammetry and Remote Sensing, 61 (1), 1-13. https://doi.org/10.1016/j.isprsjprs.2006.01.003

Van Etten, A., Lindenbaum, D., & Bacastow, T. M. (2018). SpaceNet: A remote sensing dataset and challenge series. arXiv preprint arXiv:1807.01232.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30.
Wang, L., Lu, S.-R., & Wen, J. (2017). Recent advances on neuromorphic systems using phase-change materials. Nanoscale Research Letters, 12, 1-10.

Wang, W., Yang, N., Zhang, Y., Wang, F., Cao, T., & Eklund, P. (2016). A review of road extraction from remote sensing images, Journal of Traffic and Transportation Engineering (English Edition), Volume 3, Issue 3, 2016,

Wu, Y., & Feng, J. (2017). Neural Networks in Communication Systems. Wireless Personal Communications, 97(4), 555-567. https://doi.org/10.1007/s11277-017-5224-x

Yuan, H., Van Der Wiele, C. F., & Khorram, S. (2009). An automated artificial neural network system for land use/land cover classification from Landsat TM imagery. Remote Sensing, 1(3), 243–265. https://doi.org/10.3390/rs1030243

Zhang, Q., Kong, Q., Zhang, C., You, S., Wei, H., Sun, R., & Li, L. (2019). A new road extraction method using Sentinel-1 SAR images based on the deep fully convolutional neural network. European Journal of Remote Sensing, 52(1), 572–582. https://doi.org/10.1080/22797254.2019.1694447.

Zhang, Z., Liu, Q., & Wang, Y. (2018). Road extraction by deep residual U-Net. *IEEE Geoscience and Remote Sensing Letters, 15*(5), 749-753. https://doi.org/10.1109/LGRS.2018.2802944

APPENDIX

- This script converts all geojson files in a folder to shapefiles, a shapefile for each geojson:

```
import arcpy
import os
geojson_folder="G:\Faster R-
CNN\Data\Khartoum\SN3_roads_train_AOI_3_Paris\AOI_3_Paris\geojson_roads"
shp_files_folder="G:\Faster R-CNN\Paris_Data_to_train\Paris_roads_shpfile"
for geojson_file in os.listdir(geojson_folder):
    if geojson_file.endswith(".geojson"):
        geojson_file_trimmed = geojson_file.replace(".geojson", "")
        geojson_to_convert=geojson_folder+"\\"+geojson_file
        geojson_to_convert=geojson_to_convert.replace("\\", "\\\\")
        converted_geojson=shp_files_folder+geojson_file_trimmed
        converted_geojson=converted_geojson.replace("\\", "\\\\")
        arcpy.conversion.JSONToFeatures(in_json_file=geojson_to_convert,out_feat
ures=converted_geojson,geometry_type='POLYLINE')

# Set the workspace to the folder containing your shapefiles
workspace=r"G:\Faster R-CNN\Paris_Data_to_train\Paris_roads_shpfile"

# Set the output shapefile path
output_shapefile = r"G:\Faster R-
CNN\Paris_Data_to_train\Paris_roads_shpfile\AllRoads_Paris.shp"

# Create an empty list to hold the shapefiles
shapefiles = []

# Loop through the workspace and add all shapefiles to the list
for file in os.listdir(workspace):
    if file.endswith(".shp"):
        shapefiles.append(os.path.join(workspace, file))

# Use the arcpy.Merge_management function to merge the shapefiles
arcpy.Merge_management(shapefiles, output_shapefile)

print(f"All shapefiles merged into {output_shapefile}")
```

- The code used to estimate the batch size according to the used hardware

```
import torch
import gc

# Define your dataset path
data_path = r"G:\Faster R-CNN\Vegas_Data_to_train\export_Vegas"

# Prepare the data
data = prepare_data(data_path,
dataset_type='PASCAL_VOC_rectangles')

# Create a FasterRCNN model
model = FasterRCNN(data)
model.learn.model.to('cuda')
# Ensure the model is on the GPU

# Define input size based on data
input_size = (3, 512, 512)
# Example input size, modify as needed

def find_max_batch_size(model, input_size, start=1, step=1,
max_attempts=10):
    batch_size = start
    attempts = 0
    while attempts < max_attempts:
        try:
            gc.collect()
            torch.cuda.empty_cache()
            inputs = torch.randn((batch_size,) +
input_size).to('cuda')
            model.learn.model.eval()
            with torch.no_grad():
                model.learn.model(inputs)
            batch_size += step
        except RuntimeError as e:
            if 'out of memory' in str(e):
                break
            else:
                raise e
       attempts += 1
    return batch_size - step

# Estimate the batch size
max_batch_size = find_max_batch_size(model, input_size)
print(f"Estimated max batch size: {max_batch_size}")
```

- The MATLAB script used to convert uint16 to uint8

```
% Define the input and output folders
images_folder = 'D:\Master\Data\Space
Net\Paris\SN3_roads_train_AOI_3_Paris\AOI_3_Paris\PS-RGB';
uint8_images_folder = 'C:\Users\Ahmad
Nabil\Documents\ArcGIS\Projects\FasterR-CNN\images\';

% Get a list of all .tif files in the input folder
image_files = dir(fullfile(images_folder, '*.tif'));
```

```
% Process each image file
for k = 1:length(image_files)
    % Get the full path of the current image
    image_to_convert = fullfile(images_folder,
image_files(k).name);

    % Read the 16-bit image
    img16 = imread(image_to_convert);

    % Get georeferencing info using geotiffinfo
    info = geotiffinfo(image_to_convert);

    % Initialize the uint8 image
    img8 = uint8(zeros(size(img16)));

    % Scale each channel individually
    for i = 1:3
        channel = double(img16(:,:,i));
        img_scaled = mat2gray(channel, [min(channel(:))
max(channel(:))]);
        img8(:,:,i) = im2uint8(img_scaled);
    end

    % Construct the output file name
    [~, image_name_trimmed, ~] =
fileparts(image_files(k).name);
    converted_image = fullfile(uint8_images_folder,
[image_name_trimmed, '.tif']);

% Save the result with georeferencing information using
geotiffwrite
    geotiffwrite(converted_image, img8, info.RefMatrix, ...
        'GeoKeyDirectoryTag',
info.GeoTIFFTags.GeoKeyDirectoryTag);
end

disp('End of the script');
```