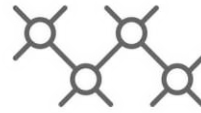




TECHNISCHE
UNIVERSITÄT
WIEN



Institut für
Computertechnik
Institute of
Computer Technology

A MASTER THESIS ON

Simultaneous Detection and Segmentation of Different Objects

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Diplom-Ingenieur

(Equivalent to Master of Science)

in

Embedded Systems (066 504)

by

Sebastian Schuster, BSc

Matr.Nr.: 01608742

Supervisor(s):

Univ.Prof. Dipl.-Ing. Dr.techn. Jantsch Axel

Projektass. Dipl.-Ing. David Breuss

Projektass. Dipl.-Ing. Daniel Schnöll

Vienna, Austria

September 2025



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Object detection and image segmentation are increasingly essential, yet computationally intensive tasks in the field of computer vision, particularly in the domains of edge computing and embedded systems. Traditional approaches employ separate, specialized networks for each task, thereby introducing computational redundancy and resource consumption, posing a challenge for real-life deployment on embedded devices. This thesis investigates the feasibility of three shared encoder strategies to reduce redundancy and quantifies their impact on predictive performance as well as resource consumption:

- **Encoder Swapping:**

Training a network for a single task before freezing and transferring its encoder to another task.

- **Encoder Sharing:**

Using a task-agnostic, frozen pretrained backbone for both tasks.

- **Dual-Head:**

Jointly training a single encoder with separate detection and segmentation heads.

Four backbones (ResNet18, MobileNetV3-Large, EfficientNetV2-S, DenseNet121) were evaluated on a 20-class COCO subset. Dual-head models consistently outperformed other training approaches as well as single-task baselines, yielding performance uplifts of up to 17% mAP and 10% mIoU while reducing overall parameter count by about 45%. Allowing limited adaptation in frozen encoders by unfreezing the bias terms recovered up to 21 percentage points of performance loss. Embedded inference testing on an NVIDIA Jetson AGX Orin confirmed that dual-head models achieve faster joint execution than sequential single-task runs. The results demonstrate that shared encoders with minimal task-specific adaptation or joint optimization can deliver both efficiency and enhanced performance for multi-task vision applications, with bias adaptation offering an alternative when multi-task training is not feasible or possible.

Kurzfassung

Objekterkennung und Bildsegmentierung sind zunehmend essenzielle, jedoch rechenintensive Computervision Aufgaben, besonders in den Bereichen Edge Computing und Embedded Devices. Herkömmliche Systeme nutzen meist separate, für eine Aufgabe konzipierte Netzwerke, was zu redundanten Berechnungen sowie erhöhtem Ressourcenaufwand führt und so den Einsatz auf Embedded Devices erschwert. Diese Arbeit untersucht die Machbarkeit von drei Strategien zur gemeinsamen Nutzung des Encoders, um Redundanz im System zu reduzieren und quantifiziert deren Einfluss auf Performance sowie Ressourcenverbrauch:

- **Encoder Swapping:** Ein Netzwerk wird für eine Aufgabe trainiert, dessen Encoder dann eingefroren und für eine andere Aufgabe eingesetzt.
- **Encoder Sharing:** Ein eingefrorener, zuvor aufgabenunabhängig trainierter Encoder wird für beide Aufgaben verwendet.
- **Dual-Head:** Gemeinsames Training eines einzelnen Encoders mit separaten Objekterkennungs- und Segmentierungsköpfen.

Vier Backbones (ResNet18, MobileNetV3-Large, EfficientNetV2-S, DenseNet121) wurden auf einem 20-Klassen COCO Subset evaluiert. Dual-Head-Modelle performten stets besser als andere Trainingsmethoden sowie Single-Task-Baselines, mit Verbesserungen von bis zu 17% mAP und 10% mIoU bei reduzierter Parameterzahl von etwa 45%. Erlaubt man begrenzte Anpassung der eingefrorenen Encoder mittels Bias-Adaption, kann der Performanceverlust um bis zu 21 Prozentpunkte reduziert werden. Inferenztests auf einem NVIDIA Jetson AGX Orin bestätigen, dass Dual-Head-Modelle schneller auszuführen sind als eine sequenzielle Verarbeitung durch entsprechende Single-Task-Modelle. Die Ergebnisse zeigen, dass gemeinsam genutzte Encoder mit minimaler aufgabenspezifischer Anpassung oder gemeinsamer Optimierung sowohl Effizienz als auch höhere Leistung in Multi-Task-Vision-Anwendungen liefern können, wobei Bias-Adaption eine Alternative darstellt, wenn gemeinsames Training nicht praktikabel oder möglich ist.

Preface

The following tools were utilized to support the writing of this thesis:

Grammarly¹

The written text was refined using Grammarly's AI-based application, which provided spellchecking, grammar correction, and clarity improvements.

¹<https://www.grammarly.com>

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Copyright Statement

I, Sebastian Schuster, BSc, hereby declare that this thesis is my own original work and, to the best of my knowledge and belief, it does not:

- Breach copyright or other intellectual property rights of a third party.
- Contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
- Contain material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.
- Contain substantial portions of third party copyright material, including but not limited to charts, diagrams, graphs, photographs or maps, or in instances where it does, I have obtained permission to use such material and allow it to be made accessible worldwide via the Internet.

Signature: _____

Vienna, Austria, September 2025

Sebastian Schuster, BSc

Contents

Abstract	iii
Kurzfassung	iv
Preface	v
1 Introduction	1
2 Background and Related Work	3
2.1 Convolutional Neural Networks	3
2.1.1 Convolutional Layers	3
2.1.2 Pooling Layers	4
2.1.3 Activation Functions	5
2.1.4 Normalization	6
2.1.5 Loss Function	6
2.1.6 Forward Propagation	7
2.1.7 Backward Propagation	7
2.1.8 Regularization	8
2.1.9 Training	9
2.2 Object Detection	9
2.2.1 Faster R-CNN	10
2.2.2 SSD	11
2.2.3 YOLO	12
2.3 Image Segmentation	13
2.3.1 U-Net	14
2.3.2 DeepLab	15
2.3.3 Bilateral Segmentation Network	15

2.4	Multi-Task Networks	16
2.4.1	Mask R-CNN	17
2.4.2	YOLOv11 Segmentation	17
2.4.3	Non-Standard Approaches to Multi-Task Detection and Segmentation	18
2.5	Transformer-based Models	19
3	Methodology	21
3.1	Core Principle	21
3.2	Dataset	22
3.3	Model Architectures	24
3.3.1	ResNet	24
3.3.2	MobileNetV3	25
3.3.3	DenseNet	26
3.3.4	EfficientNetV2	28
3.3.5	Detection Head	28
3.3.6	Segmentation Head	30
3.4	Encoder Swapping	31
3.5	Encoder Sharing	32
3.6	Dual Head Network	34
3.7	Training	35
3.7.1	Detection Loss	36
3.7.2	Segmentation Loss	36
3.8	Evaluation Metrics	37
3.8.1	Pixelwise Accuracy	37
3.8.2	Intersection over Union	38
3.8.3	AP	39
3.9	Embedded Device	40
4	Experiments	43
4.1	Preliminary Testing	43
4.1.1	Encoder Swapping	43
4.1.2	Encoder Sharing	44
4.2	Baseline	45
4.3	Encoder Swapping	47
4.4	Encoder Sharing	49

<i>Contents</i>	ix
4.5 Dual Head	51
4.6 Embedded Device	52
5 Analysis	55
5.1 Backbone Performance	55
5.2 Approach Comparison	56
5.2.1 Bias Term Adaptation	58
5.2.2 Resource Utilization	59
6 Conclusion and Future Work	61
6.1 Conclusion	61
6.2 Future Work	62
6.3 Bias Adaptation and Adjustment Strategies	62
6.4 Multi-Task Optimization and Adaptive Loss Balancing	63
6.5 Backbone Properties	63
6.6 Task Head Design	63
Bibliography	65
A Appendix	75

List of Tables

3.1	The selected 20 class subset corresponding to the Pascal VOC dataset.	23
3.2	COCO dataset splits and their image counts.	24
3.3	Parameters for preliminary detection and segmentation training.	32
3.4	Training parameters for (un)supervised pretraining and preliminary classification tasks.	34
3.5	Training parameters for networks with four different backbones.	35
4.1	Performance of models with and without swapped encoders.	44
4.2	Classification accuracy of ResNet8 on CIFAR-10 under different pretraining and architectural variants.	44
4.3	Performance of pretrained models with and without frozen backbone weights and biases. Relative change is measured with respect to the normal, fully trainable backbone.	45
4.4	Baseline segmentation performance on the COCO dataset.	46
4.5	Baseline detection performance on the COCO dataset.	46
4.6	Segmentation performance using frozen detection backbones on the COCO dataset (swap-A). Relative change compared to the baseline in %.	47
4.7	Detection performance using frozen segmentation backbones on the COCO dataset (swap-A). Relative change compared to the baseline in %.	47
4.8	Segmentation performance using frozen detection backbones and unfrozen bias terms on the COCO dataset (swap-B). Relative change compared to the baseline in %.	48
4.9	Detection performance using frozen segmentation backbones and unfrozen bias terms on the COCO dataset (swap-B). Relative change compared to the baseline in %.	48
4.10	Segmentation performance using pretrained frozen backbones on the COCO dataset (share-A). Relative change compared to the baseline in %.	49
4.11	Detection performance using pretrained frozen backbones on the COCO dataset (share-A). Relative change compared to the baseline in %.	50

4.12	Segmentation performance using pretrained frozen backbones and unfrozen bias terms on the COCO dataset (share-B). Relative change compared to the baseline in %	50
4.13	Detection performance using pretrained frozen backbones and unfrozen bias terms on the COCO dataset (share-B). Relative change compared to the baseline in %	51
4.14	Segmentation performance of the dual head models on the COCO dataset. Relative change compared to the baseline in %	51
4.15	Detection performance using pretrained frozen backbones on the COCO dataset (share-A). Relative change compared to the baseline in %	52
4.16	Inference times in ms on an NVIDIA Orin, FP32 mode.	52
4.17	Inference times in ms on an NVIDIA Orin, INT8 mode.	52
5.1	Model parameters in millions for all configurations.	59
A.1	Comprehensive results for all backbone variations and experimental configurations. . .	76
A.2	Timing results for all backbones and model variations.	76

List of Figures

2.1	Filtering using a 3×3 kernel in a convolutional layer.	4
2.2	2×2 Maximum pooling and average pooling, , stride = 2.	5
2.3	Activation functions	5
2.4	Example of a detection prediction	10
2.5	R-CNN architecture variants	11
2.6	SSD framework illustration	12
2.7	YOLO detection process	13
2.8	Example of a segmentation prediction	13
2.9	UNet architecture	14
2.10	ASPP module structure	15
2.11	BiSeNet model structure	16
2.12	Mask R-CNN architecture	17
3.1	Example of the ground truth, bounding boxes, and segmentation masks provided by the COCO dataset	22
3.2	Data augmentation techniques applied to the ground truth of Figure 3.1.	23
3.3	ResNet basic block.	25
3.4	ResNet18 encoder structure in PyTorch. Input 320x320 pixels.	25
3.5	MobileNetV3 bottleneck block.	26
3.6	MobilenetV3-Large encoder structure in PyTorch. Input 320x320 pixels.	26
3.7	3-layer DenseBlock	27
3.8	DenseNet121 encoder structure in PyTorch. Input 320x320 pixels.	27
3.9	Fused-MBConv block.	28
3.10	EfficientNetV2-S encoder structure in PyTorch. Input 320x320 pixels.	29
3.11	SSDLite detection head	30
3.12	LR-ASPP detection head	30

3.13	Visualization of the encoder swap approach.	31
3.14	Visualization of the encoder sharing approach.	33
3.15	Visualization of the dual head network.	34
3.16	Pixelwise accuracy	38
3.17	IoU	38
3.18	NVIDIA Jetson AGX Orin module	40
4.1	Baseline comparison, values normalized to the maximum.	46
5.1	Backbone comparison for all approaches.	55
5.2	Scatter plot of inference time vs. model parameters for dual head models.	56
5.3	Approach comparison for all backbones. Relative change compared to the baseline in %	57
5.4	Performance improvements in percentage points between the A and B approaches for both Swap and Switch methods.	58
5.5	Reduction in parameter count going from separate networks to a single shared encoder model.	59

Acronyms

AP Average Precision. viii, 39, 40

ASPP Atrous Spatial Pyramid Pooling. 15

CNN Convolutional Neural Network. 3, 4, 5, 6, 7, 8, 9, 10, 14, 19

COCO Common Objects in Context. xi, xii, 22, 23, 24, 40, 43, 45, 46, 47, 48, 49, 50, 51, 52

IoU Intersection over Union. xiv, 38, 39, 40, 48, 49, 55

LLM Large Language Model. 58

LR-ASPP Low-Resolution Atrous Spatial Pyramid Pooling. 30, 36, 45

mAP mean Average Precision. 39, 40, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 55, 61

MBCConv Mobile Inverted Bottleneck Convolution. 28

mIoU mean Intersection over Union. 39, 43, 44, 45, 46, 47, 48, 49, 50, 51, 55, 57

ONNX Open Neural Network Exchange. 40, 52

Pascal VOC Pascal Visual Object Classes. xi, 22, 23, 24, 32, 35, 45

R-CNN Regions with CNN features. viii, xiii, 10, 11, 17, 18

ReLU Rectified Linear Unit. 5, 6, 24, 27, 31

ResNet Residual Network. viii, xi, 24, 25, 33, 44, 46, 47, 48, 49, 50, 51, 52

RoI Region of Interest. 11, 17

SGD Stochastic Gradient Descent. 7, 34

SPP Spatial Pyramid Pooling. 4

SSD Single Shot MultiBox Detector. vii, xiii, 11, 12, 28, 29, 30, 32, 36, 43, 44, 45

YOLO You Only Look Once. vii, viii, xiii, 12, 13, 17, 18

Chapter 1

Introduction

Computer vision has emerged as one of the most transformative areas of artificial intelligence, with applications in a wide variety of fields, including autonomous driving [1, 2], surveillance systems [3], medical imaging [4, 5], and robotics [6]. Among the fundamental tasks in computer vision, object detection and image segmentation represent two of the most critical and complementary capabilities to both identify and localize objects within images, while also providing pixel-level characterization of these objects. Together, these tasks enable machines to achieve comprehensive scene understanding and are increasingly getting closer to human-level perception.

The growing demand for real-time computer vision applications, particularly in resource-constrained environments such as mobile devices, autonomous vehicles, and other embedded systems, has emphasized the need for efficient neural network architectures. Traditional approaches typically employ separate, specialized networks for each task, one optimized for object detection and another for image segmentation. While this strategy ensures task-specific optimization, it introduces significant computational overhead and resource consumption, as input images must be processed independently by multiple distinct backbone networks. This computational redundancy poses substantial challenges for deployment in real-world applications where processing power, memory, energy consumption, and physical space can be critical constraints. The proliferation of edge computing applications, where inference must occur locally on resource-limited hardware, further amplifies these challenges. Moreover, the maintenance and deployment complexity of multiple specialized models increases system overhead and reduces operational efficiency.

Recent advances in multi-task learning and shared representation learning suggest that many computer vision tasks can benefit from shared feature extraction, as they often rely on similar low-level visual

patterns and hierarchical representations. This observation motivates the exploration of architectures that can efficiently perform multiple related tasks while sharing computational resources, potentially reducing both the computational burden and the complexity of deployment.

To address these challenges, this thesis proposes multiple approaches for designing efficient shared encoder architectures. The aim is to investigate the transfer performance of the proposed shared encoder networks and to create models that are capable of simultaneously performing object detection and image segmentation without compromising accuracy, while reducing computational overhead. Specifically, this work investigates whether feature extraction components can be effectively shared between these tasks in embedded and resource-constrained environments.

The remainder of this thesis is structured as follows:

- **Background and Related Work** provides essential theoretical foundations, reviewing convolutional neural networks, object detection architectures, image segmentation methods, and multi-task learning approaches.
- **Methodology** details the experimental design, including dataset selection, network architectures, training procedures, and evaluation methods.
- **Experiments** presents comprehensive experimental results across all proposed approaches and backbone architectures. Results are systematically analyzed to identify optimal configurations and understand performance trade-offs.
- **Analysis** contextualizes the experimental results, providing comparative evaluations of backbones and proposed approaches, examining trade-offs in performance, efficiency, and adaptation across different scenarios.
- **Conclusion** synthesizes key findings, discusses practical implications, identifies limitations of the current work, and proposes directions for future research.

Chapter 2

Background and Related Work

This chapter presents the foundational concepts of machine learning, focusing on Convolutional Neural Networks (CNNs), detailing their principles and applications in object detection and image segmentation. It establishes the theoretical basis necessary to understand the proposed methodologies and situates them within the context of related research.

2.1 Convolutional Neural Networks

CNNs constitute a specialized class of feedforward neural networks that have revolutionized the fields of computer vision and image processing. Traditional fully connected networks rely on the Multilayer Perceptron (MLP) [7] as their building block, which employs dense matrix multiplications between layers. In CNNs, these dense connections are replaced with convolution operations as their core computational unit [8]. This substitution allows CNNs to process data with a grid-like topology, such as images, by exploiting spatial hierarchies and local patterns through architectural components like convolutional and pooling layers. As a result, CNNs scale effectively to large images and video sequences, making them exceptionally powerful for complex, high-dimensional visual tasks.

2.1.1 Convolutional Layers

Convolutional layers are the foundational components of CNNs. Each convolutional layer consists of multiple small, trainable filters (kernels) that independently traverse the input's spatial dimensions via a sliding-window operation. At each spatial position, a filter computes a dot product between its weights and the corresponding small patch of the input, known as the receptive field. This operation generates a two-dimensional output called a feature map, which encodes the presence and spatial locations of specific patterns detected by each filter, as shown in Figure 2.1.

The parameters of each filter are shared across the entire input, meaning that the same set of weights

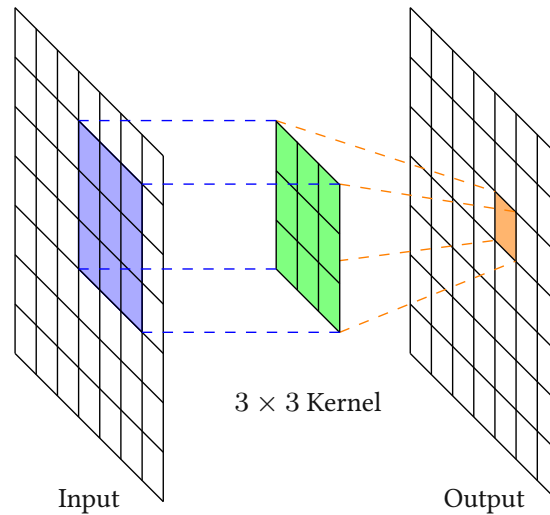


Figure 2.1: Filtering using a 3×3 kernel in a convolutional layer.

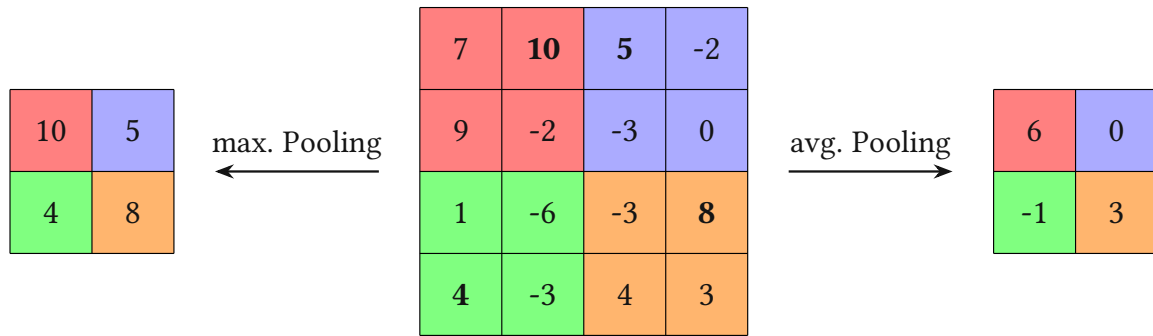
is applied at every spatial location. This property, known as parameter sharing, drastically reduces the number of trainable parameters compared to fully connected layers, ensuring the network is equivariant to translations. This means a pattern can be detected regardless of its position within the input. Each neuron is connected only to a localized region of the preceding layer, its receptive field, rather than the entire input. The output of a convolutional layer thus consists of a stack of feature maps, each corresponding to one filter, which are subsequently processed by deeper layers to recognize increasingly complex and abstract data representations.

2.1.2 Pooling Layers

Pooling layers serve as downsampling operations that reduce the spatial dimensions of feature maps while retaining important information. These layers provide several critical functions: computational efficiency, translation invariance, and hierarchical feature extraction. Two main pooling methods are commonly used in Convolutional Neural Networks:

- **Maximum Pooling:** This method selects the maximum value from each region covered by the pooling window. By retaining only the most prominent features in each region, max pooling emphasizes the strongest activations, such as edges or key textures, within the feature map.
- **Average Pooling:** Here, the mean value of the elements within the pooling window is calculated, smoothing the feature map and typically producing more generalized representations by combining values.

Modern CNN architectures leverage the specific advantages of different pooling methods and strategically select which method to use at various points in the network. More sophisticated pooling strategies, such as fractional max pooling [9], Spatial Pyramid Pooling (SPP) [10], or T-Max-Avg pooling [11],

Figure 2.2: 2×2 Maximum pooling and average pooling, $\text{stride} = 2$.

have emerged to preserve essential features more effectively, adapt to specific data, and enhance model robustness.

2.1.3 Activation Functions

Activation functions are essential components in neural networks, including CNNs, that introduce non-linearity into the networks, enabling them to learn complex patterns and relationships. After each convolutional or fully connected operation, the activation function is applied to the neuron's output to transform it based on its value, before passing it to the next layer. Without activation functions, the network would just perform a series of linear operations, no matter how many layers it contains, severely limiting a network's capability to model complex patterns and effectively reducing the entire network to a single linear transformation of the input. Two commonly used activation functions are the Sigmoid function, defined as $\sigma(x) = \frac{1}{1+e^{-x}}$, and the Rectified Linear Unit (ReLU) function [12], shown in Figure 2.3. Both Sigmoid and ReLU have their limitations: Sigmoid suffers from vanishing gradients

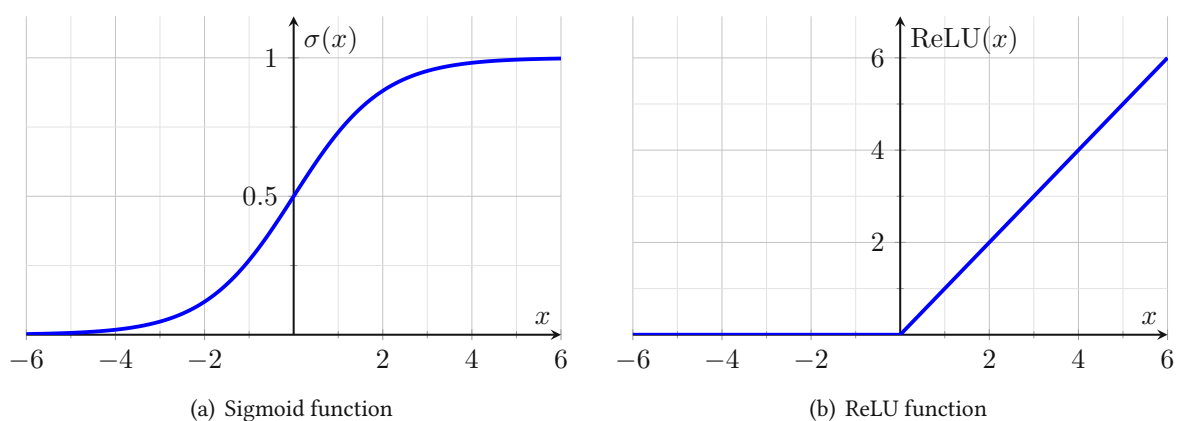


Figure 2.3: Activation functions

in saturated regions, the vanishing gradient problem [13], while ReLU can lead to inactive neurons, also known as the dying ReLU problem [14]. To address these and other challenges, researchers have proposed a multitude of specialized activation functions tailored for various tasks and architectures,

including Leaky ReLU [15], Parametric Rectified Linear Unit (PReLU) [16], Exponential Linear Unit (ELU) [17], Gaussian Error Linear Unit (GELU) [18], or the swish function [19]. These alternatives introduce mechanisms to keep gradients flowing or add flexibility to the neuron's response. Yet, the ReLU function remains the default choice in most modern convolutional neural networks due to its simplicity, performance, and robust empirical results.

2.1.4 Normalization

Normalization techniques are crucial in neural networks, including CNNs, for improving training stability and accelerating convergence. Following a convolutional or fully connected operation, normalization methods adjust the activations to have consistent statistical properties, such as zero mean and unit variance, before passing them to the next layer. Without normalization, training deep networks can be slow and unstable, often requiring careful initialization and small learning rates. One of the most widely used normalization methods is Batch Normalization [20], which normalizes the input of each batch and also introduces learned scaling and shifting parameters, thereby often improving generalization and enabling the use of higher learning rates.

Other notable approaches include Layer Normalization [21], Instance Normalization [22], and Group Normalization [23], each of which is adapted to specific architectures and tasks. These techniques help mitigate issues such as vanishing or exploding gradients and improve generalization, making normalization a standard practice in state-of-the-art convolutional neural networks.

2.1.5 Loss Function

The loss function is a fundamental component in training CNNs, used to quantitatively measure the discrepancy between the network's predictions and the actual target values, the ground truth. The choice of loss function is closely tied to the nature of the task, whether it is classification, regression, or more complex objectives such as object detection and segmentation, since each requires task-specific ground truth data and specialized handling during loss computation. A simple and widely used loss function is the Mean Squared Error (MSE). It often finds application in regression tasks and quantifies the difference between model predictions and actual target values by averaging the squares of the errors for each data point:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (2.1)$$

where y_i is the ground truth value, \hat{y}_i is the predicted value, and n is the number of data points.

A detailed description of the loss functions used and their relevance to the network architectures used

in this thesis will be presented in Chapter 3.

2.1.6 Forward Propagation

Forward propagation is the process by which input data is passed sequentially through the layers of a CNN to produce an output prediction. Beginning at the input layer, data such as an image tensor is supplied to the network, with each subsequent layer applying linear and/or nonlinear operations that progressively transform the data. This series of transformations results in intermediate representations that encode increasingly complex features of the input. Ultimately, forward propagation yields a high-level representation, often as a feature map, which serves as the basis for the network's final output. Mathematically, for each neuron in a layer, the forward propagation step to calculate its output or activation a can be expressed as:

$$a = f \left(\sum_i W_i c_i + b \right) \quad (2.2)$$

where c_i are the outputs from the previous layer, W_i are the weights connecting neuron i in the previous layer to this neuron, b is the bias for this neuron, and f is the activation function.

2.1.7 Backward Propagation

Backpropagation, also known as backward propagation, is the process by which the gradient of the loss function with respect to each model parameter is computed and propagated backward through the layers of a CNN. After forward propagation and loss computation, backward propagation applies the chain rule to efficiently determine how each weight and bias in the network contributed to the overall error, where the loss is calculated using the ground truth. Starting from the output layer, the gradients are sequentially computed for each neuron in each layer, enabling systematic updates of parameters to minimize the loss. This mechanism allows the network to learn by adjusting weights in the direction that reduces prediction error. For each neuron in a layer, the gradient of the loss function L with respect to a weight w_i is given by:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a} \cdot \frac{\partial f}{\partial z}(z) \cdot c_i \quad \text{with} \quad z = \sum_i w_i c_i + b \quad \text{and} \quad a = f(z) \quad (2.3)$$

where a is the neuron's output, z is the pre-activation input, f' is the derivative of the activation function, and c_i is the output from the previous layer. The computed gradients are then used to update the weights and biases, typically via an optimization algorithm such as Stochastic Gradient Descent (SGD) [24] or Adam [25].

2.1.8 Regularization

Regularization refers to a set of techniques designed to improve the generalization of CNNs by preventing overfitting, where a model learns to perform well on training data but fails to generalize to new, unknown data. Overfitting often occurs when only a limited or noisy dataset is available, causing the network to memorize the training data rather than truly learn the underlying data patterns. Several regularization strategies are commonly employed in CNNs:

- **L1 Regularization:** Also called Least Absolute Shrinkage and Selection Operator (LASSO), L1 regularization adds a penalty proportional to the sum of the absolute values of all network weights to the loss function [26]. This penalty encourages many weights to become very small and often close to zero during training, resulting in a sparse set of active weights or features, which helps reduce overfitting. The total loss with L1 regularization is given by:

$$L_{\text{total}} = L_{\text{original}} + \lambda \sum_{i=1}^N |w_i| \quad (2.4)$$

with λ denoting the regularization strength, N the total number of weights, and w_i the individual weights of the network.

- **L2 Regularization:** L2 regularization, also known as weight decay or ridge regression [27], adds a penalty proportional to the sum of the squared values of the model's weights to the loss function. This penalty suppresses large weights, encouraging the model to distribute importance across more features and leading to a weight distribution concentrated around zero. While L2 regularization reduces the magnitude of weights, it does not inherently prevent weights from reaching zero.

$$L_{\text{total}} = L_{\text{original}} + \lambda \sum_{i=1}^N w_i^2 \quad (2.5)$$

- **Dropout:** Dropout is a stochastic regularization technique in which, during training, a randomly selected fraction of neuron outputs are set to zero at each update step [28], thus promoting the learning of robust, redundant feature representations.
- **Data Augmentation:** While not a model-internal modification, data augmentation acts as a form of regularization by artificially increasing the training set size through dataset transformations (e.g., rotations, translations, flips, scaling, color shifts). This exposes the model to a broader distribution of inputs, improving its ability to generalize. The data augmentation techniques used in this work are detailed in section 3.2.
- **Early Stopping:** Early stopping monitors the model's performance on a validation dataset and halts training when validation performance degrades, preventing the network from overfitting

the training set.

In modern CNN architectures, combinations of these and other regularization methods are commonly adopted to improve prediction performance and generalization. The choice and configuration of regularization strategies should be tailored to the specific problem, dataset size, and model complexity.

2.1.9 Training

Training a CNN involves the coordinated interplay of model components, loss functions, and data processing techniques, to optimize the model's parameters for a given task. During training, batches of input data are iteratively passed through the network in the forward propagation phase to produce predictions. The chosen loss function quantifies the discrepancy between these predictions and the ground truth. Through backward propagation, gradients of the loss with respect to all network parameters are computed and propagated backward, enabling systematic updates to the weights and biases. Optimization algorithms leverage these gradients to update parameters in a way that aims to minimize the loss. Regularization techniques are intertwined within the training loop to control model complexity and improve generalization to unseen data. Proper combination and configuration of these methods are crucial for robust model performance: the model must learn sufficiently complex representations to solve the task while avoiding overfitting or poor convergence. The overall training procedure typically consists of multiple passes through the entire dataset, referred to as epochs, to optimize the model parameters for predictive accuracy and generalization.

2.2 Object Detection

Object detection in computer vision involves the simultaneous identification and localization of one or multiple objects in an image or video frame. Unlike image classification, which assigns a single label to an entire image, object detection must potentially manage multiple objects of varying shapes, sizes, and types and determine their locations within a single scene. This objective is commonly decomposed into two subtasks: localization, which determines the precise location and spatial extent of each object, widely represented by bounding boxes describing the objects' outlines, and classification, which assigns an object category label to each detected instance as well as the confidence in that prediction, as shown in Figure 2.4.

While pre-CNN detection methods like Scale-Invariant Feature Transform (SIFT) [30], Viola Jones Detector [31], Histogram of Oriented Gradients (HOG) [32] or Deformable Part-Based Model (DPM) [33] mainly rely on hand crafted features and masks, local descriptors and rigid templates, these techniques

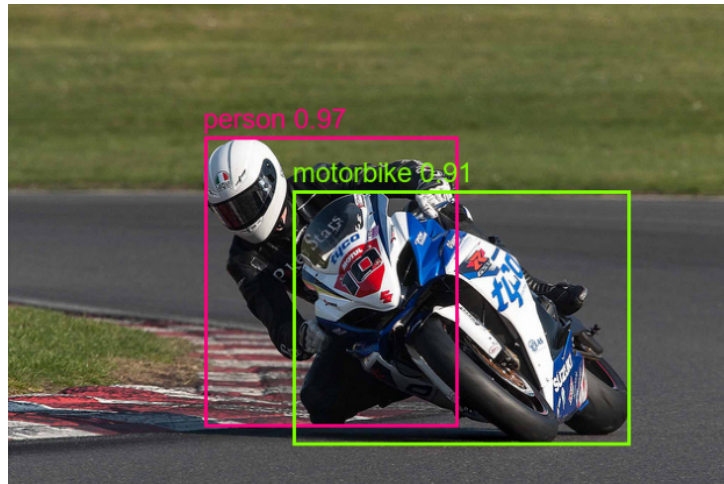


Figure 2.4: Example of a detection prediction.
COCO image id 116712 with superimposed bounding boxes, labels, and confidence score [29].

require significant task and dataset specific knowledge and expertise, limiting generalization and reusability. Leveraging the learning process of CNNs enables detection networks to automatically extract and learn hierarchical feature representations directly from raw image data, eliminating the need for manual feature engineering. This not only streamlines the design process but also yields more robust detection performance across a broader range of objects and challenging scenes.

The following section introduces some CNN-based state-of-the-art object detection models, briefly describing their structure and respective approaches to object detection.

2.2.1 Faster R-CNN

Originally proposed as Regions with CNN features (R-CNN) [34], this network employs a CNN to extract feature representations from a set of region proposals generated for each input image, as shown in Figure 2.5(a). R-CNN relies on external algorithms such as Selective Search [35] to generate approximately 2000 candidate object regions per image. The CNN then extracts features for each proposal, which are then computed independently and classified using external classifiers. Although R-CNN significantly outperforms earlier non-CNN approaches in detection accuracy, its pipeline is computationally inefficient due to the redundant and independent processing of a large number of highly overlapping regions.

The Fast Region-based Convolutional Network (Fast R-CNN) [36], depicted in Figure 2.5(b) introduces several key modifications to improve the training and testing speed as well as the detection accuracy over R-CNN. By passing the entire image through the CNN only once, Fast R-CNN eliminates redun-

dent feature extraction and efficiently shares computation across all region proposals. It also introduces Region of Interest (RoI) pooling to extract fixed-size feature representations for each region proposal directly from the shared feature map. As a result, all region proposals can be processed in parallel within a single network pass. Additionally, Fast R-CNN unifies classification and bounding box regression into a single network, streamlining the detection pipeline.

The Faster R-CNN [37], illustrated in Figure 2.5(c), further advances this framework by replacing the external selective search algorithm with a fully integrated, trainable Region Proposal Network (RPN). The RPN operates directly on the shared convolutional feature map to generate a fixed set of region proposals, reducing proposal computation time without sacrificing detection accuracy. This approach allows region proposal generation and object detection to be performed jointly and end-to-end within a single neural network. By unifying proposal generation with classification and regression, Faster R-CNN achieves significant improvements in both speed and accuracy over its predecessors.

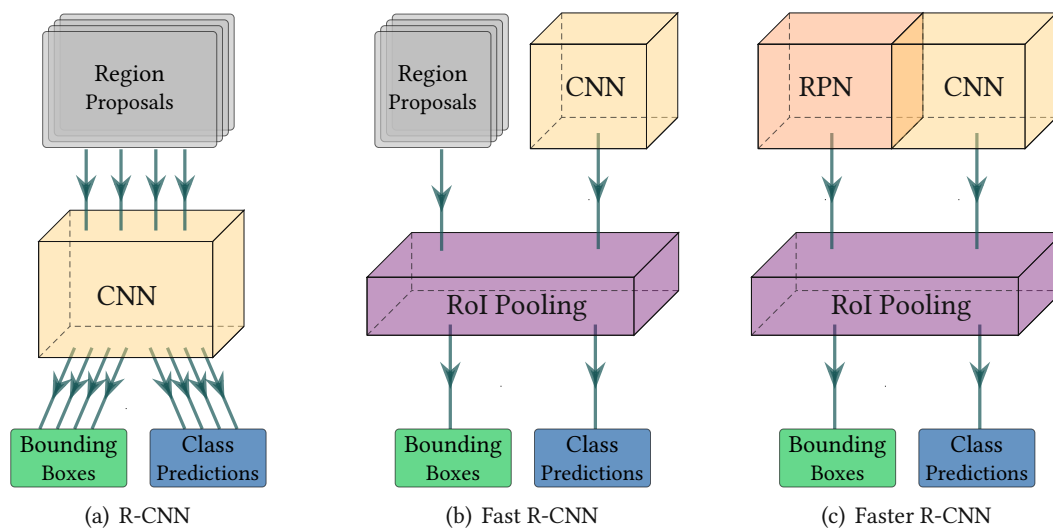


Figure 2.5: R-CNN architecture variants

Other advanced R-CNN variants, such as Cascade R-CNN [38], aim to enhance detection performance through refined architectures and improved training strategies.

2.2.2 Single Shot MultiBox Detector (SSD)

The SSD [39] introduces a fundamentally different approach to object detection compared to the R-CNN family, by eliminating the need for region proposal generation by performing object localization and classification in a single forward pass of the network. This is achieved by applying a set of convolutional filters to multiple feature maps at different scales, enabling the detection of objects of various

sizes directly and efficiently. SSD consists of a backbone network to extract features, followed by multi-scale blocks that reduce the feature maps' scale, effectively increasing its receptive field of the input image. At each cell of these feature maps, SSD defines a set of default bounding boxes of different scales and aspect ratios, which act as fixed reference boxes, as illustrated by Figure 2.6. For every default box, the network predicts both class scores (conf) and offsets that adjust the box to better fit the object's shape and position (loc), highlighted in Figure 2.6c. These default boxes tile the image across multiple feature maps, allowing SSD to cover a wide range of possible object sizes and shapes without a separate region proposal stage.

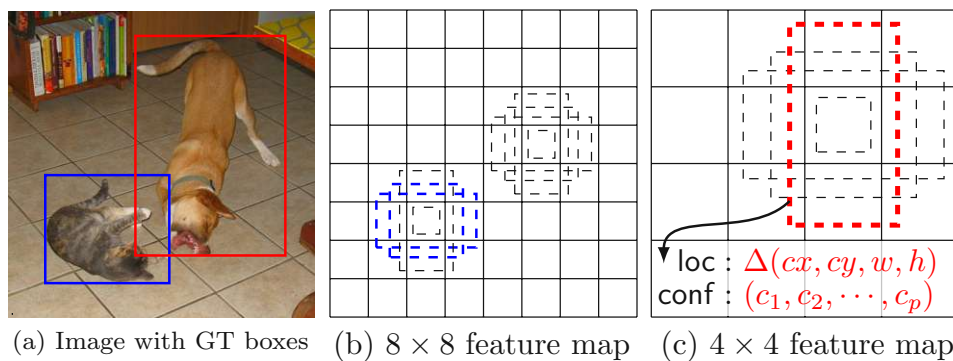


Figure 2.6: SSD framework illustration [39]

By combining predictions from multiple resolutions, SSD achieves a balance between speed and accuracy, making it well-suited for real-time applications while still delivering competitive detection performance across varied object sizes.

2.2.3 You Only Look Once (YOLO)

Like SSD, the YOLO [40] architecture also approaches object detection as a single-stage process, performing localization and classification in one forward pass through the network. The input image is segmented into an $S \times S$ grid, with each grid cell being responsible for all objects whose center lies within the cell. YOLO predicts a fixed number of B bounding boxes for each grid cell, with each box described by its coordinates, confidence score, and class probabilities C , encoding the predictions as a $S \times S \times (B \cdot 5 + C)$ tensor. The detection process is illustrated in Figure 2.7

In addition to the original YOLO model, many new versions have been released to improve object detection performance, address limitations, and expand capabilities. After YOLOv2 [41] and YOLOv3 [42], later versions such as YOLOv4 [43] onward have been developed by a diverse group of researchers and organizations [44–50]. At the time of writing, the latest version is YOLOv12 [51]

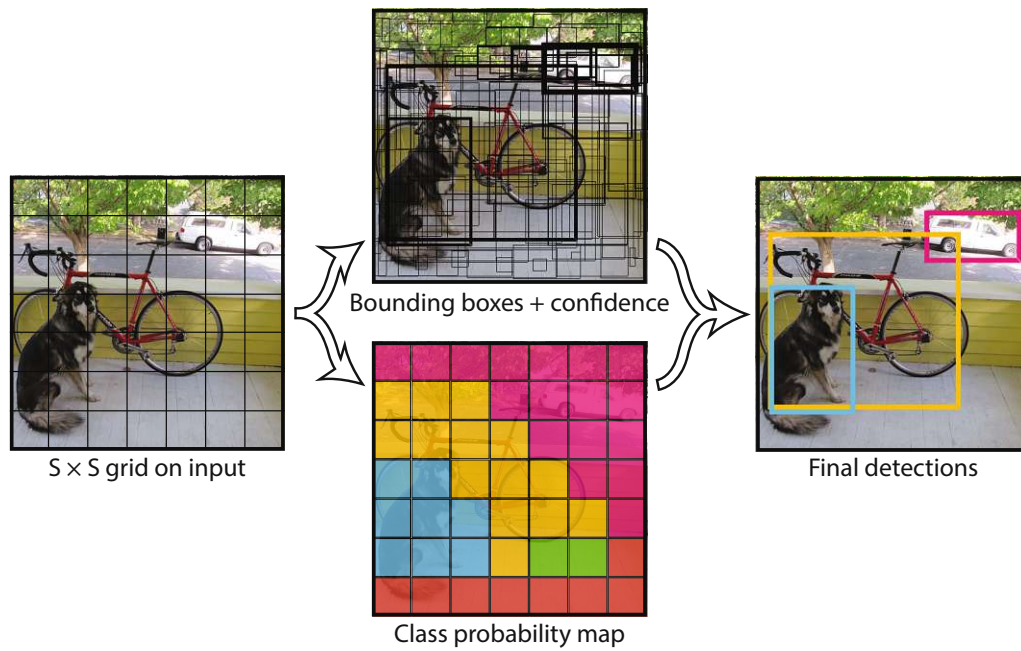


Figure 2.7: YOLO detection process [40]

2.3 Image Segmentation

Image segmentation is a core task in computer vision that involves partitioning an image at the pixel level, assigning a semantic label to every pixel, thereby delineating the spatial extent and boundaries of objects and regions within a scene. Unlike object detection, which aims to locate and classify objects using bounding boxes, segmentation provides dense, fine-grained outputs that fully outline shapes and regions, as illustrated in Figure 2.8.

Figure 2.8: Example of a segmentation prediction
COCO image id 116712 with superimposed segmentation masks [29].

Both segmentation and detection share the overarching goal of identifying what objects are present

and where they are located, and they often employ similar architectural strategies such as CNNs for hierarchical feature extraction and encoder–decoder frameworks. However, segmentation differs from detection in its explicit focus on pixel-wise classification, resulting in more precise object contours and the ability to segment amorphous regions or overlapping instances instead of coarser localization through bounding boxes achieved by object detection.

The following section introduces some CNN-based state-of-the-art image segmentation models, briefly describing their structure and respective approaches to segmentation.

2.3.1 U-Net

U-Net [52] is an image segmentation network that uses an encoder-decoder architecture supplemented with skip connections between corresponding layers, resulting in the distinct U-shape shown in Figure 2.9.

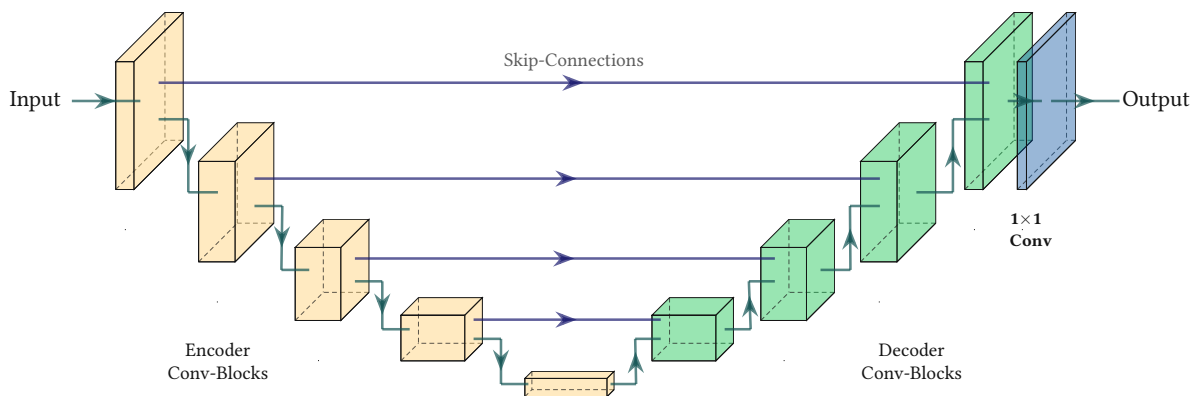


Figure 2.9: UNet architecture

The encoder progressively reduces the spatial resolution of the input image while capturing increasingly abstract features, whereas the decoder gradually restores the original resolution to produce a pixel-wise segmentation mask. The added skip connections enable the direct transfer of high-resolution details from the encoder to the decoder, allowing the network to recover fine-grained features lost during downsampling. This architecture makes U-Net and derivative architectures like Residual U-Net (ResU-Net) [53], UNet++ [54], and MultiResUNet [55] particularly effective in domains requiring precise segmentation with limited training data, such as biomedical imaging.

2.3.2 DeepLab

DeepLab [56] introduced the use of atrous (dilated) convolution to expand the receptive field without reducing spatial resolution, enabling effective dense feature extraction from images. Subsequent developments in DeepLabv2 [57] extended these ideas by adding Atrous Spatial Pyramid Pooling (ASPP), a module that captures multi-scale contextual information using parallel atrous convolutions with varying rates, as shown in Figure 2.10.

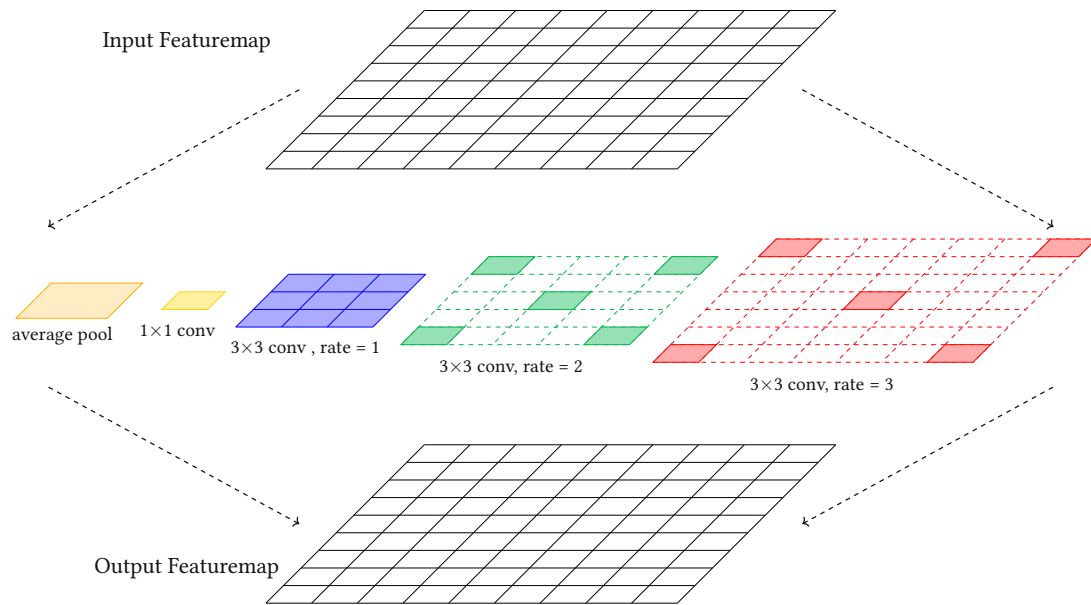


Figure 2.10: ASPP module structure

This enhancement allows the network to more robustly segment objects at different scales. DeepLabv3 [58] further refines the architecture by optimizing the ASPP module and integrating batch normalization and improved backbone designs. DeepLabv3 outputs dense, high-quality segmentation maps, and has demonstrated strong results on complex benchmarks with diverse object sizes and challenging backgrounds.

2.3.3 Bilateral Segmentation Network

Bilateral Segmentation Network (BiSeNet) [59] splits the segmentation task into different paths: a spatial path to preserve spatial information while producing high-resolution feature maps and a context path that rapidly downsamples the input to capture a large receptive field and high-level semantic context. The spatial path consists of a few convolutional layers with a small stride, maintaining the high-resolution features critical for accurate boundary localization, while the context path uses efficient backbone networks along with Attention Refinement Modules (ARM) to aggregate semantic information over the entire image. The two paths are then unified using a special Feature Fusion Module

(FFM) employing a combination of concatenation, pooling, and batch normalization. The structure of BiSeNet is shown in Figure 2.11.

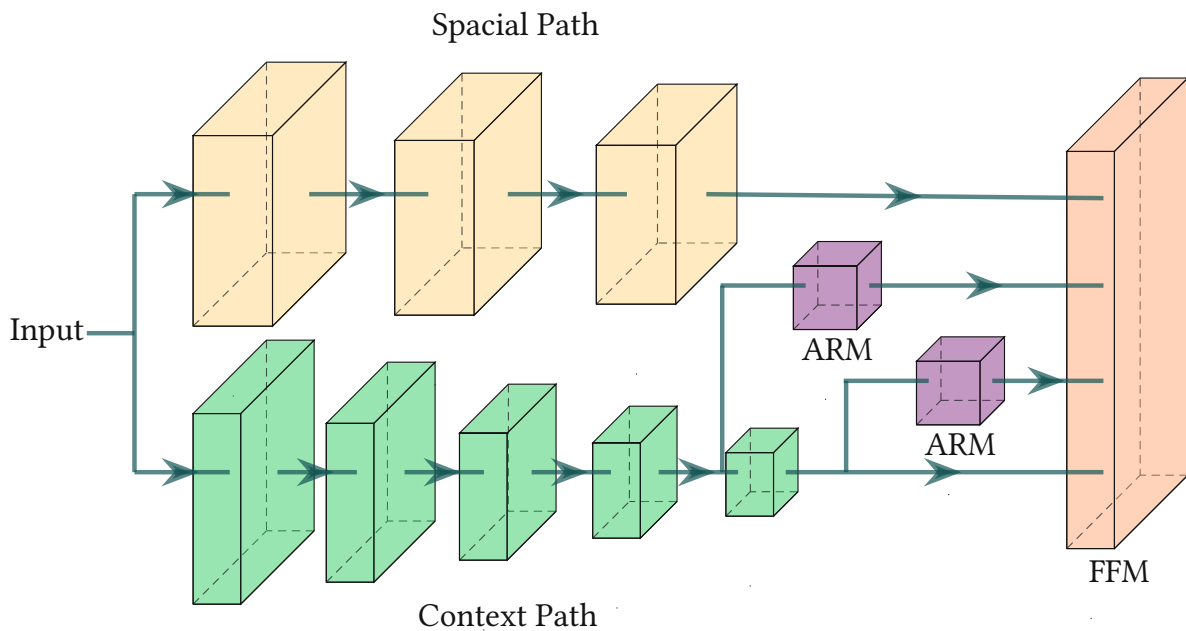


Figure 2.11: BiSeNet model structure

Subsequent versions, BiSeNetV2 [60] and BiSeNetV3 [61], build on the original by improving accuracy and efficiency. BiSeNetV2 refines the spatial and context paths for faster, stronger feature learning, while BiSeNetV3 adds transformer-based attention and multi-task capabilities for richer context modeling.

2.4 Multi-Task Networks

Multi-task networks are designed to learn and perform multiple related tasks at the same time, such as object detection, segmentation, and depth or pose estimation, within a single shared architecture, rather than training and operating separate models for each task. Compared to single-task networks, these approaches encourage feature sharing and joint optimization, often improving overall efficiency and performance for complex vision problems. As this thesis focuses on object detection and segmentation, the following multi-task networks highlighted are primarily centered on these domains.

2.4.1 Mask R-CNN

Building on the Faster R-CNN architecture introduced in Section 2.2.1, Mask R-CNN [62] extends the model by adding a segmentation prediction branch in parallel to the existing classification and bounding box regression pipelines. This allows the framework to output not only object class labels and localization via bounding boxes but also a binary mask for each detected object, enabling instance segmentation, where each detected object has its own segmentation mask.

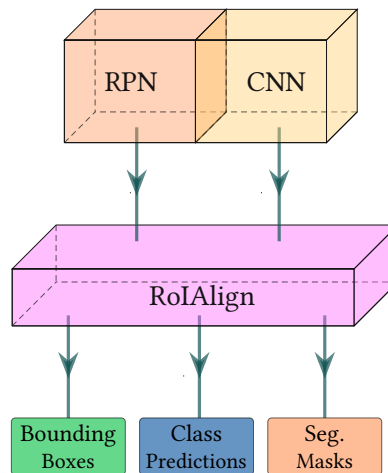


Figure 2.12: Mask R-CNN architecture

The quantization introduced by the RoI pooling in Faster R-CNN causes a misalignment between the RoI and the extracted features, which does not significantly affect classification for object detection but has a substantial negative impact on segmentation accuracy. To address this, Mask R-CNN replaces RoI pooling with RoIAlign, as shown in Figure 2.12. RoIAlign removes the quantization step by interpolating feature map values at precise spatial locations, thereby preserving the alignment between the RoI and the features. This enables the segmentation branch to receive accurately aligned feature maps, resulting in improved instance segmentation performance.

2.4.2 YOLOv11 Segmentation

YOLOv11 represents a well-established version of the YOLO model series discussed in Section 2.2.3, introducing significant advances in joint object detection and instance segmentation. Building upon the efficient, unified architecture of its predecessors, YOLOv11 integrates new components, such as the Enhanced Attention Mechanism (EAM), C3k2 block, and C2PSA (Convolutional block with Parallel Spatial Attention), that enhance feature extraction and spatial reasoning capabilities, essential for accurate detection and segmentation within a single network [50, 63]. Unlike two-stage methods, such as Mask R-CNN that detect objects first and then segment them, YOLOv11 predicts bounding boxes,

class labels, and per-instance masks during a single forward pass. The model head generates detection boxes across three different scales: low, medium, and high, as well as segmentation masks, enabling both object localization and fine-grained mask prediction. YOLOv11 is available in five model sizes (N, S, M, L, X), each providing a trade-off between speed and accuracy, making it suitable for real-time applications on edge devices or for high-precision requirements on powerful hardware.

In addition to object detection and segmentation, YOLOv11 supports extensions for pose estimation, keypoint detection, and oriented bounding box (OBB) prediction within a unified framework. However, this unified architecture can make the model less flexible for further modification and can limit independent task-specific optimization compared to other, more modular approaches.

2.4.3 Non-Standard Approaches to Multi-Task Detection and Segmentation

While most multi-task CNN models for object detection and segmentation rely on joint backbone networks with relatively straightforward parallel heads, such as the previously discussed Mask R-CNN and YOLOv11, recent literature demonstrates innovative, non-standard strategies for integrating and optimizing these tasks, especially in the field of autonomous driving applications. The following section highlights such approaches from three representative papers:

- **Shared Encoder with Task-Specific Decoders** [64]:

This approach introduces an architecture in which object detection and semantic segmentation are trained jointly using a shared, compact encoder based on ResNet10 [65]. The detection task is handled by a YOLOv2-inspired decoder branch, while segmentation is performed by an FCN8-like decoder [66]. Designed for real-time operation on low-power hardware, the architecture balances memory and computational resources along with predictive performance. Additional optimizations include restricting segmentation to regions below the horizon line, ensuring the network focuses on roadway elements relevant to driving. Furthermore, the study demonstrates the benefits of cross-task knowledge distillation in scenarios where only partial annotations are available, by allowing imitation loss from one task to guide learning in the other, thereby improving generalization and efficiency despite incomplete labeling.

- **Cross-Attention and Inner-Attention** [67]:

Here, inner-attention is employed to enhance feature representations within each task-specific branch, while cross-attention explicitly links the detection and segmentation branches. This cross-attention mechanism enables segmentation features to inform and refine detection out-

puts, with joint backpropagation further enhancing segmentation performance. As a result, features learned for segmentation actively contribute to improving object detection. The approach incorporates tailored loss functions to maintain training stability and promote mutual benefits between tasks. Experimental results demonstrate superior detection of small and occluded objects, as well as sharper and more accurate segmentation boundaries compared to conventional methods.

- **Weighted Feature Pyramid Network** [68]:

To address the loss of contextual information inherent in simple encoder–decoder architectures, a bi-directional weighted feature pyramid network (BiFPN) is introduced. This design fuses multi-scale features for both tasks, ensuring that high-level semantic information and low-level spatial details are effectively captured in the detection and segmentation heads. The framework supports multiple encoder options, such as CSPResNet50 [69] and Xception65 [70], each optimized for its respective task. Dedicated decoders for detection and segmentation preserve flexibility for downstream applications while enabling real-time inference. The approach achieves state-of-the-art performance on the custom ICV22 and Cityscapes [71] datasets, all with efficient resource utilization.

2.5 Transformer-based Models

The transformer model represents a major shift in deep learning architectures by relying exclusively on self-attention mechanisms [72]. Unlike traditional CNNs, transformers capture global dependencies by directly modeling interactions between all elements (such as pixels or image patches) in parallel, making transformers highly effective for a wide range of tasks beyond just sequential data. While the present work primarily focuses on CNN-based architectures, a brief mention of representative transformer-based models is included here for completeness.

In object detection, transformers have led to the development of models that treat detection as a direct set prediction task, replacing hand-designed, multi-stage components from CNN-based approaches and simplifying the model. Three popular and influential transformer-based detection architectures are:

- **DETR** (DEtection TRansformer) [73]: The first mainstream detection transformer, using set-based prediction to produce object boxes and labels end-to-end.
- **Deformable DETR** [74]: An improved variant of DETR that introduces multi-scale and

sparse attention to accelerate convergence and boost detection accuracy.

- **DINO** (DETR Improved with Optimization) [75]: A further enhancement of the DETR architecture with advanced attention mechanisms and training strategies.

For segmentation, transformers have proven to be well-suited for various image segmentation tasks, thanks to their ability to effectively understand scene structure, preserve fine details, and their overall robustness under varying conditions.

- **TransUNet** [76]: Building on the UNet segmentation model, TransUNet incorporates transformer blocks into the encoder structure.
- **SegFormer** [77]: SegFormer introduces a hierarchically structured transformer-based encoder to output multi-scale features, while also keeping the segmentation head lightweight.
- **SAM** (Segment Anything Model) [78]: A promptable image segmentation model that, in addition to an input image, can take optional prompts, such as keypoints or boxes to provide further context and guide the prediction, resulting in more precise and accurate segmentation outputs.

Similar to CNN-based architectures, transformer backbones can feed multiple output heads for various tasks, such as detection, segmentation, depth estimation, and more, within a single model.

Here are representative transformer-based multi-task models:

- **Mask DINO** [79]: Mask Dino enables segmentation by adding a parallel mask prediction branch to DINO’s original object detection branch, while also allowing both tasks to benefit from shared features to improve multi-task learning.
- **MTFormer** [80]: The multi-task transformer framework utilizes a shared transformer encoder-decoder architecture with light-weight branches to extract task-specific outputs from the unified pipeline.
- **DeMT** (Deformable Mixer Transformers) [81]: A deformable mixer transformer model using a shared encoder with channel-mixing and deformable convolutions combined with a task-aware transformer decoder.

However, transformer-based models typically have higher memory requirements [82] and require larger datasets [83] during training to achieve state-of-the-art results compared to CNNs, which demonstrate better parameter efficiency.

Chapter 3

Methodology

This section details the methods and design choices made in order to implement and evaluate three different approaches to a shared encoder architecture for object detection and image segmentation. A primary goal throughout this work is to develop models suitable for deployment on embedded devices, requiring careful consideration of computational efficiency and resource constraints. It also outlines the training process as well as the technical implementations and experimental setups used to assess model performance. Key hyperparameters, dataset selection, and evaluation metrics are described, providing a comprehensive overview of the procedures followed to ensure fair and consistent comparison between the proposed approaches.

3.1 Core Principle

A common approach to end-to-end network architectures, as described in chapter 2, is to use an encoder as a backbone to extract features and generate feature maps at different scales. The backbone provides these features to a function head, which utilizes them to perform specific tasks such as object detection or image segmentation. It is common practice to deploy two separate networks, with each network dedicated to a specific task. One network is optimized for object detection, while the other is optimized for image segmentation, ensuring that each function head is tailored for its respective objective. However, this approach introduces considerable computational overhead, as the input image must be processed independently by two distinct backbone networks. This redundancy not only increases resource consumption, but also complicates deployment in real-world applications.

To overcome this issue, we introduce a shared encoder structure. By reusing feature-maps computed by the encoder for detection and segmentation, unnecessary recomputation can be avoided, improving efficiency.

The training is divided into three stages, each featuring a progressively greater degree of encoder

sharing:

1. Encoder Swapping

In this approach, the encoder is first trained within a single-task network and subsequently frozen. The pre-trained encoder is then reused for the second task, serving as a fixed feature extractor without further updates during the training of the additional task.

2. Encoder Sharing

The encoder is pre-trained in a task-agnostic manner to generate diverse feature maps. Once pre-training is complete, the encoder is frozen and subsequently utilized as a fixed feature extractor for both object detection and image segmentation tasks.

3. Dual Head Network

A single shared encoder processes the input and generates feature maps, which are then simultaneously fed into two separate task-specific heads: one for object detection and one for image segmentation. This architecture allows for an end-to-end training for both tasks at the same time.

All models, training procedures, and evaluations were implemented in PyTorch [84].

3.2 Dataset

Unless stated otherwise, all experiments were conducted using the widely adopted Common Objects in Context (COCO) [29] dataset to ensure a balanced mix of class diversity, image quantity, and overall data variety. Specifically, a 20-class subset, shown in Table 3.1, was selected to match the classes found in the popular Pascal Visual Object Classes (Pascal VOC) [85] dataset. As the dataset contains both

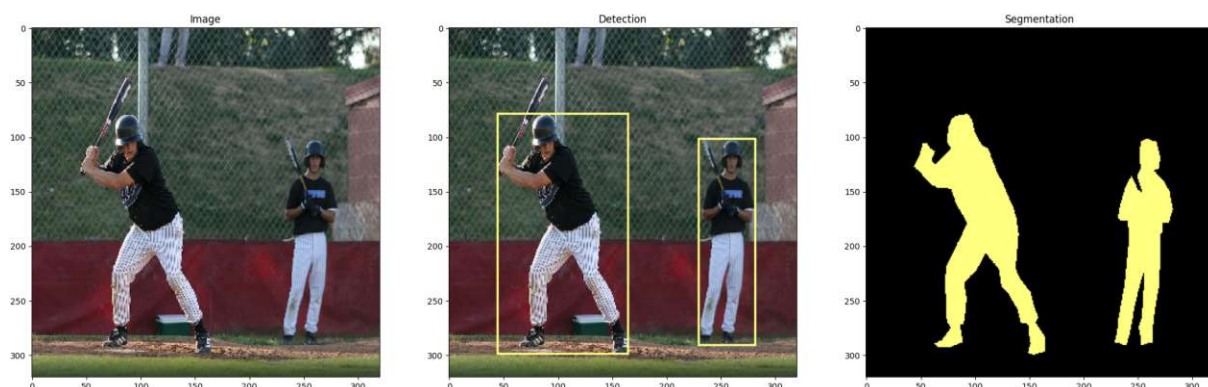


Figure 3.1: Example of the ground truth, bounding boxes, and segmentation masks provided by the COCO dataset, id 372764 [29]

object bounding box data for object detection as well as segmentation data for image segmentation, as shown in Figure 3.1, it was possible to evaluate and compare model performance across both tasks in a

Class No.	Class Name	Class No.	Class Name
1	aeroplane	11	dining table
2	bicycle	12	dog
3	bird	13	horse
4	boat	14	motorbike
5	bottle	15	person
6	bus	16	potted plant
7	car	17	sheep
8	cat	18	sofa
9	chair	19	train
10	cow	20	tv/monitor

Table 3.1: The selected 20 class subset corresponding to the Pascal VOC dataset.

unified experimental setting. All input images were resized to 320×320 pixels before being passed to the models. For better generalization, the following data augmentation techniques were applied during training:

- random horizontal flip
- Gaussian blur, kernel: 5×5 , $\sigma \in [0.1, 2.0]$
- color jitter, maximum values: brightness: 0.2, contrast: 0.2, saturation: 0.2, hue: 0.1
- random crop-resize, maximum crop: 50%



Figure 3.2: Data augmentation techniques applied to the ground truth of Figure 3.1.

Using only the selected classes, the COCO train 2017 subset comprises 95279 images, while the COCO

val 2017 subset includes 4031 images. To facilitate robust training, the COCO train 2017 data were partitioned into a training set (95%) and a validation set (5%). The entire COCO val 2017 dataset was exclusively used for final testing and evaluation. An overview of these data splits is provided in Table 3.2.

Dataset	Split	Number of images
COCO Train 2017	95% (train)	90515
COCO Train 2017	5% (validation)	4764
COCO Val 2017	test	4031
Total images		99310

Table 3.2: COCO dataset splits and their image counts.

In addition to COCO, the Pascal VOC [85] dataset and the CIFAR-10 [86] dataset were used for preliminary testing and exploratory experiments. Pascal VOC provides a well-established benchmark for object detection and segmentation with 20 labeled classes. The CIFAR-10 dataset consists of 60,000 low-resolution images across 10 object classes, making it ideal for rapid prototyping and initial validation.

3.3 Model Architectures

We selected four widely used backbone architectures to evaluate the effects of sharing backbones and the transferability of trained encoder weights. These backbones were chosen based on their widespread use, streamlined deployment, and efficiency, with a focus on suitability for embedded device deployment. Furthermore, the function heads for detection and segmentation were selected for their ease of use and compatibility with the selected backbones.

3.3.1 Residual Network (ResNet)

ResNet [87] introduces the concept of residual learning by incorporating shortcut or skip connections that bypass one or more layers in the network, allowing for direct signal propagation and facilitating the training of deeper architectures. The fundamental component of ResNet is the basic block, depicted in Figure 3.3. The basic block in ResNet consists of the following sequence:

- A 3×3 convolution, followed by batch normalization and a ReLU activation.
- Another 3×3 convolution, followed by batch normalization.
- A skip connection that adds the block's input to its output.
- An optional downsampling 1×1 convolution layer, used if the dimensions or channels between input and output do not match.

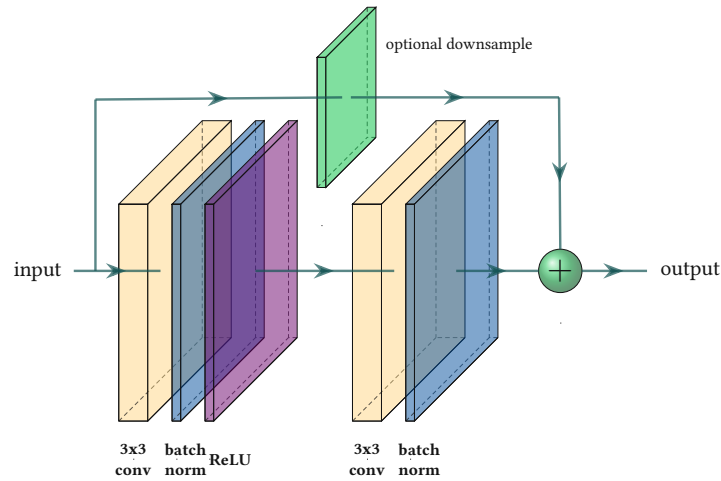


Figure 3.3: ResNet basic block.

The ResNet architecture is highly scalable, as layers and residual blocks can be easily added or removed to match the desired balance between accuracy and computational workload. In this work, the ResNet18 implementation provided by PyTorch has been used as backbone (see Figure 3.4).

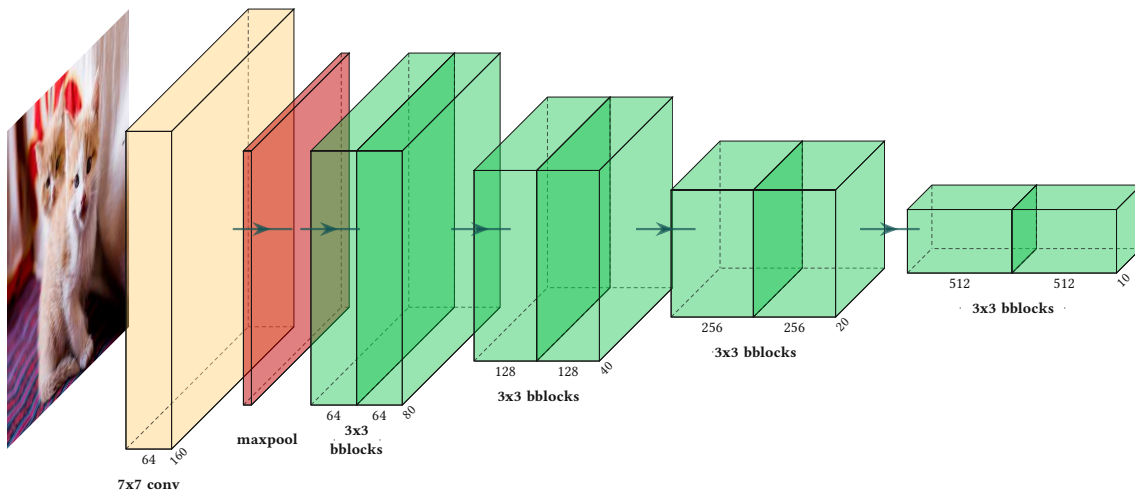


Figure 3.4: ResNet18 encoder structure in PyTorch. Input 320x320 pixels.

3.3.2 MobileNetV3

MobileNetV3 [88] is a lightweight CNN architecture optimized for both accuracy and efficiency, targeting mobile and embedded applications. The MobileNetV3 bottleneck block, its basic building block is shown in Figure 3.5 and consists of:

- An initial 1×1 point-wise convolution to expand the number of channels.
- A depthwise convolution, either 3×3 or 5×5 .
- An optional squeeze-and-excitation layer for adaptive channel-wise weighting.

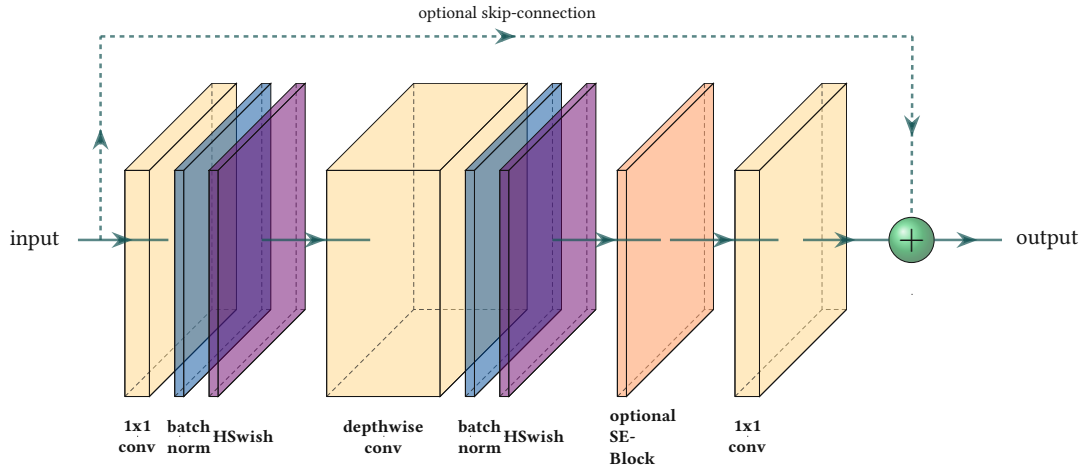


Figure 3.5: MobileNetV3 bottleneck block.

- A 1×1 convolution to compress the features to the desired bottleneck output size.
- When input and output shapes match, an optional residual connection.

The Hard-Swish [89] activation function is used, for improved accuracy as well as reduced computational workload. MobileNetV3 is available in both Large and Small variants to suit different resource constraints, offering an effective balance between performance and efficiency. In this work, the MobileNetV3-Large implementation provided by PyTorch has been used as backbone (see Figure 3.6).

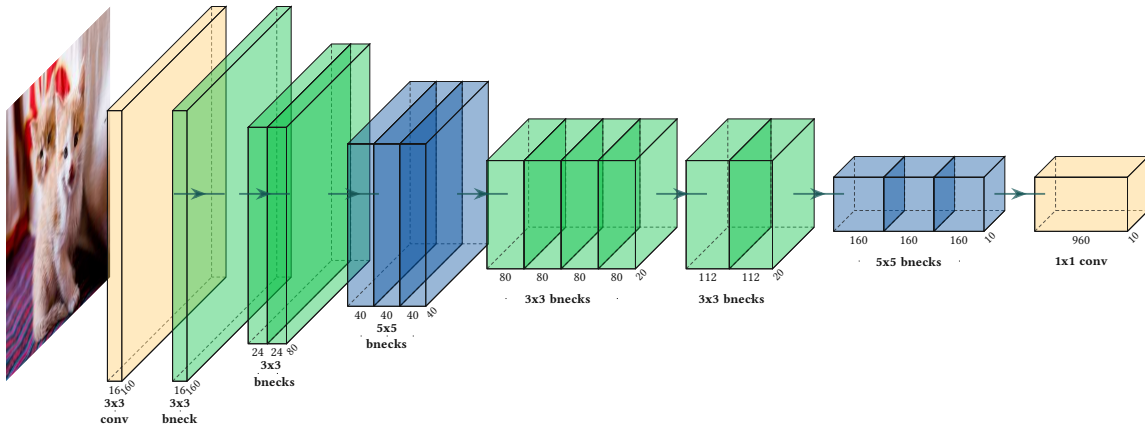


Figure 3.6: MobilenetV3-Large encoder structure in PyTorch. Input 320x320 pixels.

3.3.3 DenseNet

DenseNet [90] is a densely connected CNN designed to improve information flow and feature reuse across layers, enhancing both learning efficiency and accuracy. Its special module is the dense block, shown in Figure 3.7, where each layer is connected to every following layer in a feed-forward manner. This means that, instead of receiving input only from the previous layer, each layer takes the concate-

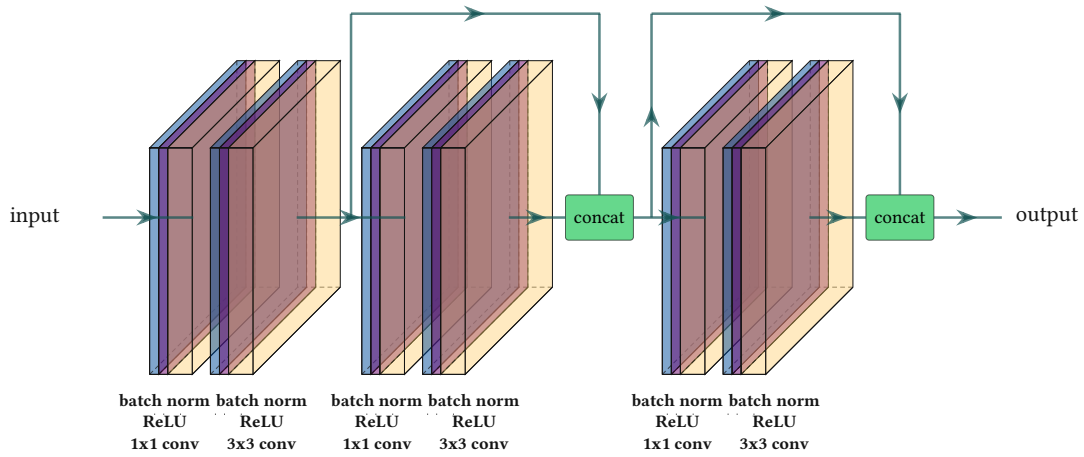


Figure 3.7: 3-layer DenseBlock

nated outputs of all preceding layers within the same block as input. Each layer inside a dense block, commonly referred to as a dense layer, has of a two-step composite structure:

- First, a sequence of batch normalization, ReLU activation, and a 1×1 convolution.
- Second, another sequence of batch normalization, ReLU activation, and a 3×3 convolution.

Between each pair of dense blocks, a specialized transition layer adjusts the dimensions of the feature maps using both convolution and pooling operations. DenseNet was introduced in four variants, primarily distinguished by their depth and number of layers: DenseNet121, DenseNet169, DenseNet201, and DenseNet264. In this work, the DenseNet121 implementation provided by PyTorch has been used as backbone (see Figure 3.8).

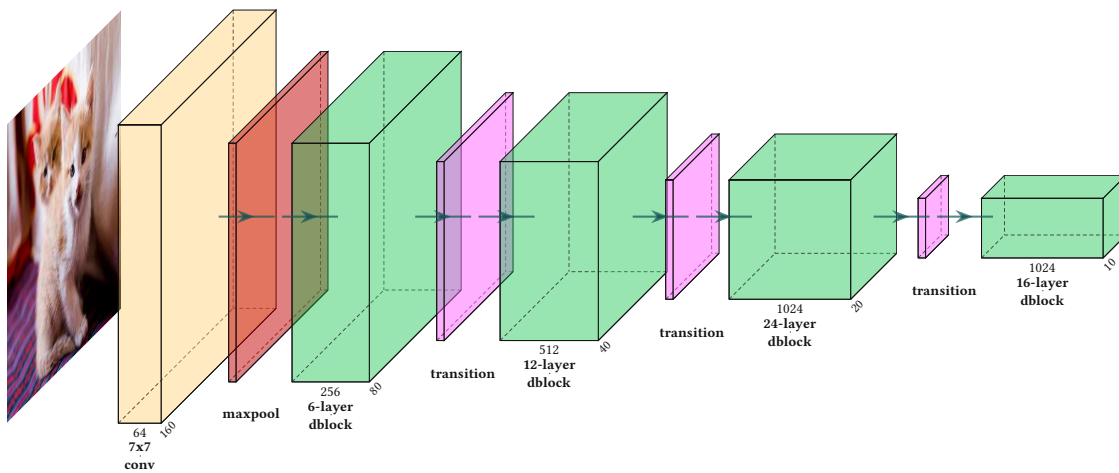


Figure 3.8: DenseNet121 encoder structure in PyTorch. Input 320×320 pixels.

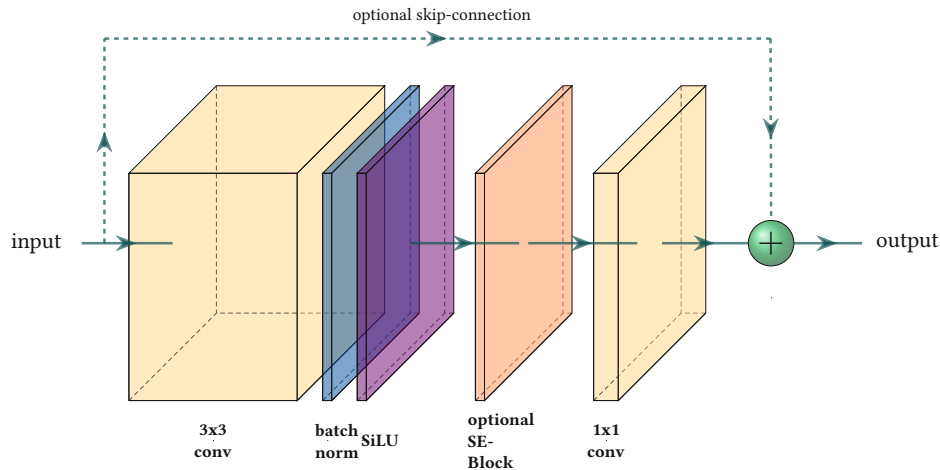


Figure 3.9: Fused-MBConv block.

3.3.4 EfficientNetV2

EfficientNetV2 [91] is a convolutional neural network architecture designed for fast training and high parameter efficiency, making it well-suited for large-scale and resource-constrained applications. It builds on the original EfficientNet [92] by using both Mobile Inverted Bottleneck Convolution (MB-Conv), the same type of block used as bottleneck in MobileNetV3 and Fused-MBConv blocks, which significantly improve training speed and reduce memory access costs by merging the expansion and depthwise convolution operations into a single 3×3 convolutional layer. The Fused-MBConv block consists of:

- A 3×3 convolution, for both expanding the channels and spatial filtering.
- An optional squeeze-and-excitation layer for adaptive channel-wise weighting.
- A 1×1 convolution to compress the features to the desired bottleneck output size.
- When input and output shapes match, an optional residual connection.

In contrast to MobileNetV3, EfficientNetV2 uses the more accurate, though computationally more demanding, SiLU or swish activation function in place of the lighter Hard-Swish used in MobileNetV3. Multiple variants, such as EfficientNetV2-S, -M, and -L, are available to accommodate different accuracy and efficiency requirements. In this work, the EfficientNet-S implementation provided by PyTorch has been used as backbone (see Figure 3.10).

3.3.5 Detection Head

For object detection, the SSD [39] head architecture was adopted. Specifically, the SSDLite design was employed due to its solid and competitive performance as well as its native compatibility with popular CNN backbones. Originally introduced in conjunction with MobileNetV2 [93], it has been adapted for

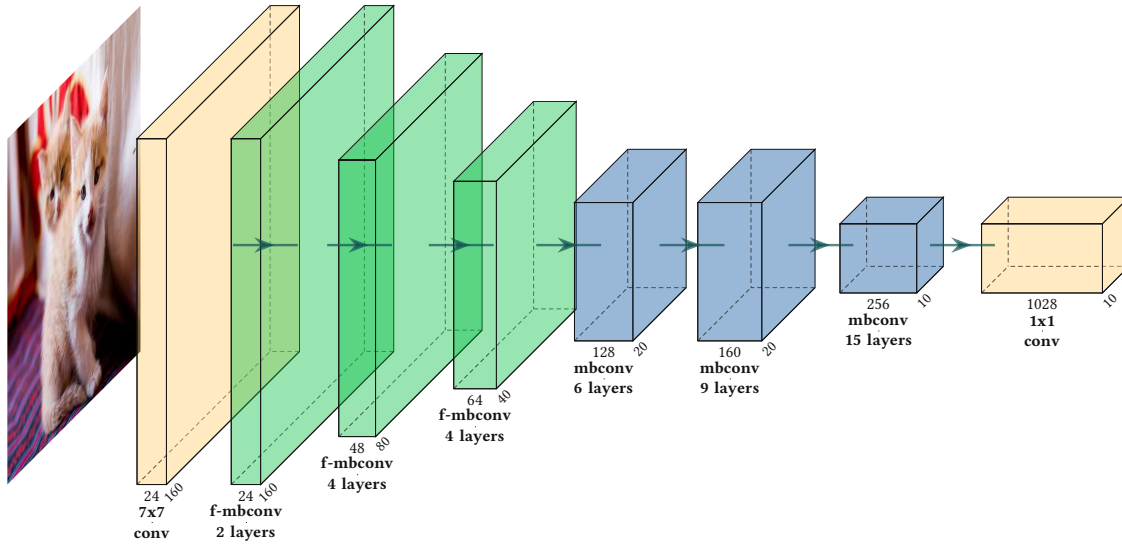


Figure 3.10: EfficientNetV2-S encoder structure in PyTorch. Input 320x320 pixels.

MobileNetV3, which is one of the backbone architectures evaluated in this work, making it particularly well-suited for integration within this experimental framework.

The SSD head consists of a series of convolutional layers that predict classification scores and bounding box offsets at multiple spatial resolutions, enabling robust detection of objects at different scales. To facilitate this multi-scale detection capability, the architecture incorporates auxiliary convolutional layers that process progressively lower-resolution feature maps, allowing the network to capture objects of varying sizes across the image. These auxiliary layers are appended on top of the backbone and are responsible for generating the additional feature maps used in detection at different scales, as shown in Figure 3.11.

The box generation mechanism in SSDLite is handled by the box generator, which is responsible for creating a fixed set of reference bounding boxes, commonly called default boxes or anchor boxes, at multiple locations, scales, and aspect ratios across feature maps of different resolutions. At each position in these feature maps, the box generator defines several anchor boxes, each parameterized by a specific scale and aspect ratio. These default boxes act as templates that the model learns to adjust (via predicted offsets) to fit the objects present in the image. For each anchor box, the network predicts both the class scores and the adjustments required to refine the box coordinates. This process allows SSDLite to efficiently localize objects of varying sizes and aspect ratios throughout the image, supporting robust multi-scale object detection.

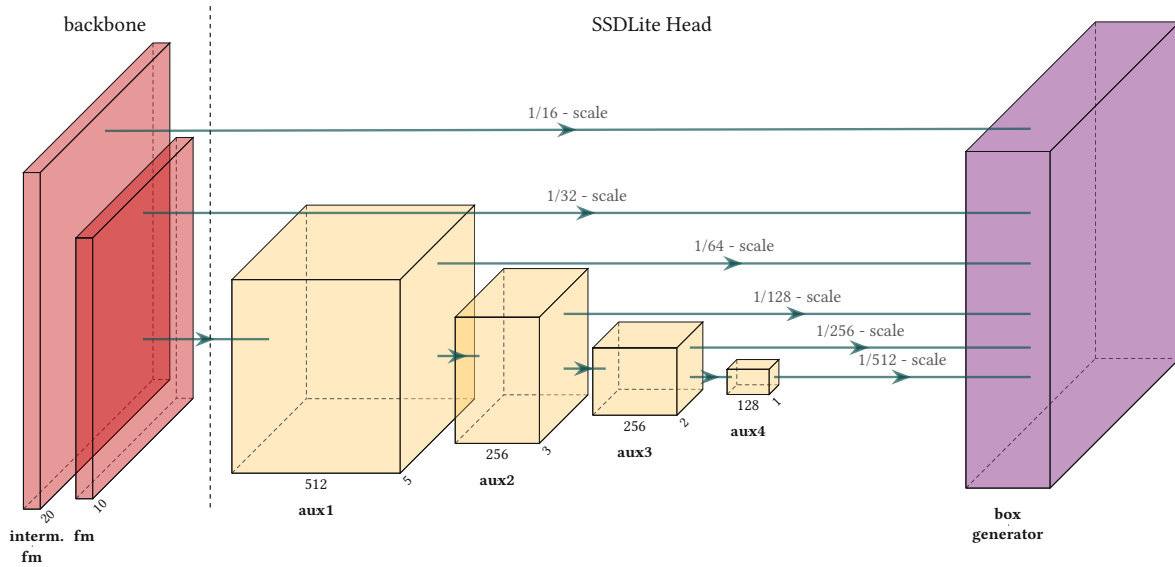


Figure 3.11: SSDLite detection head

3.3.6 Segmentation Head

For semantic segmentation, the Low-Resolution Atrous Spatial Pyramid Pooling (LR-ASPP) [88] head architecture was adopted. Like SSDLite, LR-ASPP was proposed together with MobileNetV3, forming an end-to-end efficient segmentation solution specifically tailored for mobile and resource-constrained environments. As depicted in Figure 3.12, the LR-ASPP head consists of two distinct processing paths:

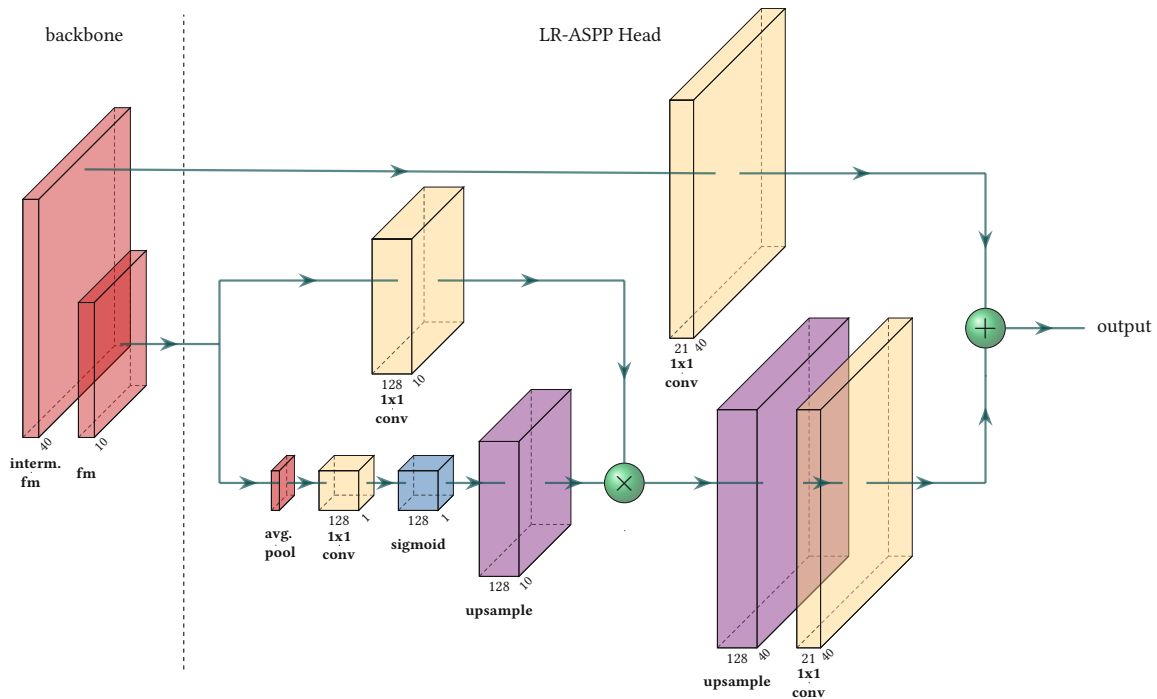


Figure 3.12: LR-ASPP detection head

one that operates directly on the backbone's final output, and another that utilizes an intermediate feature map extracted from the backbone.

In the first path, the backbone output, the high-level feature map, is first passed through a 1×1 convolution, followed by batch normalization and a ReLU activation. In parallel, the same feature map undergoes average pooling, after which it is processed by another 1×1 convolution and a sigmoid activation to generate channel-wise gating weights. These gating weights are then multiplied element-wise with the convolved feature map before being upsampled to match the spatial resolution of the extracted intermediate feature map, the low-level feature map. Lastly it is passed through a 1×1 classification convolution.

The second path directly takes the intermediate feature map and passes it through a classification layer. The two paths outputs are then combined to form the final segmentation logits.

3.4 Encoder Swapping

The most forward approach proposed in this work is the *encoder swapping* technique. In this approach, a single-task model is trained and fine-tuned on the given dataset. The encoder is then frozen to be used as the feature extractor in another single-task model, as shown in Figure 3.13. The initial model,

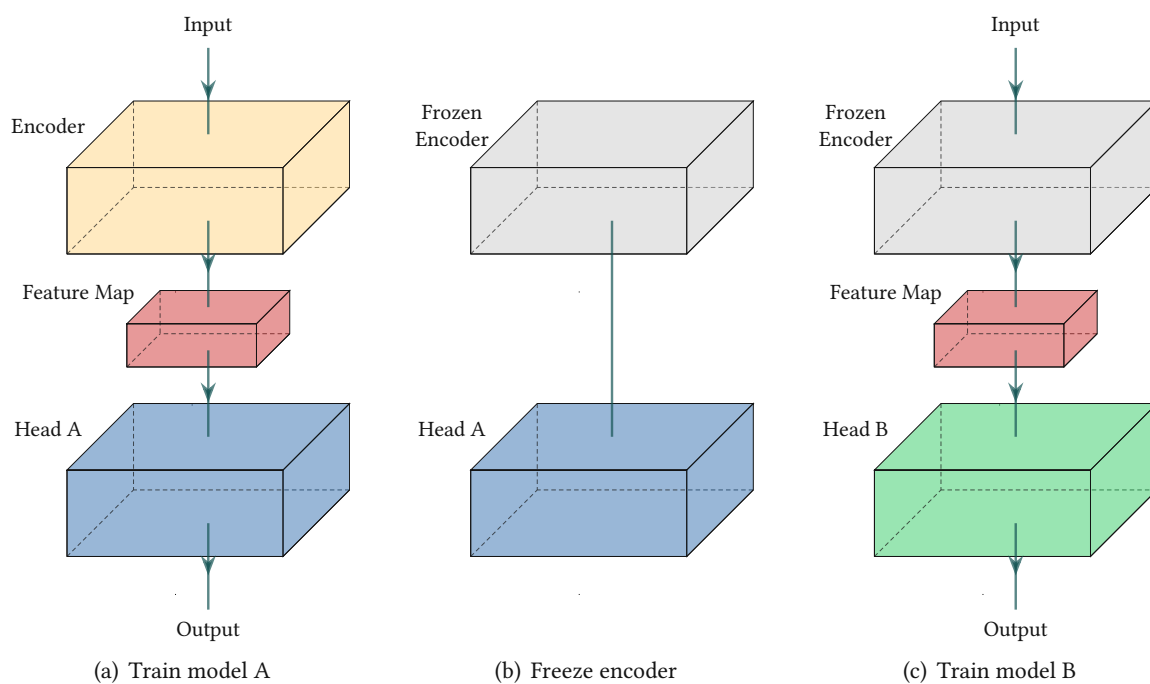


Figure 3.13: Visualization of the encoder swap approach.

such as a detection network, undergoes standard training and fine-tuning on the target dataset. Once the encoder (i.e., the backbone feature extractor) has been optimized for this task, it is extracted and its weights are frozen, exempting it from further training and effectively capturing task-specific feature representations. This frozen encoder is then integrated into the architecture of a second, distinct single-task model, such as an image segmentation network. During subsequent training, only the task-specific layers (e.g., decoder or segmentation head) are updated, while the encoder parameters remain fixed. This approach enables the assessment of how transferable and general the learned features are between tasks, and allows for direct investigation into the benefits of cross-task representation sharing, while also reducing the computational workload during training of the second network, as the encoder is excluded from gradient calculations.

To preliminarily assess the effectiveness of encoder swapping, a small-scale proof-of-concept experiment was conducted using the Pascal VOC 2007 and 2012 datasets, restricted to the two object classes: *car* and *dog*. This limited scope allowed for rapid iteration and reduced computational cost during initial development and testing. For object detection the SSD300 network [39] was selected, as its VGG16-based [94] encoder architecture facilitated straightforward modification for the extraction of intermediate feature maps necessary for encoder swapping. For the segmentation task, the SegFormer Head [77] was chosen due to its modular design, which allowed seamless integration with the VGG16 backbone. This plug-and-play compatibility between the detection and segmentation heads enabled efficient experimentation within the encoder swapping framework while minimizing development overhead. The training parameters are shown in Table 3.3.

Parameter	Detection	Segmentation
Epochs	200	200
Batch size	8	8
Initial learning rate	1×10^{-3}	1×10^{-3}
Adam weight decay	5×10^{-3}	5×10^{-5}
StepLR step size	67	20
StepLR gamma	0.1	0.9
Loss function	MultiBox loss	Cross-entropy loss

Table 3.3: Parameters for preliminary detection and segmentation training.

3.5 Encoder Sharing

This approach leverages pretraining to independently train the encoder to generate distinct and generalizable feature maps that are not specific to any particular task. After pre-training, encoder weights

are frozen to prevent further updates during training of task-specific network heads. This preserves the generality of the learned representations, ensuring that they remain broadly applicable and uninfluenced by the downstream task adaptations. Similarly to the encoder swapping method, the final network architecture consists of a frozen encoder coupled with a task-specific head tailored for the target task, as shown in Figure 3.14. This approach also decouples the resource-intensive process of

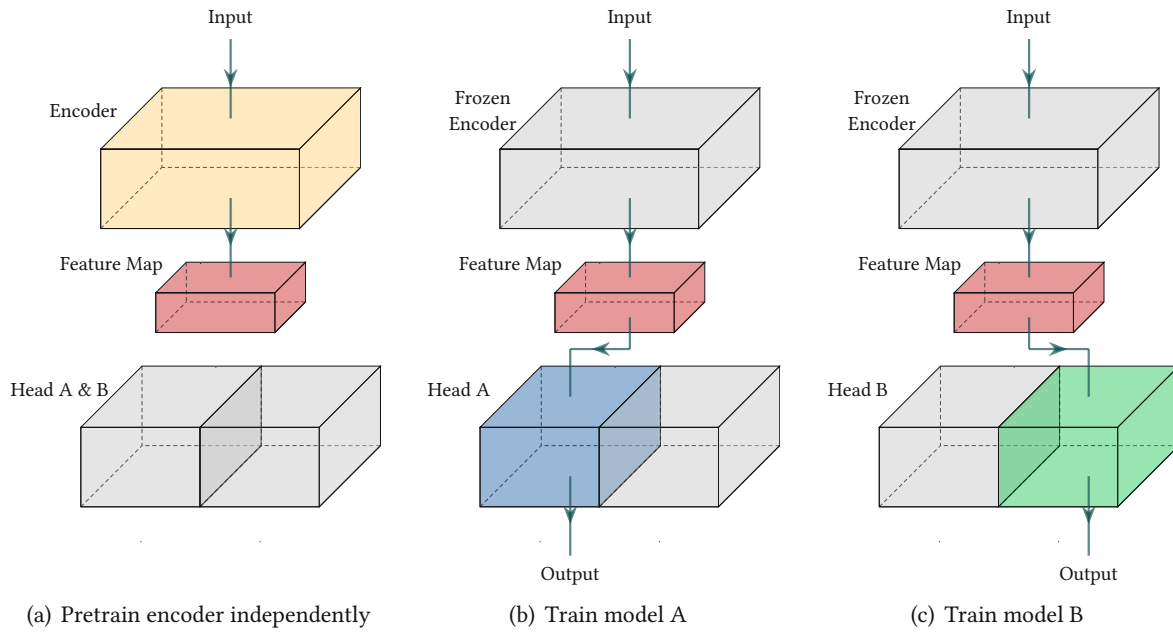


Figure 3.14: Visualization of the encoder sharing approach.

training the encoder from the task of training the network heads, enabling a single, flexible base encoder to be reused across multiple task-specific heads with only limited or no additional training. In addition, since each head is trained using a fixed encoder, this framework supports systematic comparison of task-specific architectures and provides a modular and efficient way to extend models to new tasks with minimal redundant computation.

As with the encoder swapping strategy, a preliminary proof-of-concept experiment was conducted to evaluate the practicality of the encoder sharing technique. For this feasibility study, a compact version of the ResNet [87] architecture, ResNet8 was employed as the encoder backbone for a classification task, and three distinct versions of the classification network were constructed: one with a single classification layer, one with an additional layer in the classification head, and one with three extra layers to emulate increasingly complex function heads. Using the CIFAR-10 [86] dataset, the ResNet8 encoder underwent supervised (SupCon [95]) and unsupervised (SimCLR [96]) pretraining, ensuring that learned representations remained task-agnostic. Afterward, the encoder weights were frozen and incorporated into each classification network variation. The performance of these models, trained solely

on the classification head while keeping the encoder fixed, was systematically compared against baseline networks where the entire model (encoder and head) was trained end-to-end.

Parameter	(Un)supervised Pretraining	Preliminary Classification
Epochs	500	200
Batch size	8192	64
Initial learning rate	5×10^{-2}	1×10^{-3}
Optimizer	SGD	Adam
Weight decay	1×10^{-4}	0
Momentum	0.9	–
Learning rate schedule	Decay at 350, 400, 450 epochs	Plateau (patience = 5, ratio = 0.5)
LR decay gamma	0.1	–
Loss function	SupCon loss / SimCLR loss	Cross-entropy loss

Table 3.4: Training parameters for (un)supervised pretraining and preliminary classification tasks.

3.6 Dual Head Network

A more advanced architectural approach explored in this work is the Dual Head Network. In this design, a single shared encoder is responsible for extracting feature representations from the input data. These shared feature maps are then propagated in parallel to two distinct task-specific heads as shown in Figure 3.15. By sharing a unified encoder between the two tasks, the network can leverage shared visual

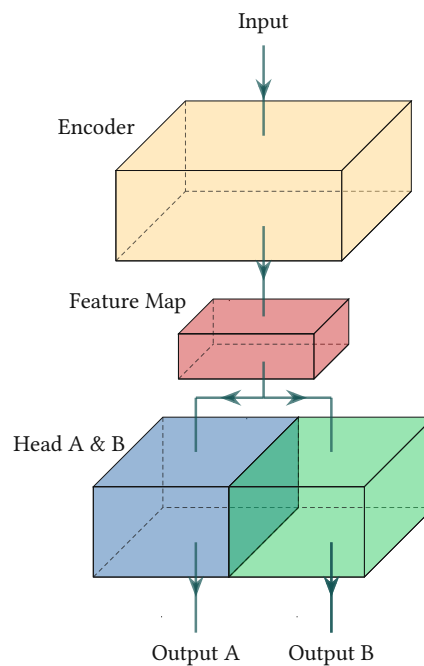


Figure 3.15: Visualization of the dual head network.

representations, which can lead to more efficient use of parameters and improved cross-task learning.

Both heads are trained together in an end-to-end manner, enabling the model to optimize shared as well as task-specific components simultaneously, while also requiring only a single training run for both tasks rather than two separate trainings. This multi-task setup is particularly advantageous for scenarios where related tasks can benefit from mutually adapted features, potentially enhancing overall performance and reducing the risk of overfitting to a single objective.

3.7 Training

For each backbone architecture, a parameter search was conducted to identify the optimal learning rates and learning rate scheduling strategies. Both cosine annealing and reduce-on-plateau learning rate schedulers were evaluated. Empirical results demonstrated that the reduce-on-plateau scheduler consistently yielded superior training performance across all backbones, while the optimal initial learning rates showed minimal variation among the tested architectures. The training parameters for preliminary tests performed on both the Pascal VOC as well as the CIFAR-10 dataset can be found in Sections 3.4 and 3.14.

Given the significant computational resources and time required for large-scale pretraining, all backbones were initialized with publicly available weights pretrained on the ImageNet [97] dataset, as provided by PyTorch. These pretrained weights, established on a comprehensive and diverse dataset, are recognized for their effectiveness in providing strong feature representations and accelerating training. The training parameters are shown in Table 3.5.

Parameter	Value
Epochs	1000
Batch size	32
Initial learning rate	2×10^{-4} (MobileNetV3_L: 8×10^{-4})
AdamW weight decay	5×10^{-4}
Plateau scheduler patience	10
Plateau scheduler ratio	0.5

Table 3.5: Training parameters for networks with four different backbones.

Early stopping was employed during training to prevent overfitting and optimize training efficiency. The training process was automatically halted if the validation loss showed no improvement over a consecutive span of 25 epochs. This strategy ensured that model development ceased once further training was unlikely to yield performance gains on the validation set, resulting in more robust and generalizable models.

3.7.1 Detection Loss

For object detection tasks, the SSDLite head employs the multi box loss [39], a composite objective function that guides the network to learn accurate object localization and classification simultaneously. The multi box loss consists of two primary components: a localization loss L_{loc} for bounding box regression and a confidence loss L_{conf} for class prediction and is formulated as:

$$L_{det}(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha \cdot L_{loc}(x, l, g)) \quad (3.1)$$

where N is the number of matched default boxes, and α is a weighting factor to balance the two terms. If $N = 0$ the loss is set to 0. As in the original SSD paper, α is set to 1. This loss function ensures both precise object localization and robust classification performance in a unified framework. The localization loss measures the discrepancy between the predicted bounding box coordinates and the ground truth. It uses a smooth L1 loss [36] between the predicted bounding box l and the ground truth box g parameters:

$$\begin{aligned} L_{loc}(x, l, g) &= \sum_{i \in Pos} \sum_{m \in \{c_x, c_y, w, h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m) \\ \hat{g}_j^{c_x} &= (g_j^{c_x} - d_i^{c_x}) / d_i^w & \hat{g}_j^{c_y} &= (g_j^{c_y} - d_i^{c_y}) / d_i^h \\ \hat{g}_j^w &= \log\left(\frac{g_j^w}{d_i^w}\right) & \hat{g}_j^h &= \log\left(\frac{g_j^h}{d_i^h}\right) \end{aligned} \quad (3.2)$$

where c_x and c_y denote the center coordinates of the default bounding box d , w and h denote its width and height, and $x_{ij}^p \in \{1, 0\}$ is a binary indicator for whether the i -th default box to the j -th ground truth box of category p match or not. The confidence loss evaluates the accuracy of predicted class probabilities c for each default anchor box and is defined as:

$$L_{conf}(x, c) = - \sum_{i \in Pos} x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)} \quad (3.3)$$

During training, these losses are computed for each positive anchor (i.e., anchors that have sufficient overlap with ground truth objects), denoted as Pos , while a subset of negative anchors (those that do not correspond to any object), denoted as Neg , are also sampled to address class imbalance.

3.7.2 Segmentation Loss

For semantic segmentation, the LR-ASPP head was trained using the cross-entropy loss, which, similarly to the confidence loss used for detection, quantifies the difference between the predicted class

probabilities, but does so for each pixel and the true pixel-wise labels. The pixel-wise cross-entropy loss encourages the model to assign high probability to the correct semantic class for every pixel in the output map:

$$L_{\text{seg}} = -\frac{1}{N} \sum_{i=1}^N \log(p_{i,y_i}) \quad \text{where} \quad p_{i,k} = \frac{\exp(x_{i,k})}{\sum_{c=1}^C \exp(x_{i,c})} \quad (3.4)$$

where N is the total number of pixels, y_i is the true class index for the i -th pixel, $x_{i,k}$ is the logit for class k at pixel i , p_{i,y_i} is the predicted softmax probability for the correct class at pixel i , and C is the number of classes. This loss function is both simple and effective, making it a standard choice for training modern semantic segmentation architectures.

During the training of the dual head networks, the detection and segmentation losses were combined into a single objective function using a weighting factor α :

$$L = L_{\text{det}} + \alpha \cdot L_{\text{seg}} \quad (3.5)$$

To ensure that both loss terms contributed equally to the overall training process, the value of α was set to 6. This choice places the detection and segmentation losses within the same value range, preventing one loss from dominating over the other, and enabling effective simultaneous learning of both tasks within the dual head architecture.

3.8 Evaluation Metrics

To assess and compare model performance across detection and segmentation tasks, a set of standard evaluation metrics is employed and detailed in this section.

3.8.1 Pixelwise Accuracy

A straightforward way to evaluate the performance of a segmentation model is to calculate its pixelwise accuracy, by assign a score of 1 to every correctly predicted pixel in the segmentation mask and a score of 0 otherwise. The final value is then calculated as the average over all the pixel scores in the image. For a given ground truth A and prediction B it is calculated as follows:

$$\text{Accuracy}(A, B) = \frac{1}{N} \sum_{i=1}^N p_{A_i, B_i} \quad \text{where} \quad p_{A_i, B_i} = \begin{cases} 1 & \text{if } A_i = B_i \\ 0 & \text{otherwise} \end{cases} \quad (3.6)$$

Pixelwise accuracy on its own is not always a reliable metric, as illustrated in Figure 3.16: even a poor prediction can yield a deceptively high accuracy score if the background comprises the majority of pixels, thereby skewing the result. Nevertheless, when comparing different segmentation models on the same dataset, pixelwise accuracy remains a useful tool for relative evaluation.

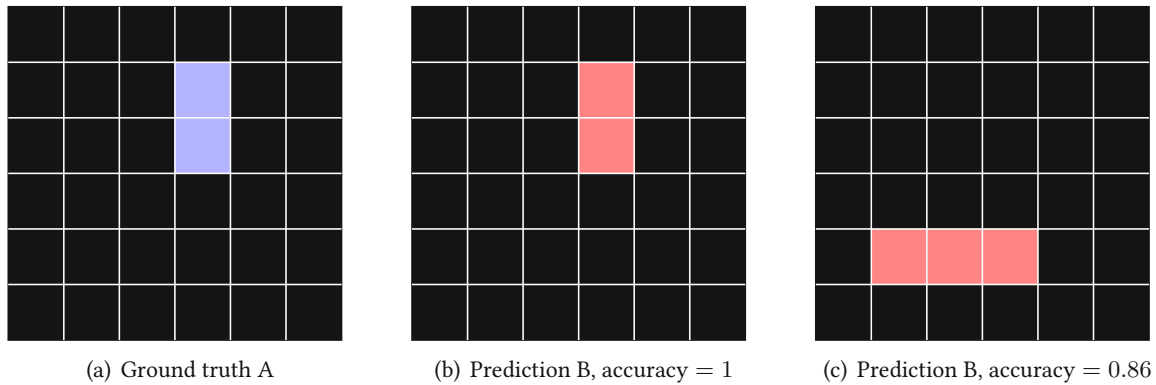


Figure 3.16: Pixelwise accuracy

3.8.2 Intersection over Union

The Intersection over Union (IoU) or Jaccard index [98] is a standard metric used to evaluate both object detection and image segmentation tasks. For a given predicted region, be it a bounding box or a segmentation mask, and the ground truth to compare against, IoU is defined as:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} = \frac{|A \cap B|}{|A \cup B|} \quad (3.7)$$

with A denoting the ground truth and B denoting the prediction, as illustrated in Figure 3.17. A higher

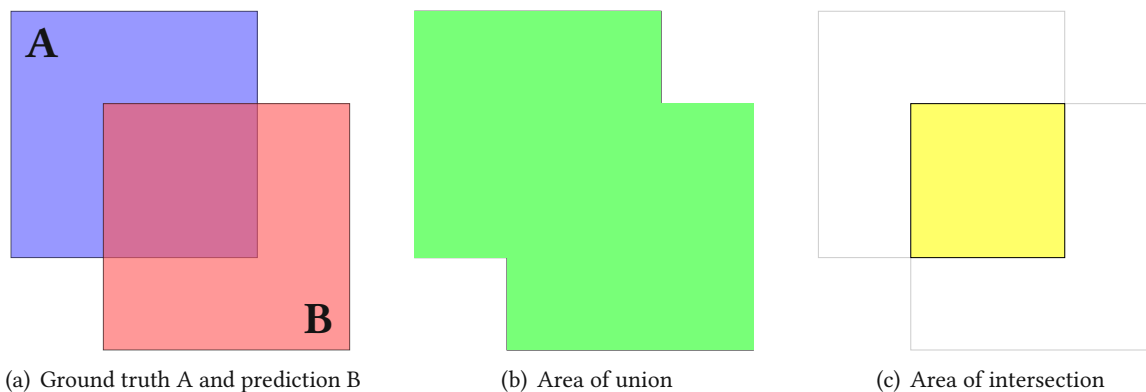


Figure 3.17: IoU

IoU indicates a closer match between the prediction and the ground truth.

In image segmentation, the IoU is computed for each class by finding the ratio of the number of overlapping pixels in both the predicted and ground truth masks to the number of pixels present in either mask. It provides a robust measure of similarity between the predicted segmentation and the annotation. In multi-class or multi-object segmentation, IoU is computed per class and averaged to report a mean Intersection over Union (mIoU), to offer a more balanced evaluation even with class imbalance.

For object detection, IoU is used to measure the degree of overlap between the predicted bounding box and the ground truth bounding box for each detected object. A predicted bounding box is considered a true positive if its IoU with a ground truth box exceeds a specific threshold, commonly 0.5 or 0.75, otherwise, it is counted as a false positive. This criterion allows for judging the accuracy of object localization.

3.8.3 Average Precision (AP)

AP is a widely used metric for evaluating the performance of object detection models, as it summarizes how well a detector can both identify and localize objects by computing the area under the precision-recall curve for each class. Precision P is calculated as the number of true positives T_P divided by the number of true positives plus false positives F_P , while recall R is defined as the number of true positives over the number of true positives plus the number of false negatives F_N :

$$P = \frac{T_P}{T_P + F_P} \quad \text{and} \quad R = \frac{T_P}{T_P + F_N} \quad (3.8)$$

To calculate AP, the predicted bounding boxes are ranked according to their confidence scores. For various thresholds, precision and recall values are computed based on matches between predicted boxes and ground truth boxes, typically using an IoU threshold, such as 0.5 to determine correct detections. The Average Precision is then given by the area under the precision-recall curve:

$$AP = \int_0^1 p(r) dr \quad (3.9)$$

where $p(r)$ is the precision as a function of recall. In practice, the area under the curve is approximated by interpolation at discrete recall values. A higher AP indicates that the detector achieves high precision across a wide range of recall values, meaning it finds most objects with few false positives. In most benchmarks, AP is computed per class and then averaged across all classes to obtain the mean Average Precision (mAP), which provides an overall measure of detection performance.

To provide a more comprehensive evaluation of different levels of strictness for the detected bounding boxes, it is common to calculate APs at multiple IoU thresholds. For the COCO dataset, it is common practice to compute AP at ten IoU thresholds ranging from 0.50 to 0.95 in increments of 0.05. The mean of these AP values is reported as the main performance metric, typically referred to as $mAP@[.50:.95]$ or *COCO mAP*. This approach ensures the evaluation considers both loose and very precise localization, offering a balanced assessment of a detection model’s ability to both find objects and accurately predict their boundaries.

3.9 Embedded Device

To evaluate the practical performance advantages of the proposed approaches and assess their suitability for real-world applications, the trained models were deployed on an embedded device for inference testing. The deployment tests were conducted on an NVIDIA Jetson AGX Orin [99], a popular edge computing platform designed for AI workloads.

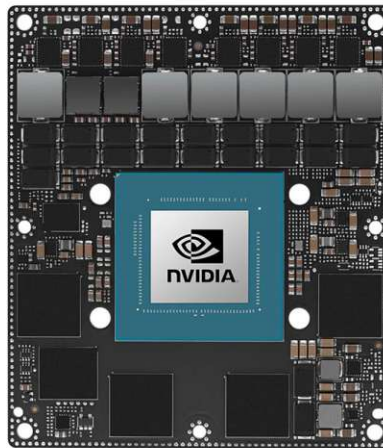


Figure 3.18: NVIDIA Jetson AGX Orin module [99]

The Jetson AGX Orin features an Ampere GPU with 2048 CUDA cores and 64 Tensor cores, making it representative of high-performance embedded systems used in autonomous vehicles, robotics, and industrial applications. To export and test the PyTorch models as `.onnx` models, the following settings were used:

- PyTorch version: 2.8.0
- Export input tensor: `shape=(1, 3, 320, 320), dtype=float32`
- Open Neural Network Exchange (ONNX) operator set: 12
- Inference precision: FP32 and INT8
- Configuration: Orin Nano 8GB

- TensorRT version: 10.3
- JetPack version: 6.2.1

Chapter 4

Experiments

This section presents an evaluation of the proposed shared encoder architectures for object detection and image segmentation. The experiments are designed to assess and compare the performance of the three approaches systematically, encoder swapping, encoder sharing, and dual head networks, across the four introduced backbone architectures on the 20-class subset of the COCO dataset. Each experiment aims to analyze the impact of encoder sharing strategy and backbone selection on model accuracy, computational efficiency, and cross-task feature transferability.

4.1 Preliminary Testing

As described in Sections 3.4 and 3.5, preliminary testing with a limited scope was conducted for both encoder swapping and encoder sharing. The purpose of these initial experiments was to assess the viability of each approach before proceeding to implement the full-scale models and run comprehensive trainings and evaluations.

4.1.1 Encoder Swapping

To determine the extent of performance degradation resulting from replacing the backbone encoder in each single-task model, baseline models and their encoder-swapped counterparts were evaluated. Detection models were assessed using mAP at 0.5 IoU, while segmentation models were evaluated using mIoU. The results are summarized in Table 4.1. The standard SSD300 detection head achieves an mAP of 0.744, while the variant utilizing a segmentation encoder experiences a substantial reduction to 0.356 . Similarly, for semantic segmentation, the SegFormer baseline attains an mIoU of 0.931, whereas swapping the encoder weights to those from the detection model reduces mIoU to 0.430 . These initial findings show that a simple encoder swap, without further modification or task-specific fine-tuning, leads to a notable decrease in quantitative performance for both detection and segmentation.

Metric	Model	Score	Rel. Change
mAP@0.5 IoU	SSD300	0.744	–
	SSD300 – swapped	0.356	–52.2%
mIoU	SegFormer	0.931	–
	SegFormer – swapped	0.430	–53.8%

Table 4.1: Performance of models with and without swapped encoders.

Consequently, this motivated further exploration of the other, more sophisticated approaches proposed in Section 3.1. In addition, the encoder swapping approach itself was also investigated in more detail through additional experiments, in order to better understand the factors underlying its performance and to assess whether further adjustments could mitigate the observed reductions.

4.1.2 Encoder Sharing

The impact of task-agnostic pretraining on classification performance was initially evaluated through small-scale experiments with ResNet8 models trained on the CIFAR-10 dataset. Models were tested with varying numbers of additional 1×1 convolutional layers (none, one, or three) following the encoder, in order to mimic more complex classifier head architectures. For each configuration, standard end-to-end training, supervised contrastive pretraining with frozen encoder weights, and unsupervised pretraining with frozen encoder weights were compared. Results are summarized in Table 4.2. The re-

Extra Layers	Training	Classification Accuracy (%)	Delta Acc. (pp.)
0	Normal	84.14	–
	Supervised + frozen	73.95	-12.11
	Unsupervised + frozen	58.51	-30.46
1	Normal	84.22	–
	Supervised + frozen	73.99	-12.15
	Unsupervised + frozen	63.81	-24.23
3	Normal	84.78	–
	Supervised + frozen	74.33	-12.33
	Unsupervised + frozen	64.76	-23.61

Table 4.2: Classification accuracy of ResNet8 on CIFAR-10 under different pretraining and architectural variants.

sults show that, as expected, standard end-to-end training consistently outperforms both supervised and unsupervised contrastive pretraining with frozen encoders. While adding extra classifier layers leads to a slight increase in absolute accuracy compared to configurations without them, the relative reduction in performance, when compared to the fully unfrozen, end-to-end trained model, becomes even larger.

Building on these findings, the next phase of experimentation increased both the architectural and

dataset complexity by advancing to the target detection and segmentation heads described in Section 3.3. For these experiments, MobileNetV3-Large was chosen as the encoder backbone. Furthermore, training and evaluation shifted to the Pascal VOC 2009 and 2012 datasets, which offer higher-resolution images and a greater number of classes than CIFAR-10 while still being considerably smaller than the ultimate target, the COCO dataset.

To investigate whether allowing some limited adaptation of the pretrained backbone could enhance overall model performance, an additional test configuration was introduced in which only the bias terms of the backbone were unfrozen during training, while the remaining weights remained fixed.

Head	Metric	Backbone	Score	Rel. Change (%)
SSDLite	mAP@0.5IoU	normal	0.7289	–
		biases unfrozen	0.5816	-20.2
		frozen	0.5162	-29.2
LR-ASPP	mIoU	normal	0.7031	–
		biases unfrozen	0.6732	-4.3
		frozen	0.6577	-6.5

Table 4.3: Performance of pretrained models with and without frozen backbone weights and biases. Relative change is measured with respect to the normal, fully trainable backbone.

As observed in Table 4.3, fully freezing the backbone resulted in the largest performance reductions for both detection and segmentation tasks: SSDLite detection accuracy dropped by 29.2%, and LR-ASPP segmentation performance decreased by 6.5%. Allowing only the biases to be updated during training, the performance reductions were decreased to 20.2% and 4.3% for SSDLite and LR-ASPP, respectively. This indicates that allowing limited adaptation, such as updating only bias parameters, can partially recover performance lost due to full backbone freezing.

This shows that while comprehensive fine-tuning provides the highest accuracy, selectively unfreezing bias terms offers a practical compromise, retaining much of the benefit of pretrained encoders while modestly improving adaptability to new tasks or domains.

4.2 Baseline

To establish a solid reference for subsequent experiments, single-task models were trained individually for each of the four selected backbone architectures. All models followed standard training protocols and were initialized with pretrained weights, as described in Section 3. The segmentation and detection results are summarized in Tables 4.4 and 4.5 respectively.

Backbone	mIoU	Pixelwise Accuracy	Parameters (M)
ResNet18	0.5103	0.8911	11.31
MobileNetV3-Large	0.5545	0.8991	3.22
EfficientNetV2-S	0.6101	0.9128	20.51
DenseNet121	0.5940	0.9074	7.22

Table 4.4: Baseline segmentation performance on the COCO dataset.

Backbone	mAP@[.50:.95]	mAP@0.5IoU	Parameters (M)
ResNet18	0.2403	0.4024	11.97
MobileNetV3-Large	0.2597	0.4278	4.03
EfficientNetV2-S	0.3142	0.4992	21.42
DenseNet121	0.2878	0.4728	8.10

Table 4.5: Baseline detection performance on the COCO dataset.

Among the evaluated backbones, EfficientNetV2-S achieved the highest segmentation (mIoU: 0.6101, Pixelwise Accuracy: 0.9128) and detection performance (mAP@[.50:.95]: 0.3142). However, this came at the cost of a substantially higher parameter count compared to the other architectures. Conversely, MobileNetV3-Large offered the most compact models (lowest parameter count), while still delivering competitive results.

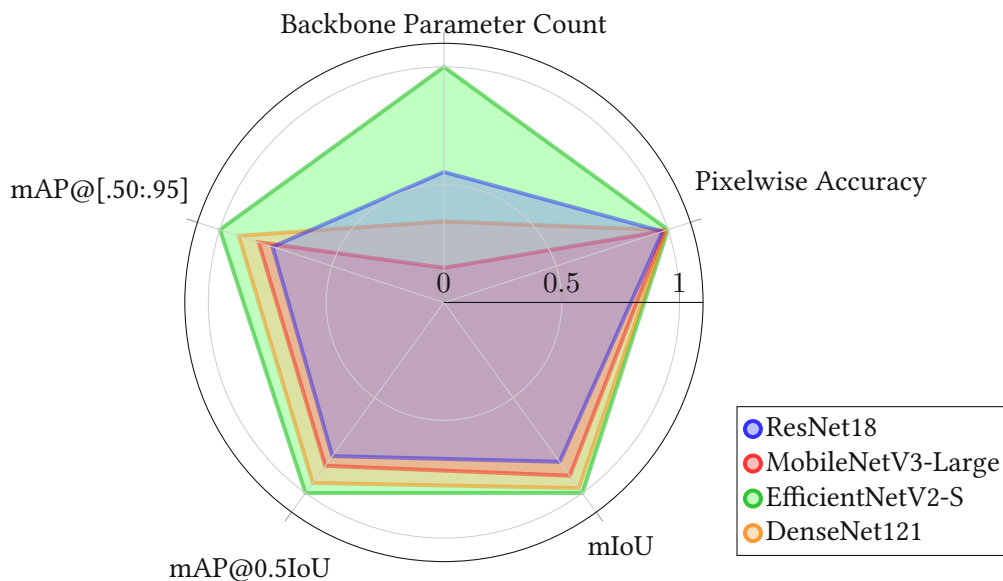


Figure 4.1: Baseline comparison, values normalized to the maximum.

Drawing on the detailed evaluation results in Table A.1, both the quality of predictions and the efficiency characteristics of each backbone, training speed, inference speed, and a compactness score as the reciprocal of the parameter count, are visually summarized in Figure 4.1. These radar plots clearly illustrate the inherent trade-off between model capacity and resource efficiency, thereby establishing a baseline against which all subsequent experimental results can be compared against.

4.3 Encoder Swapping

The results of the encoder swapping experiments are summarized in Tables 4.6 and 4.7, which report the performance of the tested backbone architectures on segmentation and detection tasks, respectively, after undergoing the encoder freezing and swapping procedure (swap-A).

Backbone	mIoU (rel. change %)	Pixelwise Accuracy (rel. change %)
ResNet18	0.4358 (-14.60)	0.8721 (-2.13)
MobileNetV3-Large	0.4649 (-16.16)	0.8768 (-2.48)
EfficientNetV2-S	0.5663 (-7.18)	0.8965 (-1.79)
DenseNet121	0.5072 (-14.61)	0.8858 (-2.38)

Table 4.6: Segmentation performance using frozen detection backbones on the COCO dataset (swap-A). Relative change compared to the baseline in %.

For the segmentation models, a clear trend emerges: all architectures experience an expected decline in performance compared to models trained end-to-end on the segmentation task. The reduction in mean Intersection-over-Union (mIoU) ranges from 7.18% (EfficientNetV2-S) to 16.16% (MobileNetV3-Large) relative to baseline, with a similar, albeit less pronounced, relative decrease observed in pixelwise accuracy. Notably, EfficientNetV2-S, the biggest model among the four, demonstrates the smallest drop in performance, suggesting that features learned during detection pretraining with this backbone may generalize more effectively to the segmentation task. Conversely, lightweight models such as MobileNetV3-Large exhibit greater sensitivity to task mismatch, consistent with their constrained capacity and stronger task specialization.

Backbone	mAP@[.50:.95] (rel. change %)	mAP@0.5IoU (rel. change %)
ResNet18	0.1970 (-18.02)	0.3314 (-17.64)
MobileNetV3-Large	0.2035 (-21.64)	0.3378 (-21.04)
EfficientNetV2-S	0.2252 (-28.33)	0.3737 (-25.14)
DenseNet121	0.2242 (-22.10)	0.3797 (-19.69)

Table 4.7: Detection performance using frozen segmentation backbones on the COCO dataset (swap-A). Relative change compared to the baseline in %.

On the detection task, all backbones also display performance degradation. Conversely to the segmentation task, the relative reduction in mAP@[.50:.95] performance is most pronounced for EfficientNetV2-S at 28.33%, while the architecturally much simpler ResNet18 shows the least drop at 18.02%. Notably, unlike in the segmentation task, where models showed consistent trends across both mIoU and pixelwise accuracy, the reduction in detection metrics is less uniform, as changes in mAP@0.5IoU do not always mirror those in the more comprehensive mAP@[.50:.95]. For instance, DenseNet121 exhibits a greater reduction in mAP@[.50:.95] compared to MobileNetV3-Large, yet a smaller reduction in mAP@0.5IoU, suggesting that for some architectures, localization performance degrades more substantially than clas-

sification ability. This pattern indicates that, while models may still successfully detect and classify objects at a coarse level, their ability to localize objects with high precision is disproportionately affected by encoder swapping, particularly in more complex architectures.

To enable targeted fine-tuning within the encoder, a secondary experiment was performed in which only the bias terms were unfrozen, allowing limited adaptation of these parameters while keeping the remainder of the encoder weights locked.

Backbone	mIoU (rel. change %)	Pixelwise Accuracy (rel. change %)
ResNet18	0.4525 (-11.33)	0.8782 (-1.45)
MobileNetV3-Large	0.4790 (-13.62)	0.8806 (-2.06)
EfficientNetV2-S	0.5844 (-4.21)	0.9023 (-1.15)
DenseNet121	0.5150 (-13.30)	0.8891 (-2.02)

Table 4.8: Segmentation performance using frozen detection backbones and unfrozen bias terms on the COCO dataset (swap-B). Relative change compared to the baseline in %.

Regarding the results in Table 4.8, all backbones show a clear improvement in segmentation performance compared to the swap-A approach. EfficientNetV2-S recovers the most, with only a 4.21% reduction in mIoU, indicating strong feature transfer and capacity to leverage minimal adaptation. Across all models, segmentation appears less sensitive to the encoder swap when bias terms are unfrozen, suggesting that even limited adaptability is particularly valuable for pixel-level tasks.

Backbone	mAP@[.50:.95] (rel. change %)	mAP@0.5IoU (rel. change %)
ResNet18	0.1967 (-18.15)	0.3297 (-18.07)
MobileNetV3-Large	0.2149 (-17.25)	0.3545 (-17.13)
EfficientNetV2-S	0.2548 (-18.91)	0.4144 (-16.99)
DenseNet121	0.2343 (-18.61)	0.3949 (-16.48)

Table 4.9: Detection performance using frozen segmentation backbones and unfrozen bias terms on the COCO dataset (swap-B). Relative change compared to the baseline in %.

Comparing the detection results in Table 4.9 shows that mAP generally improves with the swap-B approach, bringing the relative drop in performance for all backbones into a similar range, approximately 18% compared to the baseline. ResNet18 is a notable outlier, as its detection scores marginally worsen with the swap-B approach, in contrast to the consistent improvements observed for the other backbone architectures. The most substantial improvement is seen with EfficientNetV2-S, which reduces the performance drop by almost 10 percentage points.

Overall, the observed asymmetry in transfer effectiveness, with detection models experiencing a greater degradation than segmentation models, aligns with the inherent differences between the de-

tection and segmentation tasks. Encoders pretrained on detection may be better suited for adaptation to segmentation models, as their learned representations already encode strong class-discriminative features required by the segmentation head, for pixelwise classification. In contrast, segmentation-pretrained encoders primarily focus on pixel-level classification without explicit object localization, which might limit their utility when transferred to detection tasks that depend on precise localization. This distinction is reflected in the larger relative drop in strict detection metrics, such as $\text{mAP}@[.50:.95]$, compared to more lenient ones like $\text{mAP}@0.5\text{IoU}$, indicating that object localization performance is more adversely affected than classification. Importantly, the swap-B results demonstrate that enabling fine-tuning of only the bias terms helps to mitigate the drop in performance for both tasks, particularly narrowing the gaps in detection metrics, although not all architectures benefit equally.

4.4 Encoder Sharing

As with encoder swapping, the encoder sharing approach was evaluated in two variants: one with the pretrained backbones entirely frozen (share-A), and another where only the bias terms of the backbones were unfrozen to allow for selective finetuning during training (share-B). The test results for the encoder sharing strategy A are shown in Tables 4.10 and 4.11.

Backbone	mIoU (rel. change %)	Pixelwise Accuracy (rel. change %)
ResNet18	0.4340 (-14.95)	0.8642 (-3.02)
MobileNetV3-Large	0.4596 (-10.62)	0.8820 (-1.90)
EfficientNetV2-S	0.4661 (-23.60)	0.8681 (-4.90)
DenseNet121	0.5282 (-11.08)	0.8878 (-2.16)

Table 4.10: Segmentation performance using pretrained frozen backbones on the COCO dataset (share-A). Relative change compared to the baseline in %.

In the strictly frozen scenario (share-A), all models demonstrate an expected decline in segmentation performance compared to baseline, with MobileNetV3-Large exhibiting the smallest reduction in mIoU (10.62%) and EfficientNetV2-S the largest (23.60%). Interestingly, DenseNet121, while consistently performing worse than EfficientNetV2-S in both encoder swapping and baseline segmentation experiments, achieves the best results in this setting. This outcome may be attributed either to differences in the quality of the pretrained weights used or to architectural properties that make DenseNet121's features more robust to repurposing without further finetuning.

For the detection task, the results also exhibit a notable reduction in performance compared to their respective baselines across all backbones, consistent with the segmentation results. The largest relative drop in $\text{mAP}@[.50:.95]$ occurs with EfficientNetV2-S, which decreases by 39.59%, while ResNet18 experiences the smallest reduction at 23.43%. This significant decline for EfficientNetV2-S is mirrored

Backbone	mAP@[.50:.95] (rel. change %)	mAP@0.5IoU (rel. change %)
ResNet18	0.1840 (-23.43)	0.3104 (-22.86)
MobileNetV3-Large	0.1890 (-27.22)	0.3201 (-25.18)
EfficientNetV2-S	0.1898 (-39.59)	0.3354 (-32.81)
DenseNet121	0.2066 (-28.21)	0.3571 (-24.47)

Table 4.11: Detection performance using pretrained frozen backbones on the COCO dataset (share-A). Relative change compared to the baseline in %.

in its mAP@0.5IoU performance, further indicating that its ability to both localize and detect objects is particularly hindered when the encoder remains entirely frozen. DenseNet121 again stands out, achieving the highest absolute mAP@[.50:.95] (0.2066) and mAP@0.5IoU (0.3571) among the tested models, despite not performing best in encoder swapping or baseline tests. This suggests that DenseNet121’s dense connectivity may provide greater resilience to extreme transfer constraints, such as the complete freezing of an untuned encoder, allowing it to maintain stronger detection capabilities than other architectures.

Allowing limited finetuning of the encoder via unfrozen bias terms (share-B) leads to substantial improvements over the strictly frozen scenario for both tasks. The testing results are presented in the Tables 4.12 and 4.13.

Backbone	mIoU (rel. change %)	Pixelwise Accuracy (rel. change %)
ResNet18	0.4860 (-4.76)	0.8807 (-1.17)
MobileNetV3-Large	0.5265 (-5.05)	0.8910 (-0.90)
EfficientNetV2-S	0.5975 (-2.07)	0.9040 (-0.96)
DenseNet121	0.5455 (-8.16)	0.8944 (-1.43)

Table 4.12: Segmentation performance using pretrained frozen backbones and unfrozen bias terms on the COCO dataset (share-B). Relative change compared to the baseline in %.

In segmentation, all models still exhibit reduced performance drops, with EfficientNetV2-S now achieving the best results (mIoU: 0.5975, -2.07%) and showing only minimal degradation compared to end-to-end baseline. The other backbones also demonstrate notably improved performance, when compared to shared-A results, indicating that even slight adaptation in the encoder can significantly enhance the utility of pretrained or transferred weights for segmentation. The improved pixelwise accuracy across all architectures further underscores this effect.

Similar trends can be observed for detection, as the drop in both mAP@[.50:.95] and mAP@0.5IoU is decreased for all backbones compared to share-A, although the difference is not as pronounced compared to the segmentation results. EfficientNetV2-S now attains the highest absolute performance (mAP@[.50:.95]: 0.2460, mAP@0.5IoU: 0.4084), and the differences among backbones in terms of rela-

Backbone	mAP@[.50:.95] (rel. change %)	mAP@0.5IoU (rel. change %)
ResNet18	0.1923 (-19.96)	0.3261 (-18.96)
MobileNetV3-Large	0.2094 (-19.37)	0.3530 (-17.48)
EfficientNetV2-S	0.2460 (-21.71)	0.4084 (-18.19)
DenseNet121	0.2317 (-19.49)	0.3947 (-16.52)

Table 4.13: Detection performance using pretrained frozen backbones and unfrozen bias terms on the COCO dataset (share-B). Relative change compared to the baseline in %.

tive change become less pronounced. These results underscore that allowing limited flexibility in the encoder, such as enabling bias finetuning, can reduce the amount of detection performance lost due to parameter freezing. However, the degree of improvement may vary by task, with some tasks benefiting more than others from this adaptation. Overall, the share-B variant demonstrates that minimal encoder adaptation is an effective strategy to mitigate transfer-induced degradation, especially benefiting larger or more flexible architectures such as EfficientNetV2-S.

4.5 Dual Head

To evaluate the potential benefits of jointly optimizing detection and segmentation tasks within a unified architecture, the dual head models were trained and assessed using the same set of backbone encoders. The results, summarized in Tables 4.14 and 4.15, indicate a marked improvement in both segmentation and detection performance compared to baseline single-task models.

Backbone	mIoU (rel. change %)	Pixelwise Accuracy (rel. change %)
ResNet18	0.5621 (10.15)	0.9029 (1.32)
MobileNetV3-Large	0.5613 (1.23)	0.9009 (0.20)
EfficientNetV2-S	0.6452 (5.75)	0.9187 (0.65)
DenseNet121	0.6097 (2.64)	0.9118 (0.48)

Table 4.14: Segmentation performance of the dual head models on the COCO dataset. Relative change compared to the baseline in %.

For segmentation, all backbones achieve higher mIoU and pixelwise accuracy scores relative to their respective baselines, with EfficientNetV2-S again demonstrating the best performance (mIoU: 0.6452, Pixelwise Accuracy: 0.9187). Particularly notable is the substantial relative gain observed for ResNet18 (mIoU: +10.15%), highlighting how joint training can compensate for more limited model capacity and enhance feature utility for pixelwise predictions.

The detection results reflect the same trend, as all backbones see an increase in both mAP@[.50:.95] and mAP@0.5IoU, with ResNet18 (mAP@[.50:.95]: +17.44%) again achieving the most pronounced gain. A notable outlier is MobileNetV3-Large, whose performance increase for both segmentation and detection is modest compared to the other backbones, suggesting that the improvements may be constrained

Backbone	mAP@[.50:.95] (rel. change %)	mAP@0.5IoU (rel. change %)
ResNet18	0.2822 (17.44)	0.4475 (11.21)
MobileNetV3-Large	0.2737 (5.39)	0.4391 (2.64)
EfficientNetV2-S	0.3437 (9.39)	0.5213 (4.43)
DenseNet121	0.3160 (9.80)	0.4963 (4.97)

Table 4.15: Detection performance using pretrained frozen backbones on the COCO dataset (share-A). Relative change compared to the baseline in %.

by its lightweight and compact architecture.

These results demonstrate that the dual head approach not only avoids the trade-offs seen in hard encoder sharing or swapping but can actually surpass the single-task baselines across both vision tasks. The consistently positive effect of dual head training suggests that feature representations learned under joint optimization are better able to support the object localization and both instance-level, as well as fine-grained pixelwise classification required for detection and segmentation, regardless of model structure and complexity.

4.6 Embedded Device

To evaluate the actual reduction in computation overhead and compare the inference times of the different backbones, the models were deployed on an NVIDIA Jetson AGX Orin, and timing tests were performed as described in Section 3.9. Each model was exported to ONNX and deployed on the Orin device. Inference times were measured for single-task segmentation, single-task detection, and the multi-task dual head model. Tests included both FP32 for maximum accuracy and INT8 to leverage hardware-accelerated, low-precision operation.

Backbone	Segmentation	Detection	Dual Head
ResNet18	5.43	5.56	6.68
MobileNetV3-Large	5.49	5.52	7.00
EfficientNetV2-S	17.72	18.02	19.31
DenseNet121	22.37	22.40	23.83

Table 4.16: Inference times in ms on an NVIDIA Orin, FP32 mode.

Backbone	Segmentation	Detection	Dual Head
ResNet18	2.19	2.18	3.28
MobileNetV3-Large	3.25	3.32	4.44
EfficientNetV2-S	7.94	7.98	9.01
DenseNet121	10.22	10.36	11.45

Table 4.17: Inference times in ms on an NVIDIA Orin, INT8 mode.

Regarding the results presented in Table 4.16 and 4.17, the dual head architecture achieves efficient joint inference, consistently outperforming running segmentation and detection separately. INT8 quantization leads to substantial speedup for all backbones, making real-time inference feasible for demanding applications with more computationally intensive models like EfficientNetV2-S and DenseNet121, benefiting greatly from quantization and encoder sharing. However, a possible reduction in predictive performance may occur, especially for tasks requiring fine-grained accuracy, as INT8 quantization can introduce errors due to reduced numerical precision. A comprehensive list of all timing test results is provided in the appendix, in Table A.2.

Chapter 5

Analysis

5.1 Backbone Performance

Based on the results gathered in Chapter 4, the selected backbone architectures exhibit distinct differences in predictive performance, presented in Figure 5.1.

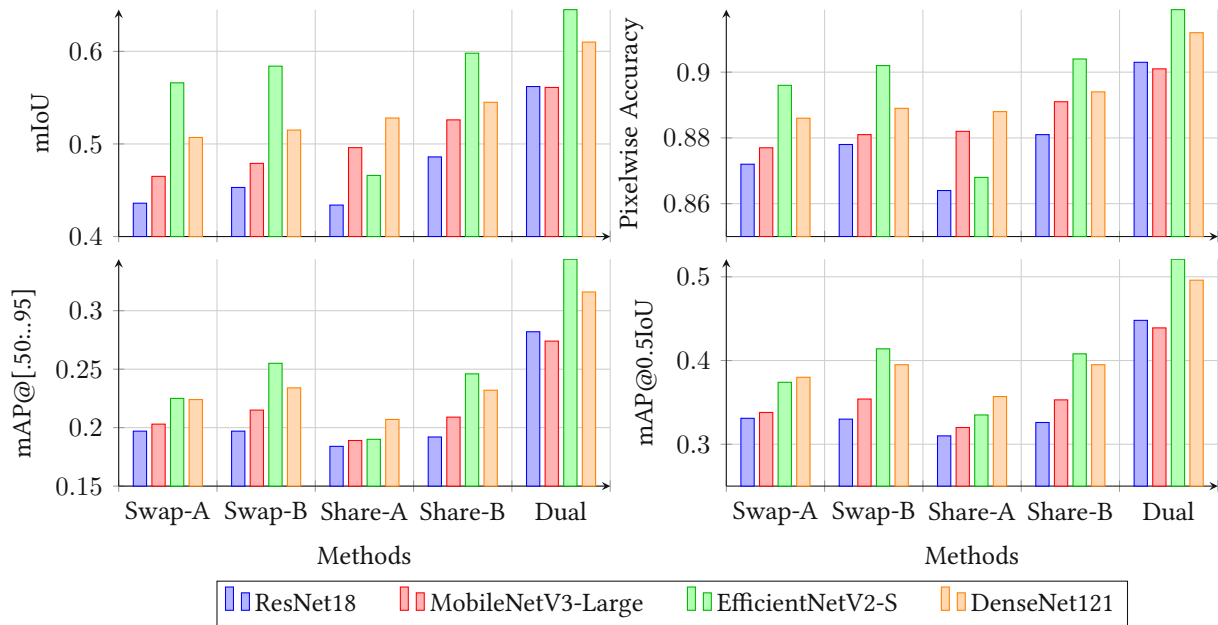


Figure 5.1: Backbone comparison for all approaches.

EfficientNetV2-S shows the strongest overall effectiveness, achieving the highest segmentation accuracy, with an mIoU of 0.6452 and pixelwise accuracy of 0.9187, and detection performance at 0.5213 mAP@0.5IoU and an mAP@[.50:.95] of 0.3437. Only falling behind DenseNet121 in the Share-A experiments. However, this comes at the cost of both a higher parameter count and slower inference, as presented in Figure 5.2, making EfficientNetV2-S the largest backbone (20.18M parameters) and, with

19.31ms inference time, much slower than MobileNetV3-Large or ResNet18. Consequently, making it less suitable for embedded or latency-constrained applications.

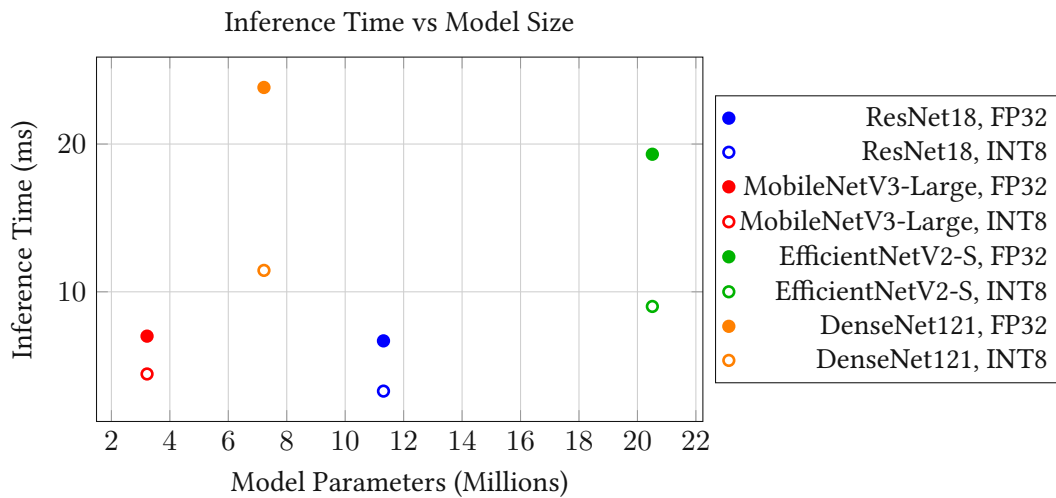


Figure 5.2: Scatter plot of inference time vs. model parameters for dual head models.

ResNet18, by contrast, shows the fastest inference speeds (6.68ms), but consistently yields the lowest predictive effectiveness among all evaluated backbones, only marginally outperforming MobileNetV3-Large in the dual-head configuration. Its limited representational capacity, relative to the more architecturally complex models, appears limited despite its computational efficiency.

DenseNet121 represents a middle ground in the comparison, striking a balance between predictive accuracy and parameter efficiency. With a model size roughly a third of that of EfficientNetV2-S and smaller than ResNet18, DenseNet121 delivers competitive results. However, it comes with the longest inference time (23.83ms), likely due to its dense connectivity pattern, which increases computational requirements despite the reduced parameter count.

MobileNetV3-Large, which outperforms ResNet18 across most approaches, stands out for its compact model size (2.97M parameters) and rapid inference (7.00ms), while still delivering robust predictive results. These characteristics make it especially well-suited for deployment in resource-constrained or real-time environments.

5.2 Approach Comparison

The experimental results from Section 4.3 to 4.5, summarized in Figure 5.3, show that the encoder swapping experiments demonstrated significant asymmetric transfer effectiveness between detection and segmentation tasks, with performance degradations ranging from 7.18% to 28.33% depending on the

backbone architecture and transfer direction. Detection-pretrained encoders consistently transferred more effectively to segmentation tasks than vice versa, reflecting the inherent differences in learned representations: detection encoders capture strong class-discriminative features suitable for pixel-wise classification, while segmentation-pretrained encoders focus primarily on spatial boundaries without explicit object localization capabilities.

The encoder sharing approach, utilizing ImageNet pretraining, showed that while generic visual representations provide a solid foundation, they require careful adaptation for optimal performance. The substantial performance drops observed in the fully frozen scenarios (Share-A) across all architectures highlight the limitations of purely generic feature representations for specialized tasks. However, the dramatic improvements achieved through minimal bias parameter adaptation (Share-B) demonstrate that even slight task-specific calibration can recover a significant portion of the lost performance.

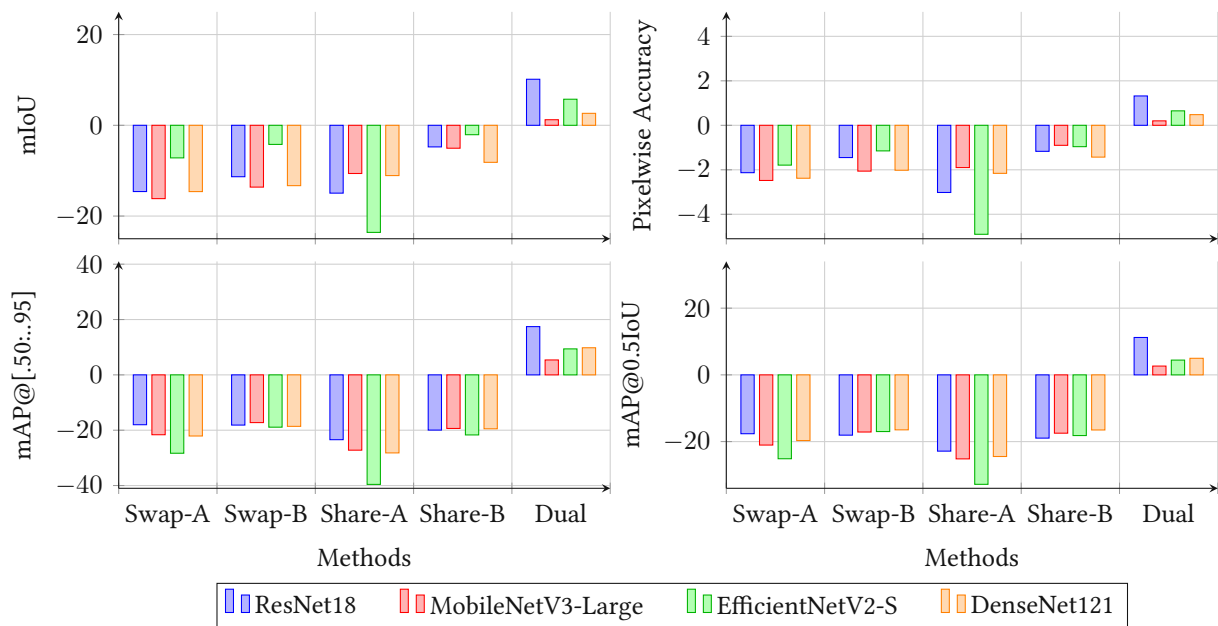


Figure 5.3: Approach comparison for all backbones. Relative change compared to the baseline in %

Most significantly, the dual head network consistently outperformed all other approaches, achieving performance improvements over single-task baselines across all tested architectures. This approach particularly benefited models with limited capacity, such as ResNet18, which achieved the most substantial relative gains (10.15% in segmentation mIoU and 17.44% in detection mAP@[.50:.95]), suggesting that multi-task learning helps overcome individual model limitations through more efficient feature utilization. These findings establish a dual- or multi-head approach as both a computational efficiency strategy as well as a mechanism for achieving superior performance compared to traditional

single-task approaches. However, multi-task learning requires access to datasets labeled for all tasks involved, which can be a major obstacle.

5.2.1 Bias Term Adaptation

The substantial performance improvements observed when transitioning from completely frozen encoders to bias-unfrozen variants in Sections 4.3 and 4.4 show the critical role of bias parameters in cross-task adaptation. Research on Large Language Models (LLMs) has shown that allowing limited fine-tuning of pretrained models leads to substantial performance gains, employing methods like inserting adapter modules [100] or Diff Pruning [101]. Specifically, Zaken et. al. propose the BitFit [102] framework, which demonstrates that fine-tuning only bias terms while freezing most of the encoder often approaches or matches full fine-tuning results. These findings can explain the dramatic performance recovery documented in the results: EfficientNetV2-S achieved a 21.53 percentage point improvement in segmentation mIoU (from -23.60% to -2.07% relative to baseline) and a 17.88 percentage point recovery in detection mAP@[.50:.95] (from -39.59% to -21.71%) when bias parameters were unfrozen. Similarly, all tested architectures showed consistent improvements, with the notable exception of ResNet18, which experienced a slight degradation in detection performance for the Swap-B approach. These re-

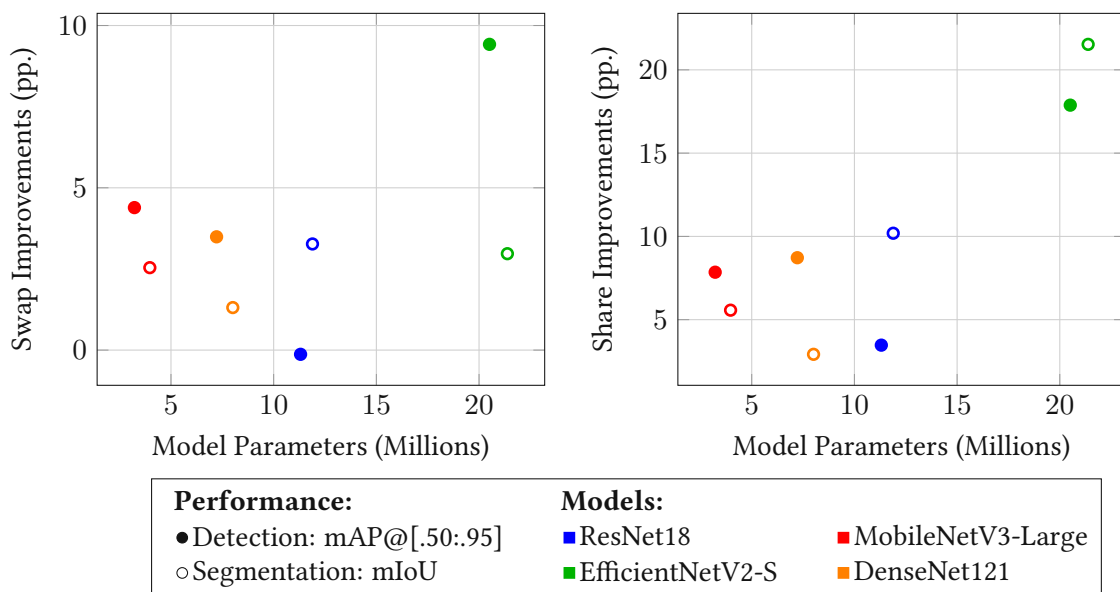


Figure 5.4: Performance improvements in percentage points between the A and B approaches for both Swap and Switch methods.

sults indicate that frozen encoder weights preserve the general visual representations learned during pre-training, while unfrozen bias parameters provide the necessary flexibility for task-specific recalibration without requiring extensive parameter updates. The thesis findings also imply architectural dependencies, as larger, more expressive models, such as EfficientNetV2-S, showed the greatest benefit

from bias adaptation. At the same time, compact architectures like MobileNetV3-Large demonstrated more modest but consistent improvements.

5.2.2 Resource Utilization

The data shown in Table 5.1 demonstrates the substantial parameter efficiency gains achieved through the dual-head architecture when compared to deploying separate detection and segmentation models.

Architecture	Backbone	Detection	Segmentation	Dual Head
ResNet18	11.18	11.76	11.31	11.89
MobileNetV3-Large	2.97	3.73	3.22	3.97
EfficientNetV2-S	20.18	21.05	20.51	21.38
DenseNet121	6.95	7.75	7.22	8.01

Table 5.1: Model parameters in millions for all configurations.

When combining the parameter counts of individual detection and segmentation models required for parallel execution, the totals range from 6.95 million parameters for MobileNetV3-Large to 41.56 million for EfficientNetV2-S. In contrast, the corresponding dual-head models require only 3.97 million and 21.38 million parameters, respectively, representing significant reductions in parameters. Overall, the parameter reduction achieved by the dual-head approach is consistent across all tested architectures, ranging from 42% to 48%, as shown in Figure 5.5.

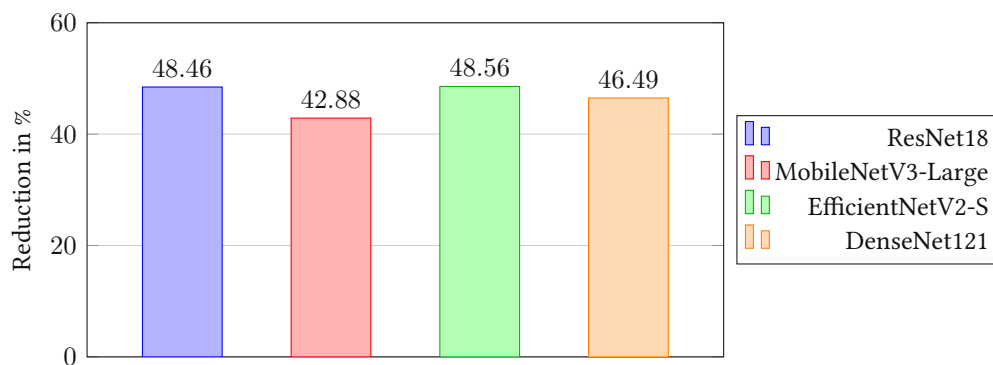


Figure 5.5: Reduction in parameter count going from separate networks to a single shared encoder model.

This consistency indicates that the efficiency gains are largely independent of the underlying backbone architecture, making the dual-head approach universally applicable across different model types and scales.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis investigated the feasibility and effectiveness of shared encoder architectures for simultaneous object detection and image segmentation with regard to resource-constrained environments. Systematic experimentation across multiple backbone architectures and sharing strategies yielded several key findings that offer both theoretical insight and practical guidance for the deployment of multi-task computer vision models.

Backbone architectures: The experimental results revealed that different backbone architectures exhibit markedly different responses to shared encoder approaches. While EfficientNetV2-S consistently demonstrated the highest absolute performance across tasks, its substantial performance degradation under frozen encoder conditions (up to 39.59% in detection $mAP@{.50:.95}$) highlighted that model capacity alone does not guarantee effective feature transferability. Conversely, architectures like DenseNet121 showed greater resilience to encoder freezing, suggesting that architectural design patterns may inherently support more robust feature sharing.

Limited adaptability: Allowing only the encoder's bias parameters to adapt during training presents a remarkably effective compromise between full fine-tuning and complete parameter freezing, recovering substantial performance while preserving computational efficiency and generalization. Across all tested architectures, unfreezing only bias terms generally recovered substantial performance losses, with improvements of up to 21.53 percentage points in segmentation mIoU and up to 17.88 percentage points in detection $mAP@{.50:.95}$. A notable exception is ResNet18 with the Swap-B approach, as it actually shows a slight degradation in performance compared to the fully frozen encoder used in Swap-A.

Dual-head: Combining the two tasks for joint training and optimization consistently achieved the best performance, surpassing single-task baselines across all architectures and metrics. ResNet18 particularly benefited from multi-task learning, achieving 17.44% improvement in detection mAP@[.50:.95] and 10.15% in segmentation mIoU, demonstrating that joint optimization can compensate for architectural limitations. However, this superior performance comes with the significant constraint of requiring datasets labeled for all target tasks, a limitation that may restrict its applicability in many real-world scenarios where such comprehensive annotations can be rare or prohibitively expensive to obtain.

Model size: Critically, parameter count alone proved to be an insufficient predictor of sharing effectiveness. Despite having the smallest parameter count (3.22M), MobileNetV3-Large often outperformed the much larger ResNet18 (11.31M parameters) in sharing scenarios, while DenseNet121 (7.22M parameters) demonstrated superior robustness to encoder freezing compared to EfficientNetV2-S (20.51M parameters). These findings underscore the importance of architectural design principles over mere model size in determining multi-task learning effectiveness.

Resource utilization: The results highlight a critical advantage for deployment scenarios where both detection and segmentation capabilities are required. Rather than maintaining separate models that would consume nearly twice the computational resources, the dual-head architecture provides both functionalities with less than 60% of the combined parameter count. This reduction is particularly significant for edge computing applications where memory constraints and power consumption are critical factors, as the approach enables comprehensive computer vision capabilities while maintaining resource efficiency.

6.2 Future Work

6.3 Bias Adaptation and Adjustment Strategies

Experiments have shown that allowing limited adaptation of the frozen encoder, by unfreezing bias parameters, can recover a substantial portion of the lost transfer performance for both the detection and the segmentation tasks. Future research should explore this way of limited fine-tuning further, systematically evaluating the effects of stacking or composing multiple bias terms, and introducing lightweight adjustment modules or transformation layers into the encoder structure. Techniques inspired by Adapter modules [100], UniAdapters [103], BitFit [102], or Low-Rank Adaptation (LoRA) [104]

can be systematically evaluated and adapted to the computer vision domain. Such techniques promise to provide improved parameter efficiency by allowing most of the encoder weights to remain frozen, while selectively allowing meaningful task-specific adaptation through a small subset of parameters, potentially bridging much of the remaining gap between full fine-tuning and completely frozen backbones.

6.4 Multi-Task Optimization and Adaptive Loss Balancing

While this thesis uses a fixed weighting system to balance the loss for the individual tasks in dual-head training, further investigation and optimization of the interplay between the different tasks' loss functions and their optimization dynamics may provide more effective training results. Approaches such as GradNorm [105], Fast Adaptive Multitask Optimization (FAMO) [106], and MetaWeighting [107] all provide flexible frameworks for dynamically adjusting the relative per-task loss weighting during multi-task training, aiming to mitigate task imbalance. Integrating such adaptive mechanisms into the training pipeline could help improve training stability and final predictive performance.

6.5 Backbone Properties

Given the considerable variation in performance changes observed across different backbone architectures in this thesis, a thorough investigation of encoder structures, including module selection, compositional details, and the locations of intermediate feature extraction, is essential for understanding multi-task learning effectiveness in shared-encoder scenarios. Identifying the specific components within the feature extraction pipeline that enhance the robustness of learned feature maps to task or dataset changes is crucial for optimizing backbone architectures in encoder sharing and transfer learning scenarios. Such insights can directly inform architectural modifications that boost overall predictive performance and resource efficiency in multi-task and adaptable machine vision systems.

6.6 Task Head Design

Similarly, as this thesis restricts each task to a single function head architecture, a comprehensive investigation into the adaptability and impact of different task-head designs presents a promising direction for future research. The structure and complexity of a task head can significantly impact how effectively shared encoder features are utilized, as well as the degree to which task-specific nuances are captured or lost during transfer learning. Certain head designs may be inherently better suited for

leveraging generic features, while others could require more specialized or fine-tuned backbones to achieve optimal results.

Bibliography

- [1] J. Janai, F. Güney, A. Behl, and A. Geiger, “Computer vision for autonomous vehicles: Problems, datasets and state of the art,” *Foundations and Trends® in Computer Graphics and Vision*, vol. 12, no. 1–3, pp. 1–308, 2020.
- [2] J. Zhang, J. Cao, J. Chang, X. Li, H. Liu, and Z. Li, “Research on the application of computer vision based on deep learning in autonomous driving technology,” Jun. 2024.
- [3] G. Sreenu and M. A. Saleem Durai, “Intelligent video surveillance: a review through deep learning techniques for crowd analysis,” *Journal of Big Data*, vol. 6, no. 1, Jun. 2019.
- [4] R. Aggarwal, V. Sounderajah, G. Martin, D. S. W. Ting, A. Karthikesalingam, D. King, H. Ashrafian, and A. Darzi, “Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis,” *npj Digital Medicine*, vol. 4, no. 1, Apr. 2021.
- [5] R. Najjar, “Redefining radiology: A review of artificial intelligence integration in medical imaging,” *Diagnostics*, vol. 13, no. 17, p. 2760, Aug. 2023.
- [6] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, “Scalable deep reinforcement learning for vision-based robotic manipulation,” in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 651–673. [Online]. Available: <https://proceedings.mlr.press/v87/kalashnikov18a.html>
- [7] F. Rosenblatt *et al.*, *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan books Washington, DC, 1962, vol. 55.
- [8] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” vol. 1, no. 4. MIT Press, Dec. 1989, pp. 541–551.

- [9] B. Graham, “Fractional max-pooling,” Dec. 2014.
- [10] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” pp. 346–361, Jun. 2014.
- [11] L. Zhao and Z. Zhang, “A improved pooling method for convolutional neural networks,” *Scientific Reports*, vol. 14, no. 1, Jan. 2024.
- [12] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.
- [13] S. Hochreiter, “Hochreiter, sepp,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.
- [14] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, “Dying relu and initialization: Theory and numerical examples,” *Communications in Computational Physics*, vol. 28, no. 5, pp. 1671–1706, Jan. 2019.
- [15] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.
- [16] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” Feb. 2015.
- [17] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” Nov. 2015.
- [18] D. Hendrycks and K. Gimpel, “Gaussian error linear units (gelus),” Jun. 2016.
- [19] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” Oct. 2017.
- [20] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” Feb. 2015.
- [21] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” Jul. 2016.
- [22] D. Ulyanov, A. Vedaldi, and V. Lempitsky, “Instance normalization: The missing ingredient for fast stylization,” Jul. 2016.
- [23] Y. Wu and K. He, “Group normalization,” Mar. 2018.
- [24] S. Ruder, “An overview of gradient descent optimization algorithms,” Sep. 2016.
- [25] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” Dec. 2014.

- [26] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 58, no. 1, pp. 267–288, Jan. 1996.
- [27] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, Feb. 1970.
- [28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [29] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," May 2014.
- [30] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*. IEEE, 1999, pp. 1150–1157 vol.2.
- [31] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, ser. CVPR-01, vol. 1. IEEE Comput. Soc, pp. I-511–I-518.
- [32] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, pp. 886–893.
- [33] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Jun. 2008, pp. 1–8.
- [34] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.
- [35] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, Apr. 2013.
- [36] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.
- [37] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," Jun. 2015.

- [38] Z. Cai and N. Vasconcelos, “Cascade r-cnn: Delving into high quality object detection,” Dec. 2017.
- [39] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” pp. 21–37, Dec. 2015.
- [40] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” Jun. 2015.
- [41] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” Dec. 2016.
- [42] —, “Yolov3: An incremental improvement,” Apr. 2018.
- [43] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” Apr. 2020.
- [44] G. Jocher, L. Changyu, A. Hogan, Lijun Yu, Changyu98, P. Rai, and T. Sullivan, “ultralytics/yolov5: Initial release,” 2020.
- [45] C. Li, L. Li, Y. Geng, H. Jiang, M. Cheng, B. Zhang, Z. Ke, X. Xu, and X. Chu, “Yolov6 v3.0: A full-scale reloading,” Jan. 2023.
- [46] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” Jul. 2022.
- [47] R. Varghese and S. M., “Yolov8: A novel object detection algorithm with enhanced performance and robustness,” in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*. IEEE, Apr. 2024, pp. 1–6.
- [48] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, “Yolov9: Learning what you want to learn using programmable gradient information,” Feb. 2024.
- [49] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding, “Yolov10: Real-time end-to-end object detection,” in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 107 984–108 011. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/c34ddd05eb089991f06f3c5dc36836e0-Paper-Conference.pdf
- [50] R. Khanam and M. Hussain, “Yolov11: An overview of the key architectural enhancements,” Oct. 2024.
- [51] Y. Tian, Q. Ye, and D. Doermann, “Yolov12: Attention-centric real-time object detectors,” Feb. 2025.

- [52] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," May 2015.
- [53] M. Z. Alom, M. Hasan, C. Yakopcic, T. M. Taha, and V. K. Asari, "Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation," Feb. 2018.
- [54] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation," Jul. 2018.
- [55] N. Ibtehaz and M. S. Rahman, "Multiresunet : Rethinking the u-net architecture for multimodal biomedical image segmentation," *Neural Networks*, vol. 121, pp. 74–87, Jan. 2019.
- [56] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," Dec. 2014.
- [57] —, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," Jun. 2016.
- [58] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," Jun. 2017.
- [59] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "Bisenet: Bilateral segmentation network for real-time semantic segmentation," Aug. 2018.
- [60] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, "Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation," Apr. 2020.
- [61] G. Rosi, C. Cuttano, N. Cavagnero, G. Averta, and F. Cermelli, "The revenge of bisenet: Efficient multi-task image segmentation," Apr. 2024.
- [62] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," Mar. 2017.
- [63] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: <https://github.com/ultralytics/ultralytics>
- [64] G. Sistu, I. Leang, and S. Yogamani, "Real-time joint object detection and semantic segmentation network for automated driving," Jan. 2019.
- [65] J. Gong, W. Liu, M. Pei, C. Wu, and L. Guo, "Resnet10: A lightweight residual network for remote sensing image classification," in *2022 14th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*. IEEE, Jan. 2022, pp. 975–978.

- [66] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.
- [67] Z. Nan, J. Peng, J. Jiang, H. Chen, B. Yang, J. Xin, and N. Zheng, "A joint object detection and semantic segmentation model with cross-attention and inner-attention mechanisms," *Neurocomputing*, vol. 463, pp. 212–225, Nov. 2021.
- [68] S. Abdigapporov, S. Miraliev, V. Kakani, and H. Kim, "Joint multiclass object detection and semantic segmentation for autonomous driving," *IEEE Access*, vol. 11, pp. 37 637–37 649, 2023.
- [69] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2020, pp. 390–391.
- [70] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.
- [71] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.
- [72] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- [73] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," May 2020.
- [74] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable detr: Deformable transformers for end-to-end object detection," Oct. 2020.
- [75] H. Zhang, F. Li, S. Liu, L. Zhang, H. Su, J. Zhu, L. M. Ni, and H.-Y. Shum, "Dino: Detr with improved denoising anchor boxes for end-to-end object detection," Mar. 2022.
- [76] J. Chen, Y. Lu, Q. Yu, X. Luo, E. Adeli, Y. Wang, L. Lu, A. L. Yuille, and Y. Zhou, "Transunet: Transformers make strong encoders for medical image segmentation," Feb. 2021.

- [77] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, “Segformer: Simple and efficient design for semantic segmentation with transformers,” May 2021.
- [78] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, “Segment anything,” Apr. 2023.
- [79] F. Li, H. Zhang, H. xu, S. Liu, L. Zhang, L. M. Ni, and H.-Y. Shum, “Mask dino: Towards a unified transformer-based framework for object detection and segmentation,” Jun. 2022.
- [80] X. Xu, H. Zhao, V. Vineet, S.-N. Lim, and A. Torralba, *MTFormer: Multi-task Learning via Transformer and Cross-Task Reasoning*. Springer Nature Switzerland, 2022, pp. 304–321.
- [81] Y. Xu, Y. Yang, and L. Zhang, “Demt: Deformable mixer transformer for multi-task learning of dense prediction,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 3, pp. 3072–3080, Jun. 2023.
- [82] A. Khan, Z. Rauf, A. Sohail, A. R. Khan, H. Asif, A. Asif, and U. Farooq, “A survey of the vision transformers and their cnn-transformer based variants,” *Artificial Intelligence Review*, vol. 56, no. S3, pp. 2917–2970, Oct. 2023.
- [83] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” Oct. 2020.
- [84] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. K. Luk, B. Maher, Y. Pan, C. Puhersch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, S. Zhang, M. Suo, P. Tillet, X. Zhao, E. Wang, K. Zhou, R. Zou, X. Wang, A. Mathews, W. Wen, G. Chanan, P. Wu, and S. Chintala, “Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS ’24. ACM, Apr. 2024, pp. 929–947.
- [85] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Sep. 2009.
- [86] A. Krizhevsky, “Learning multiple layers of features from tiny images,” 2009.

- [87] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” Dec. 2015.
- [88] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, “Searching for mobilenetv3,” May 2019.
- [89] R. Avenash and P. Viswanath, “Semantic segmentation of satellite images using a modified cnn with hard-swish activation function,” in *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. SCITEPRESS - Science and Technology Publications, 2019.
- [90] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” Aug. 2016.
- [91] M. Tan and Q. V. Le, “Efficientnetv2: Smaller models and faster training,” *International Conference on Machine Learning, 2021*, Apr. 2021.
- [92] —, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *International Conference on Machine Learning, 2019*, May 2019.
- [93] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018*, pp. 4510-4520, Jan. 2018.
- [94] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” Sep. 2014.
- [95] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, “Supervised contrastive learning,” Apr. 2020.
- [96] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” Feb. 2020.
- [97] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Jun. 2009.
- [98] Jaccard, Paul, “Lois de distribution florale dans la zone alpine,” 1902.
- [99] NVIDIA. Nvidia jetson orin. Accessed: 2025-08-30. [Online]. Available: <https://www.nvidia.com/en-eu/autonomous-machines/embedded-systems/jetson-orin/>

- [100] N. Houlsby, A. Giurigu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, “Parameter-efficient transfer learning for nlp,” Feb. 2019.
- [101] D. Guo, A. M. Rush, and Y. Kim, “Parameter-efficient transfer learning with diff pruning,” Dec. 2020.
- [102] E. B. Zaken, S. Ravfogel, and Y. Goldberg, “Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models,” Jun. 2021.
- [103] H. Lu, Y. Huo, G. Yang, Z. Lu, W. Zhan, M. Tomizuka, and M. Ding, “Uniadapter: Unified parameter-efficient transfer learning for cross-modal modeling,” Feb. 2023.
- [104] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” Jun. 2021.
- [105] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, “GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks,” in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 794–803. [Online]. Available: <https://proceedings.mlr.press/v80/chen18a.html>
- [106] B. Liu, Y. Feng, P. Stone, and Q. Liu, “Famo: Fast adaptive multitask optimization,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 57 226–57 243. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/b2fe1ee8d936ac08dd26f2ff58986c8f-Paper-Conference.pdf
- [107] Y. Mao, Z. Wang, W. Liu, X. Lin, and P. Xie, “Metaweighting: Learning to weight tasks in multi-task learning,” in *Findings of the Association for Computational Linguistics: ACL 2022*. Association for Computational Linguistics, 2022, pp. 3436–3448.

Appendix A

Appendix

backbone	method	mIoU	pixel acc.	mAP[.50:95]	mAP-50	mAP-55	mAP-60	mAP-65	mAP-70	mAP-75	mAP-80	mAP-85	mAP-90	mAP-95
ResNet18	Baseline	0.5103	0.8911	0.2403	0.4024	0.3824	0.3549	0.3245	0.2893	0.2453	0.197	0.1305	0.0641	0.0126
ResNet18	Swap-A	0.4358	0.8721	0.197	0.3314	0.3145	0.2945	0.2672	0.2327	0.1892	0.149	0.1082	0.0601	0.0234
ResNet18	Swap-B	0.4525	0.8782	0.1967	0.3297	0.3106	0.2926	0.2646	0.2309	0.1942	0.1524	0.1029	0.0704	0.0185
ResNet18	Share-A	0.434	0.8642	0.184	0.3104	0.2952	0.276	0.2449	0.2159	0.1757	0.1386	0.0988	0.056	0.0288
ResNet18	Share-B	0.486	0.8807	0.1923	0.3261	0.3107	0.2895	0.2617	0.2286	0.1868	0.1435	0.0905	0.0552	0.0305
ResNet18	Dual	0.5621	0.9029	0.2822	0.4475	0.4243	0.3996	0.3707	0.3361	0.2927	0.2385	0.175	0.1049	0.0328
MobileNetV3-Large	Baseline	0.5545	0.8991	0.2597	0.4278	0.4042	0.3773	0.3481	0.3079	0.2605	0.2123	0.157	0.0808	0.0214
MobileNetV3-Large	Swap-A	0.4649	0.8768	0.2035	0.3378	0.3201	0.3021	0.2742	0.2394	0.1993	0.1567	0.1123	0.0665	0.0258
MobileNetV3-Large	Swap-B	0.479	0.8806	0.2149	0.3545	0.3376	0.318	0.2884	0.2556	0.2115	0.1636	0.1214	0.0698	0.0291
MobileNetV3-Large	Share-A	0.4956	0.882	0.189	0.3201	0.3031	0.2821	0.2514	0.2181	0.1858	0.1438	0.0989	0.0596	0.0268
MobileNetV3-Large	Share-B	0.5265	0.891	0.2094	0.353	0.3342	0.3127	0.2838	0.2478	0.201	0.1573	0.1077	0.067	0.0295
MobileNetV3-Large	Dual	0.5613	0.9009	0.2737	0.4391	0.4158	0.3872	0.3583	0.3251	0.2751	0.2267	0.1726	0.1088	0.0283
EfficientNetV2-S	Baseline	0.6101	0.9128	0.3142	0.4992	0.4733	0.443	0.4105	0.3736	0.3267	0.2755	0.202	0.1114	0.0271
EfficientNetV2-S	Swap-A	0.5663	0.8965	0.2252	0.3737	0.355	0.3305	0.3008	0.263	0.22	0.1748	0.1307	0.0776	0.0256
EfficientNetV2-S	Swap-B	0.5844	0.9023	0.2548	0.4144	0.3942	0.3719	0.3429	0.3076	0.2593	0.2055	0.1406	0.0817	0.0304
EfficientNetV2-S	Share-A	0.4661	0.8681	0.1898	0.3354	0.3149	0.2896	0.2559	0.2205	0.1808	0.1362	0.0872	0.0536	0.0237
EfficientNetV2-S	Share-B	0.5975	0.904	0.246	0.4084	0.3875	0.3643	0.3345	0.2968	0.2452	0.1876	0.1316	0.0778	0.0263
EfficientNetV2-S	Dual	0.6452	0.9187	0.3437	0.5213	0.4986	0.4702	0.4412	0.4002	0.3526	0.3075	0.2386	0.1583	0.0485
DenseNet121	Baseline	0.594	0.9074	0.2878	0.4728	0.4468	0.4196	0.3857	0.3421	0.2945	0.2413	0.1688	0.0841	0.0228
DenseNet121	Swap-A	0.5072	0.8858	0.2242	0.3797	0.359	0.3331	0.3025	0.2667	0.2218	0.1723	0.1168	0.0694	0.021
DenseNet121	Swap-B	0.515	0.8891	0.2343	0.3949	0.3744	0.3482	0.3202	0.2827	0.2329	0.1783	0.1225	0.0728	0.0158
DenseNet121	Share-A	0.5282	0.8878	0.2066	0.3571	0.3361	0.3126	0.2825	0.2452	0.2011	0.1454	0.1003	0.0584	0.0278
DenseNet121	Share-B	0.5455	0.8944	0.2317	0.3947	0.3751	0.3508	0.3186	0.2774	0.2291	0.1747	0.1137	0.0606	0.022
DenseNet121	Dual	0.6097	0.9118	0.316	0.4963	0.4725	0.4433	0.4124	0.3746	0.3286	0.2722	0.2041	0.1173	0.0387

Table A.1: Comprehensive results for all backbone variations and experimental configurations.

Backbone	Encoder		Segmentation		Detection		Dual Head	
	TP	Lat.	TP	Lat.	TP	Lat.	TP	GPU
ResNet18	237.343	4.420	208.685	5.429	188.422	5.564	170.391	6.684
ResNet18 - int8	837.175	1.394	640.684	2.186	517.620	2.177	396.131	3.280
MobileNetV3-Large	257.229	4.154	217.522	5.494	192.053	5.524	168.511	6.998
MobileNetV3-Large - int8	453.467	2.432	385.936	3.245	326.783	3.318	274.104	4.437
EfficientNetV2-S	60.960	16.688	58.629	17.720	56.036	18.019	54.196	19.311
EfficientNetV2-S - int8	-*	-*	137.005	7.936	129.282	7.976	121.509	9.008
DenseNet121	47.435	21.345	45.797	22.370	44.972	22.395	43.171	23.831
DenseNet121 - int8	89.813	11.375	103.911	10.221	98.382	10.362	93.234	11.454
ResNet18	237.343	4.420	208.685	5.429	188.422	5.564	170.391	6.684
ResNet18 - int8	837.175	1.394	640.684	2.186	517.620	2.177	396.131	3.280
MobileNetV3-Large	257.229	4.154	217.522	5.494	192.053	5.524	168.511	6.998
MobileNetV3-Large - int8	453.467	2.432	385.936	3.245	326.783	3.318	274.104	4.437
EfficientNetV2-S	60.960	16.688	58.629	17.720	56.036	18.019	54.196	19.311
EfficientNetV2-S - int8	-*	-*	137.005	7.936	129.282	7.976	121.509	9.008
DenseNet121	47.435	21.345	45.797	22.370	44.972	22.395	43.171	23.831
DenseNet121 - int8	89.813	11.375	103.911	10.221	98.382	10.362	93.234	11.454

*Could not be tested due to TensorRT incompatibility

Table A.2: Timing results for all backbones and model variations. Throughput (TP) in qps, Latency (Lat.) in ms, GPU-time (GPU) in ms.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

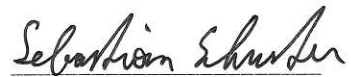
Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Copyright Statement

I, Sebastian Schuster, BSc, hereby declare that this thesis is my own original work and, to the best of my knowledge and belief, it does not:

- Breach copyright or other intellectual property rights of a third party.
- Contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
- Contain material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.
- Contain substantial portions of third party copyright material, including but not limited to charts, diagrams, graphs, photographs or maps, or in instances where it does, I have obtained permission to use such material and allow it to be made accessible worldwide via the Internet.

Signature:



Vienna, Austria, September 2025

Sebastian Schuster, BSc