POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Politecnico di Torino

Master's Degree Thesis

Slimmable and Early Exit Neural Networks for Object Detection on Nano-Drones

Supervisors

Prof. Daniele Jahier PAGLIARI

Prof. Alessio BURRELLO

Beatrice Alessandra MOTETTI

Candidate

Carlo MARRA

Academic Year 2025-2026

Abstract

Deploying deep learning on nano–drone platforms imposes strict constraints on latency, memory, and energy. This thesis investigates the use of dynamic inference solutions to tackle this problem, focusing in particular on object detection, a common task in many applications such as navigation, obstacle avoidance, search and rescue, etc.

The proposed approach modifies a MobileNetV2 SSDLite (input 512×512) into a so-called slimmable model, where four width configurations ($0.25\times$, $0.5\times$, $0.75\times$, $1.0\times$) can be dynamically selected for layers after the 6^{th} feature extractor block. The four widths share a single set of weights, except for width-private batch normalization statistics, thus incurring a minimal memory overhead with respect to the original model. Width selection can be performed on a per-sample basis, for example depending on external conditions, such as remaining battery life or expected task difficulty.

In addition to this multi-width operation, the model also supports an adaptive mode, where a binary gate classifier is added at the end of the fixed backbone, trained to detect empty or trivial frames and trigger an early-exit strategy, further reducing computation without excessive performance loss.

Training is divided into two phases. In the first phase, the slimmable detector is optimized across all widths using in-place ensemble bootstrapping with an EMA teacher and knowledge distillation. This strategy ensures stable convergence and preserves accuracy at every configuration. In the second phase, the gate is trained for adaptive inference. By learning to discriminate between empty and labeled images, the gate reduces the computation effort for the former, by either directly returning an empty prediction, or by forcing the execution of the slimmest head, which can potentially correct false negatives.

For evaluation, we use a Cityscapes-derived detection benchmark: each 2048×1024 image is converted to COCO format and split into eight non-overlapping 512×512 tiles, with extensive photometric and geometric augmentations.

As a reference, a non-slimmable MobileNetV2 SSDLite-512, which retains the same gate as the slimmable model but is fixed to the full width, achieves 21.57 mAP at a computational cost of 1.76 GMAC per forward pass. Our slimmable model

spans a smooth accuracy–efficiency frontier: at 1.0× it reaches mAP 21.96 at 1.44 GMAC/forward; at 0.75× it attains mAP 14.12 at 0.94 GMAC (−34.7% vs 1.0×); and at 0.25× it yields mAP 2.09 at 0.38 GMAC (−73.6%). The adaptive mode, leveraging the trained gate, skips 1884/4000 images (47.1%), correctly flags about 57% of empty images, and maintains mAP 16.64 while reducing the average cost to 0.85 GMAC (−41.0% vs 1.0×).

Overall, the proposed detector combines the flexibility of slimmable design with adaptive early-exit strategies. It allows instant switching between operating points, either user-controlled or automatically chosen, and achieves substantial compute savings in real-time drone workloads with controlled accuracy loss. Results indicate that slimmable and input-adaptive networks offer a unified and practical approach to deploying deep learning on resource-limited platforms.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

AP
Average Precision

BN
Batch Normalization

CNN
Convolutional Neural Network

DL
Deep Learning

DNN
Deep Neural Network

FSSD
Feature Fusion Single Shot Multibox Detector

IoT
Internet of Thing

IoU
Intersection over Union

KD
Knowledge Distillation

MAC

Multiply-and-Accumulate

mAP

Mean Average Precision

ML

Machine Learning

MoI

Manifold of Interest

NAS

Neural Architecture Search

NMS

Non-Maximum Suppression

RoI

Region of Interest

RPN

Region Proposal Network

SGS

Sandwich Gate Sparsification

SNN

Slimmable Neural Network

SoA

State of the Art

SSD

Single-Shot MultiBox Detector

Chapter 1

Introduction

In recent years, the need to deploy deep learning (DL) models on mobile and embedded devices has led to growing interest in efficient neural networks capable of adapting to strict constraints in memory, computational power, and energy consumption. Within this context, Slimmable Neural Networks (SNNs) have emerged as a family of architectures designed to operate at variable widths, i.e., with a different number of active channels per layer. This property not only makes it possible to dynamically balance accuracy and computational complexity, but also to eliminate the need to train and maintain multiple distinct models for different usage scenarios. SNNs therefore aim to address some of the key challenges associated with running neural networks in real-world resource-costrained environments, which are the variability of resources across devices, fluctuations in energy budgets on the same device and the overhead of distributing and managing multiple models, by allowing instantaneous switching between "lightweight" and relatively "heavy" configurations.

The concept of slimmable networks was first introduced by Yu et al. (2018) [1], who coined the term "Slimmable Neural Networks" for models that can run at multiple widths. From there, SNNs have matured from a niche idea into a widely used strategy in efficient deep learning. As of 2025, the SoA efficient models for many tasks leverage the principles of slimmability and one-shot training. For instance, leading image classification models optimized for mobile devices often come from weight-sharing NAS techniques akin to OFA [2] or BigNAS [3], which can be essentially seen as improved descendants of slimmable networks. In object detection, it's not uncommon to see frameworks where a single backbone can be toggled between a fast mode and an accurate mode. Moreover, the concept has been integrated into various AutoML [4] toolkits and hardware-aware model deployment pipelines, indicating broad adoption. The research community has also extended slimmable ideas to transformers for vision (ViT) [5], creating slimmable vision transformers that can drop heads or layers on demand, which are becoming relevant

in vision applications that need the adaptivity.
In this work, we focus on building an object detection model tailored for resource-constrained drones. Our design takes inspiration from the Dynamic Slimmable Network (DS-Net) proposed by Li et al. [6], which extends slimmable networks with input-dependent gating. Whereas traditional SNNs assume that the width is chosen statically for a given deployment scenario, DS-Net introduced the idea of dynamically adjusting width on a per-input basis, thereby saving computation on easier images while preserving accuracy on harder ones.
Building on this principle, we develop a slimmable version of MobileNetV2 SSDLite-512 [7] equipped with a channel-gating mechanism that enables operation at four widths (0.25×, 0.5×, 0.75×, and 1.0×), each providing a distinct balance between accuracy and efficiency. The active width can be selected according to external factors independent of the input, such as remaining battery or estimated operational time.
In addition to multi-width operation, we introduce an adaptive mode that leverages the same gating mechanism to identify simple or empty images and dynamically adjust computation. Relative to the full-width (1.00×) configuration, the default *Try-Best* setting processes such images with the slimmest sub-network, reducing cost by 35.5% (0.93 GMACs vs. 1.44 GMACs) while maintaining 16.90% mAP (−23.0%). The more aggressive *Early-Exit* variant instead skips these images altogether, achieving a 41.0% cost reduction (0.85 GMACs per image) with a comparable accuracy of 16.64% mAP (−24.2%). This trade-off demonstrates the suitability of the proposed detector for real-world drone applications, where efficiency and responsiveness are critical.
In general, the thesis is organized as follows:

- Chapter 2 provides the background for this work, starting with an overview of the MobileNet V2 architecture and its core design principles, followed by the main concepts of object detection, knowledge distillation, and slimmable neural networks.
- Chapter 3 reviews the most relevant related works on SNNs, highlighting their key ideas, challenges, and contributions.
- Chapter 4 details the methods and techniques adopted in this thesis. It explains the architectural choices, the training procedure, and the specific strategies used to implement and evaluate the proposed model.
- Chapter 5 presents the experimental results. It provides both a critical analysis and visualizations of the most significant outcomes achieved during the project.
- Chapter 6 concludes the thesis by summarizing the main findings and outlining possible directions for future work.

Chapter 2

Background

2.1 Mobilenet V2

MobileNet is a family of convolutional neural network architectures (CNNs) designed for mobile and embedded vision applications, known for their limited size and low latency [7, 8]. Made in response to a growing demand for efficient models with competitive accuracy, the principle behind these architectures is balancing perfomance and the memory requirements.

The first generation of MobileNet, MobileNetV1 [9], was published in 2017. From an architectural point of view, the main innovation was the heavy use of depthwise separable convolutions [10], resulting in a reduction of the number of parameters and operations by almost an order of magnitude when compared to standard convolutions.

MobileNetV2 [7, 8] was released a year later, introducing inverted residual blocks and linear bottlenecks, improving both efficiency and capacity of representation. This big step forward allowed the MobileNetV2 to be a widely adopted backbone for various computer vision tasks, ranging from image classification to segmentation [11, 12].

Later releases, namely MobileNetV3 (2019) [13] and MobileNetV4 (2023) [14] further refined this design philosophy, incorporating techniques like hardware-aware neural architecture search (NAS), complemented by algorithms like NetAdapt [15] to tune performance for mobile CPUs, and attention mechanism [16]. As of today, MobileNetV2 is still considered a pivotal step in the family, as it shown in Figure 2.1, finding an effective balance between computational efficiency and accuracy.

As already stated, the MobileNetV2 was proposed by Sandler et al. as a refinement of the original MobileNet architecture. One of the major drawbacks of MobileNetV1 was its representational capacity, particularly when compared to larger, full-scale

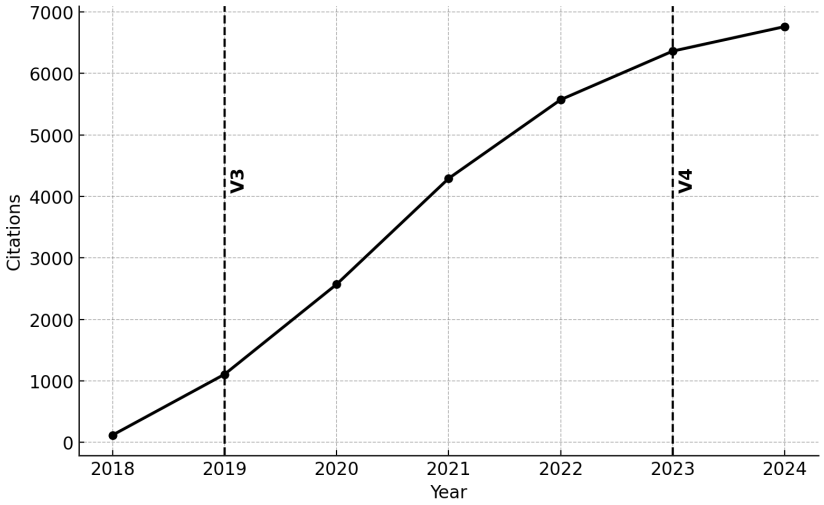


Figure 2.1: Citations per year of MobileNetV2, with the release of MobileNetV3 (2019) and MobileNetV4 (2023) indicated.

CNNs. In particular, MobileNetV1 struggled with effectively reusing features across layers, a crucial property that granted the success of deeper architectures such as ResNet [17]. The design challenge for MobileNetV2 was therefore to keep the computational efficiency of V1 while introducing mechanisms to recover part of the lost accuracy.
In the following subsections, the main properties and architectural components of MobileNetV2 are presented. In particular, Subsection 2.1.2 explains the concept of the linear bottleneck, while Subsection 2.1.3 details the inverted residual connection and its motivations. Finally, the overall network architecture is summarized in Subsection 2.1.4.

2.1.1 Manifold of Interest

Modern CNNs produce, for each layer L_i, an activation tensor with shape $h_i \times w_i \times d_i$, where h_i and w_i denote the spatial dimensions (height and width of the feature map), and d_i the number of channels. What Sandler et al. observed is that each spatial position can be represented as a d_i-dimensional vector. Across different inputs, these vectors do not uniformly fill the whole space $\mathbb{R}^{d_i}$, but rather concentrate on a subset of it, the so-called "manifold of interest" (MoI).
From an empirical point of view, the MoI captures the statistical regularities of images and the invariances learned by the network. Although each activation lives in a d_i-dimensional space, the data actually occupy only a much smaller subset of it. In other words, the representational capacity of the network is usually larger than

what is effectively needed, which is why the MoI tends to have a lower intrinsic dimension than d_i. This observation motivated the channel compression strategy adopted in MobileNetV1, implemented through the width multiplier parameter α. In other words, the linear transformations performed by convolutional layers can preserve the essential information of the manifold while, at the same time, reducing memory usage and computational cost.
However, what was observed by the authors is that compression alone is not enough, since CNNs are not purely linear systems. After a linear transformation, infact, all activations pass through a pointwise nonlinearity (e.g. ReLU [18]).
The ReLU activation function (Figure 2.2), for example, sets all negative input values to zero. Geometrically, this means that the activation space is partitioned into regions depending on which channels remain positive (the so-called "active" channels). Within each region the layer behaves like a linear transformation, and the nonlinearity only appears at the boundaries between regions.

$$\text{ReLU}(x) = x^+ = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

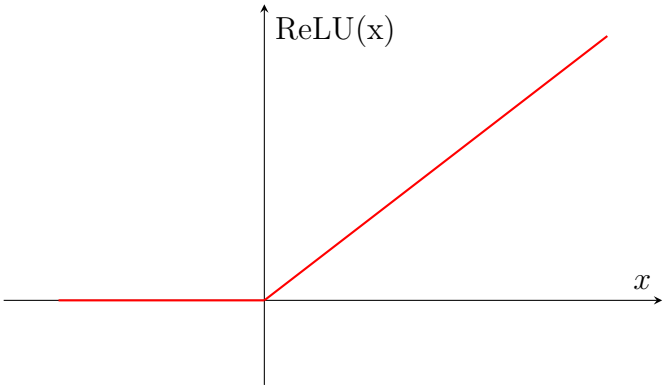


Figure 2.2: The ReLU (Rectified Linear Unit) activation function.

Sandler et al. found out that compressing the representation first and then applying ReLU, entire pieces of information can be suppressed, irreversibly collapsing parts of the MoI.
As Figure 2.3 illustrates, the ReLU activation can distort low-dimensional representations, potentially causing a loss of information. The example shows an input MoI (a spiral) undergoing ReLU transformations when projected into different output dimensions (with dimensionality equal to 2, 3, 15, and 30). In low-dimensional cases, such as the ones with dimension equal to 2 and to 3, the spiral becomes compressed and distorted, leading to a collapse of certain regions of the manifold.

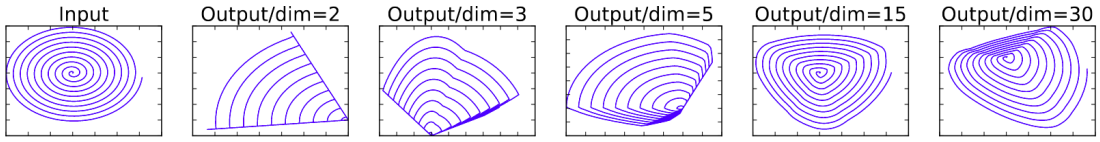


Figure 2.3: Examples of ReLU transformations of low-dimensional manifolds embedded in higher-dimensional spaces [7].

This results in significant loss of information in the learned representation (i.e., the activations). By contrast, when the output dimensionality is higher, such as with dimension equal to 15 or to 30, the transformation preserves the structure of the manifold much more effectively, and the information loss is greatly reduced.

2.1.2 Linear Bottleneck

MobileNetV1 manages to reduce the computation with depthwise separable convolutions and width multipliers, but an aggressive shrinking degraded capacity of representation when nonlinear transformations acted in the narrow space.
MobileNetV2 solves this issue by moving the nonlinearity to higher-dimensional spaces and keeping the compression step linear, avoiding destructive clipping. From an architectural point of view, each of the building blocks follows the pattern:

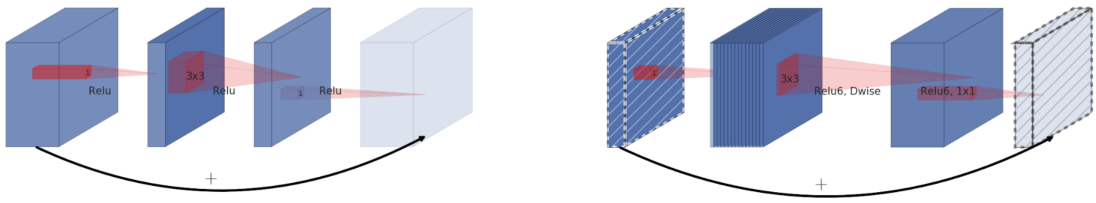
$$\text{Expansion} \rightarrow \text{Depthwise Convolution} \rightarrow \text{Projection}$$

- **Expansion:** A 1×1 convolution expands the channels from d_{in} to $t\cdot d_{\text{in}}$ with $t > 1$ (typically $t \approx 6$), followed by batch normalization and a bounded activation. This creates a higher-dimensional space where nonlinear transformations are less likely to collapse the MoI.
- **Depthwise convolution:** A depthwise $k \times k$ convolution followed by batch normalization (BN) and a ReLU6.
- **Projection:** A 1×1 convolution brings the features back to d_{out}. It is fundamental to note that no nonlinear transformations are applied after this phase.

In the original design, the authors employed the ReLU6 activation function, a variant of ReLU that clips outputs to the range $[0, 6]$. This bounded version was shown to be more robust in low-precision computations commonly used on mobile devices [19, 20].
The Projection stage and its relative output are usually refered as "linear bottleneck". The term “linear” emphasizes the absence of any non-linear activation function at this narrowest point.

2.1.3 Inverted Residual

In classical architectures such as ResNet [17], residual connections are placed between layers with a large number of channels, so that the shortcut links wide feature maps. MobileNetV2 takes the opposite approach: the shortcut is applied at the narrow bottleneck rather than the expanded representation. The intuition is that the bottleneck already preserves the essential information, while the expansion mainly serves as an intermediate computation. This inversion of the residual connection is what gives the block its name, the "inverted residual".



(a) Residual block **(b)** Inverted residual block

Figure 2.4: Comparison between residual and inverted residual blocks [7].

There are two main motivations for placing the shortcut at the bottleneck:

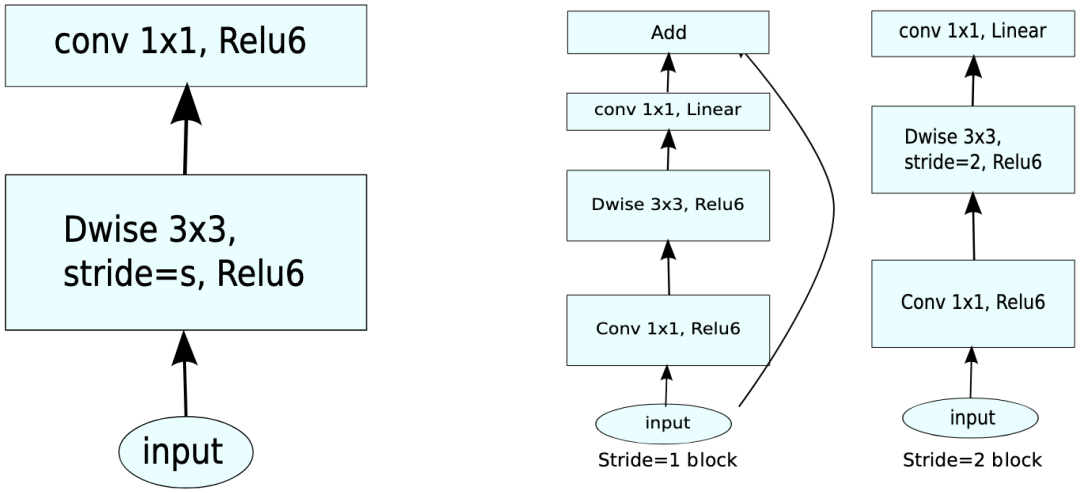
1. The usage of shortcut improves gradient propagation across multiple stacked blocks, same as with classical residual connections.

2. Inverted design is considerably more memory efficient. Moreover, experiments conducted by the authors prove that it is also as accurate as the wide-skip alternative.

Input	Operator	Output
$h \times w \times k$	1×1 conv2d, ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3×3 dwise $s = s$, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1×1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 2.1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t [7].

The basic implementation structure is defined and displayed in Table 2.1 and represented in Figure 2.5 compared to a MobileNetV1 basic block.



(a) MobileNetV1 **(b)** MobileNetV2

Figure 2.5: Comparison of convolutional blocks for MobileNetV1 and MobileNetv2 architectures [7].

The total number of Multiply-and-Accumulate (MACs) operations required, given a block of size $h \times w$, expansion factor t and kernel size k with d' input channels and d'' output channels, is:

$$\text{MACs} = h \cdot w \cdot d' \cdot t(d' + k^2 + d'')$$

2.1.4 Model Architecture

The architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers described in Table 2.2. All the nonlinear transformations are ReLU6, as it has been shown to be particularly robust with low-precision computation [9]. All depthwise kernels are 3×3. Each stage is specified by the expansion factor t, the output channels c, the number of block repeats n, and the stride s of the first block in that stage (subsequent repeats use stride 1).
The last 1×1 projection of each block is followed by BN without an activation (as explained in subsection 2.1.2).
The head is composed by a 1×1 convolution that expands the activation to 1280 channels at 7×7, followed by global average pooling and a final 1×1 convolution to k classes.
The primary network, with width multiplier $\alpha = 1$ and an input size of 224×224,

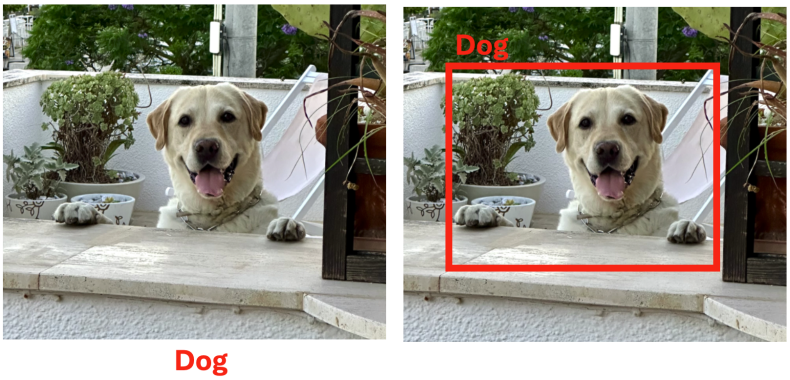
Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1×1	-	1280	1	1
$7^2 \times 1280$	avgpool 7×7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1×1	-	k	-	-

Table 2.2: MobileNetV2 architecture, width multiplier $\alpha = 1$, input size of 224×224. Each layer is defined by input resolution, operator, expansion factor t, output channels c, number of repeats n, and stride s.

has a computational cost of approximately 300 million MACs and comprises 3.4 million parameters [7].

2.2 Object Detection

Object detection is a computer vision task that consists in identifying and locating objects within an image or video. Specifically, object detection models generate bounding boxes around the entities detected within the input frame, thus allowing to locate them (either statically in a single image, or dynamically in the context of videos) in a scene, and label them at the same time.
It is important, before moving forward, to clarify the distinctions between object detection and image classification.
Image classification consists in assigning one (or more) label to an image. On the other hand, object detection provides more information about the image in question drawing bounding boxes around each recognized object in the frame. Figure 2.6 presents a practical example demonstrating the distinction between the two tasks. Object detection capabilities make it the go-to option for a various number of application, citing a few: crowd counting, visual perception in self-driving cars, video surveillance, face detection, anomaly detection.



(a) Image classification **(b)** Object detection

Figure 2.6: Comparison between image classification and object detection.

2.2.1 Evaluation Metrics

Assessing the performance of object detection models is crucial for providing a quantitative measure to compare across different methodologies. Different metrics provide insights into how well a model is able to detect, classify and localize different objects of interest in an image. We consider two families of evaluation metrics, i.e., those that can be used to assess the performance of the model on the task, and those which allow to quantify the inference cost, or the network's complexity.

Detection Metrics

- **Intersection over Union (IoU):** The overlap between the bounding box predicted by the model and the ground truth. From a mathematical perspective, it is defined as the ratio of the area of intersection and the area of union between the predicted bonding box ($B_{\mathbf{p}}$) and the ground truth ($B_{\mathbf{gt}}$) one:

$$\text{IoU} = \frac{|B_{\text{p}} \cap B_{\text{gt}}|}{|B_{\text{p}} \cup B_{\text{gt}}}$$

  ranges from 0 (no overlap between predicted and ground-truth boxes) to 1 (perfect alignment).

- **Precision:** It measures the proportion of true positive predictions among all positive predictions made by the model. It can be viewed as a measure of the accuracy of positive predictions. Mathematically, precision is defined as the ratio of true positives (TP) to the sum of true positives and false positives (FP), given by the following:

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

Precision ranges from 0 to 1, where a precision of 1 indicates that every positive prediction made by the model is correct. Precision is crucial when the cost of false positives is high.

- **Recall**: Also known as sensitivity, it measures the proportion of actual positives that are correctly identified by the model. It is a measure of the model's ability to capture all relevant instances. Recall is mathematically defined as the ratio of true positives (TP) to the sum of true positives and false negatives (FN), expressed as follows:

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

Recall ranges from 0 to 1, with a recall of 1 indicating that the model correctly identifies all positive instances. High recall is critical when the cost of false negatives is high, ensuring that the model captures as many true positives as possible.

- **Average Precision (AP):** Metric that represents the precision across different levels of recall, integrating the precision–recall curve. AP is computed as the area under the precision–recall curve, which is computed as the weighted mean of precisions achieved at each threshold, with the change in recall serving as the weight. Defining R_n and P_n as the recall and precision at the n-th threshold respectively, the formula for AP is given by the following:

$$\text{AP} = \sum_n (\text{R}_n - \text{R}_{n-1})\text{P}_n$$

AP is a key metric in object detection as it accounts for the trade-off between precision and recall across all thresholds.

- **Mean average precision (mAP):** Extends the concept of AP to multiple object classes, providing a unified metric that assesses the performance across all classes. Mathematically, mAP is the mean of the AP values calculated for each class, given by the following:

$$\text{mAP} = \frac{1}{n} \sum_{i=1}^{N} \text{AP}_i$$

where N is the number of classes in the dataset and AP_i is the average precision for the i-th class. mAP is the most widely used metric in object detection challenges as it provides a comprehensive measure of the model's ability to detect and localize objects across different categories.

Metrics for Resource-Constrained Devices

When deploying object detection models on mobile or embedded devices, traditional metrics such as mAP are not sufficient on their own. In these scenarios, additional factors become critical to ensure that the model can run under limited computational and memory budgets. The most relevant metrics are:

- **Frames per Second (FPS):** How many images the model can process in one second. A higher FPS indicates real-time performance, which is often required in applications such as autonomous driving or augmented reality.
- **Inference Latency:** Time required to process a single image (often reported in milliseconds). While related to FPS, latency is particularly important for interactive systems, where fast response times are essential.
- **Model Size:** Number of parameters or the storage footprint of the model (typically measured in MB). Smaller models are easier to deploy on devices with limited memory and storage, and can also reduce power consumption.
- **Peak Memory:** The maximum CPU/GPU memory consumed during inference, including model weights, intermediate activations, and buffers. Lower peak memory usage allows deployment on devices with stricter memory constraints, such as mobile or embedded platforms.

Together, these metrics provide a more complete picture of a model's suitability for real-world applications on constrained devices, complementing accuracy-based measures such as mAP.

2.2.2 Early Approaches

The field of object detection has experienced significant evolution, beginning with early techniques that laid the groundwork for future advancements.
The first ever approaches to object detection in computer vision relied heavily on handcrafted features. In the earliest stages, the primary challenge was how to effectively encode objects in a way that was both feasible from a computational point of view and robust enough to variations in scale, rotation, and illumination. One of the most influential early methods was the scale-invariant feature transform (SIFT) [21]. SIFT was designed to detect features in images that were independent to scale and rotation. SIFT became a powerful tool for matching objects across different images. Its robustness to changes in viewpoint and illumination made it a popular choice for early object recognition tasks. However, despite its accuracy, SIFT was computationally intensive, which limited its application in real-time systems.

Around the same time, another critical advancement was the introduction of the histogram of oriented gradients (HOG) [22]. HOG features were specifically designed for human detection and became widely used due to their simplicity and effectiveness. The HOG method consists in splitting an image into small, connected regions called cells, computing a histogram of gradient directions within each cell. This approach made it particularly effective for detecting pedestrians in images. The HOG combination with Support Vector Machines (SVMs) in a sliding window approach became the standard solution for pedestrian detection. Although HOG was relatively faster than SIFT, it still struggled with detecting objects in complex scenes or under varying lighting conditions.
Another important development in early object detection was the introduction of Haar-like features. Haar-like features are rectangular features used to represent the difference in intensity between adjacent rectangular groups of pixels. These features could be computed rapidly, allowing for real-time detection. Viola & Jones [23] combined these features with a cascaded classifier, where a series of increasingly complex classifiers were applied sequentially. The Viola–Jones detector became one of the first successful real-time object detection systems, widely adopted in applications ranging from security cameras to consumer electronics. Despite the success, the Viola–Jones still struggled with variations in pose, scale, and occlusion. It is essential to note that, in every proposed methods cited, the reliance on predefined features made them ofter lacked the flexibility to handle the complexity of real-world data. This was exactly the catalyst factor that would eventually drove the field toward ML and DL approaches that could learn features directly from the data.

2.2.3 Two-Stage Detectors

The introduction of CNNs marked a game-changing moment in the field of object detection, improving detection accuracy but also enabling models to generalize better across varying datasets and conditions. The breakthrough in using CNNs for object detection began with the development of the regions with convolutional neural networks (R-CNN) model [24]. R-CNN was one of the first methods that leveraged CNNs for the feature extraction phase. The main idea of the model was to generate region proposal for the image, namely candidate bounding boxes obtained via algorithms like Selective Search [24], and then pass them all through the feature extractor. The output is finally fed into a classifier to predict the presence and category of objects within each region.
However, R-CNN's performance gains came at the cost of a very low computational efficiency. Processing each region proposal independently led to significant redundancy, as many overlapping regions shared similar features. This problem motivated the development of Fast R-CNN [26], which introduced several key

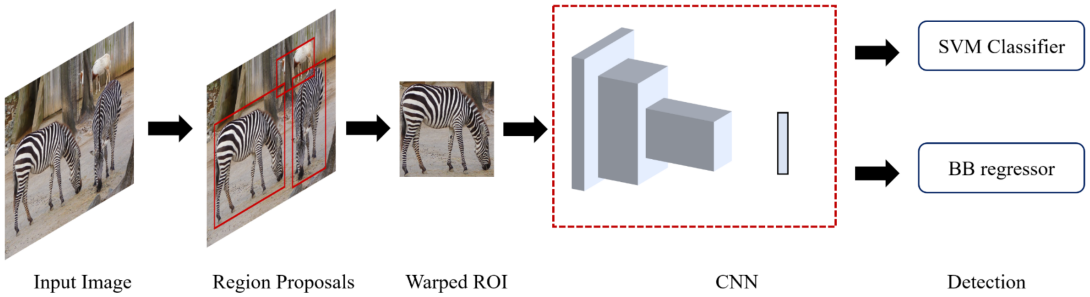


Figure 2.7: R-CNN architecture [25].

innovations. By employing a technique known as Region of Interest (RoI) pooling, Fast R-CNN allowed the sharing of convolutional computations across all region proposals. Instead of feeding each proposal through the entire CNN, the model needed to computed just a single feature map for the entire image and then extracted features for each proposal from this shared map. This optimization drastically reduced the computational burden and also enabled faster training and inference without no accuracy sacrificed.

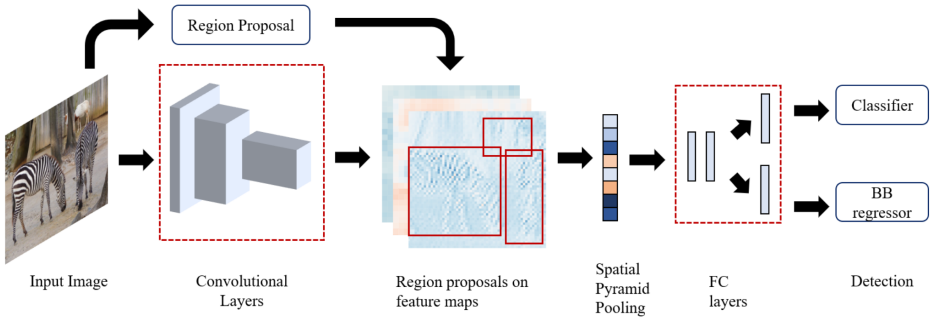


Figure 2.8: Fast R-CNN architecture [25].

Later on, the Faster R-CNN model [27] introduced another fundamental novelty: the region proposal network (RPN). The RPN replaced the external region proposal generation step with a fully trainable network. By generating region proposals directly from the CNN feature map, Faster R-CNN eliminated the need for external proposal algorithms like Selective Search, speeding up the detection process. The RPN consists of sliding a small network over the feature map and predicting object bounds and scores at each position. This approach also improved the quality of the proposals, leading to even better detection performance, quickly making Faster R-CNN the benchmark for high-accuracy object detection.
All the detectors explained so far are referred to in the literature as two-stage

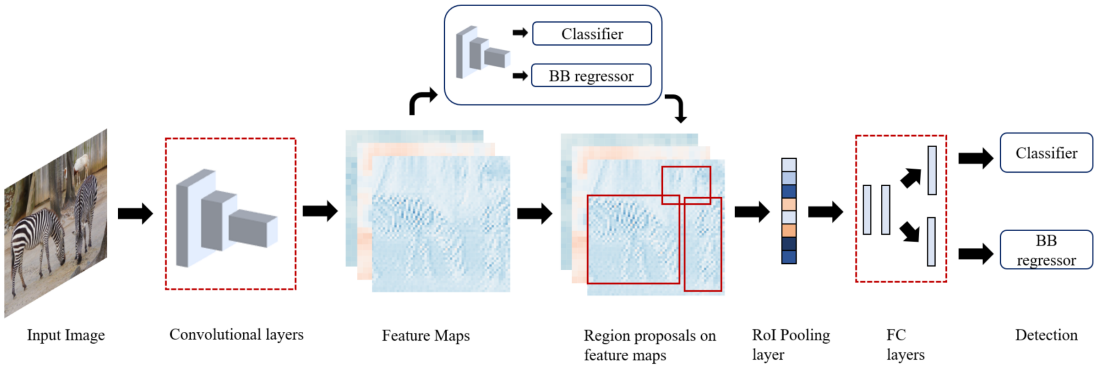


Figure 2.9: Faster R-CNN architecture [25].

detectors, since they first generate a set of region proposals and then classify and refine them in a second stage. Two-stage detectors, such as the R-CNN family, are typically computationally intensive models known for their high accuracy and reliability. However, their multi-stage pipelines make them less suitable for deployment in resource-constrained systems.

2.2.4 One-Stage Detectors

The pursuit of faster and more efficient solutions led to the development of one-stage detector models, of which the YOLO (You Only Look Once) family and SSDs (deeply covered in section 2.2.4) profoundly redefined the landscape of real-time object detection.

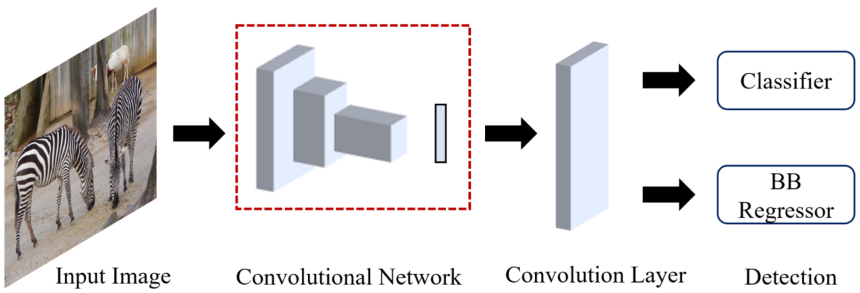


Figure 2.10: YOLO architecture [25].

The introduction of YOLO [28] marked a significant advancement in object detection. YOLO reframed object detection as a single regression problem, predicting bounding boxes and class probabilities straightly from the entire image in one unified process.

This end-to-end approach, shown in Figure 2.10, allows YOLO to process images in a single pass, significantly increasing the speed of detection.
The main innovation of this model consist of dividing the input image into a grid, with each grid cell predicting a fixed number of bounding boxes and confidence scores for the presence of objects. The directness of this approach allowed YOLO to operate at speeds never seen before, achieving real-time performance even on modest hardware [28]. Breaking free from having to focus on individual RoIs, YOLO manages to treat object detection as a global problem, enabling the capture of contextual information from the entire image.
However, this design choice also introduced several challenges. Indeed, early versions of YOLO struggled with small object detection and localizing objects close to each other, primarily due to the grid method used for predictions. Moreover, the fixed structure of the grid limited the model's ability to adapt to instances of objects that did not fit neatly within predefined cells. Despite these challenges, YOLO has seen significant improvements through subsequent iterations. Each iteration of YOLO has pushed the boundaries of what is considered possible in real-time object detection, making it a popular choice for applications where speed is critical, such as in autonomous driving, robotics, and real-time video analysis.
Both SSD and YOLO represent a shift toward more deployable object detection systems. Their ability to operate at high frame rates without significant sacrifices in accuracy has made them highly desirable systems in resource-constrained devices and latency-sensitive applications. This has opened up new possibilities for real-time applications, from drone navigation to augmented reality, where traditional two-stage object detection methods would be impractical.

Single Shot MultiBox Detector

The Single Shot Detector (SSD) [29], proposed in 2016 by Wei Liu et al., is one of the first attempts at using CNNs' pyramidal feature hierarchy for efficient detection of objects of various sizes [30].
As its base model, SSD uses the VGG-16 [31] model (pre-trained on ImageNet [32]) for extracting image features. After the VGG-16, SSD adds on top several convolutional layers of decreasing sizes, represented in Figure 2.11.
The main idea behind this model is that, since large fine-grained feature maps at earlier levels are mostly good at capturing small objects and small coarse-grained feature maps can detect large objects well, the detection happens in every pyramidal layer, targeting at objects of various sizes.

Anchor Boxes Unlike YOLO, SSD does not divide each image into a strict and arbitrary grid, but instead predicts, for each location of the feature map, the offset of predefined anchor boxes. All the anchor boxes tile the whole feature map

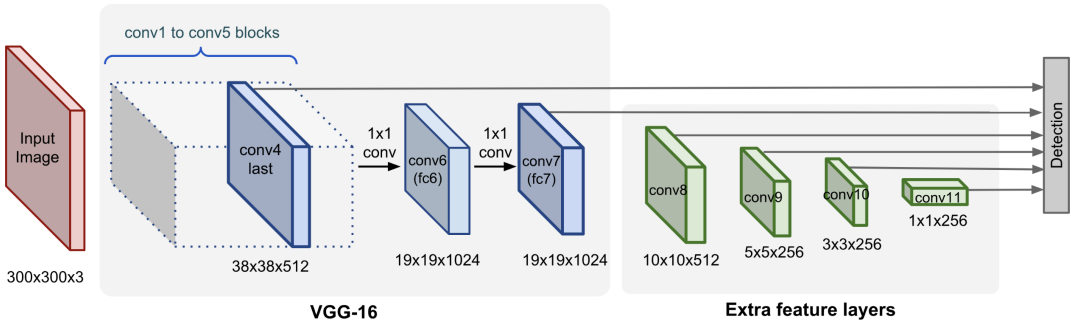
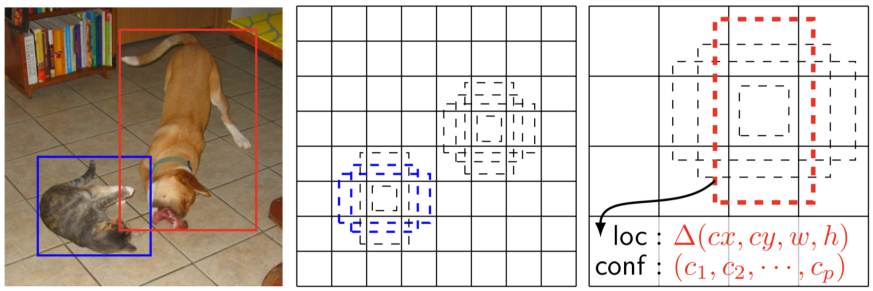


Figure 2.11: SSD architecture [30].

in a convolutional manner. In other words, every feature map cell is associated with a set of default bounding boxes of different dimensions and aspect ratios. These default bounding boxes are manually chosen in order to grant, in theory, the possibility for SSD to generalise for all types of input.
As explained, each feature map, depending on the level, has different sizes of receptive field. All the anchor boxes are rescaled so that each one of the feature map is responsible to detect object at a particular scale. Figure 2.12 shows an example of this concept, showing how the dog can only be well detected in the 4×4 feature map, which corresponds to a higher level, while the cat is captured by the 8×8 feature map, which represents a lower level.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 2.12: (a) The training data contains images and ground truth boxes for every object. (b) In a fine-grained feature map (8 x 8), the anchor boxes of different aspect ratios correspond to smaller area of the raw input. (c) In a coarse-grained feature map (4 x 4), the anchor boxes cover larger area of the raw input [29].

Each cell in the feature map is associated with a set of default bounding boxes that tiles the map in a convolutional manner. In this way, the position of each box relative to its corresponding cell remains fixed. At each cell, SSD predicts

the offsets relative to the default box ain the cell, together with a score for all the classes in the dataset that indicate the presence of the instance of a specific class in the boxes.
Formally, for each box out of the k locations, SSD computes c class scores and the 4 offsets relative to the default box shape. This results in a total of $(c+4)k$ filters applied on each location in the $m \times n$ feature map, yielding $(c+4)kmn$ outputs for each feature map. This approach is very similar to the one proposed in the Faster R-CNN work, with the only difference that, in this case, it is applied to feature maps at different levels of resolution.
Anchor boxes in the network are designed so that specific feature maps learn to respond to diffente scales of the objects. Given the usage of l feature maps in the network for prediction, the scale for each one is computed as:

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{l-1}(k-1) \qquad k \in [1, l]$$

where, in the original paper, $s_{\min}$ and $s_{\max}$ are respectively 0.2 and 0.9. This means that the lowest layer has a scale of 0.9 and all the layers in between are evenly spaced, as shown in Figure 2.13.

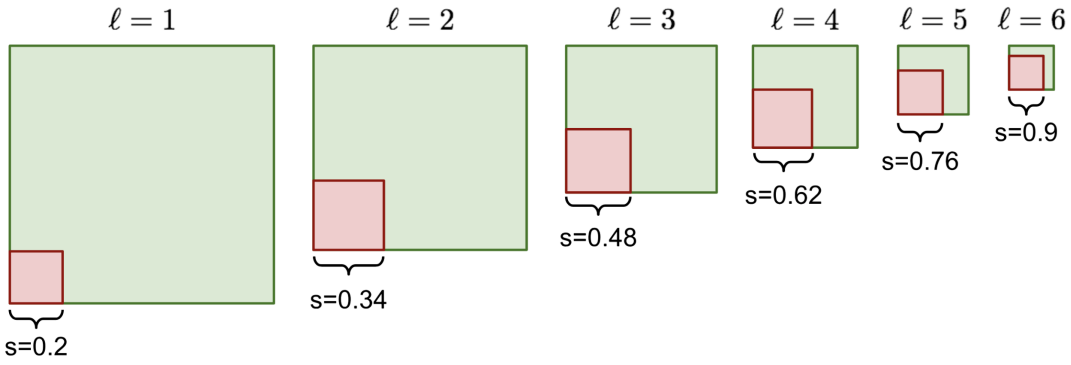


Figure 2.13: An example of how the anchor box size is scaled up with layer index l for $l = 6$, $s_{\min} = 0.2$, $s_{\min} = 0.9$. Only the boxes with aspect ration $r = 1$ are shown.

The aspect ratios of the boxes, noted as a_r, are imposed during the definition of the model. In the original paper, the authors used:

$$a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$$

The width (w_k^a) and the height (h_k^a) of the default boxes can be computed as such:

$$w_k^a = s_k\sqrt{a_r} \qquad h_k^a = s_k/\sqrt{a_r}$$

Combining predictions for all default boxes at different scales and aspect ratios from all the locations of many feature maps guarantees a diverse set of predictions, covering various input object sizes and shapes.

Bounding Box Regression Similar to the procedure used in the R-CNN detector, given a predicted bounding box coordinate $\mathbf{p} = (p_x, p_y, p_w, p_h)$ (center coordinate, width, height) and its corresponding ground truth box coordinates $\mathbf{g} = (g_x, g_y, g_w, g_h)$, the regressor is configured to learn scale-invariant transformation between two centers and log-scale transformation between widths and heights. All the transformation functions take $\mathbf{p}$ as input:

$$\begin{aligned}\hat{g}_x &= p_w d_x(\mathbf{p}) + p_x \\ \hat{g}_y &= p_h d_y(\mathbf{p}) + p_y \\ \hat{g}_w &= p_w \exp(d_w(\mathbf{p})) \\ \hat{g}_h &= p_h \exp(d_h(\mathbf{p}))\end{aligned}$$

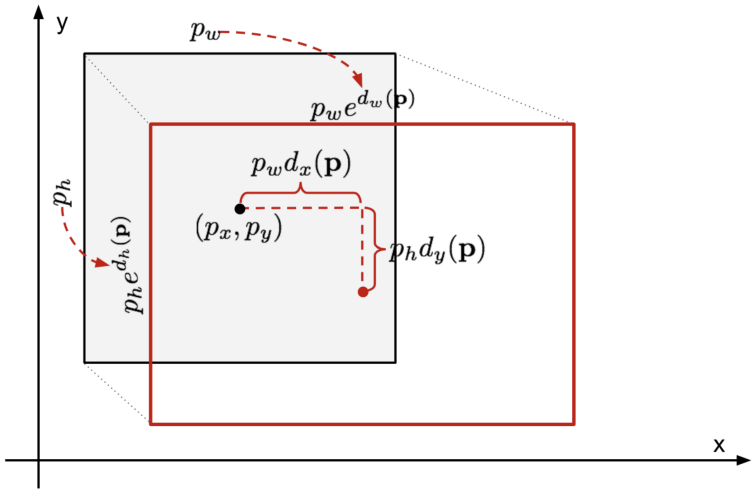


Figure 2.14: Illustration of transformation between predicted and ground truth bounding boxes [30].

This conversion, shown graphically in Figure 2.14, is applied so that all the bounding box corrections functions $d_i(\mathbf{p})$ $(i \in \{x, y, h, w\})$ can take any value between $[-\infty, +\infty]$ and a standard regression model can solve the problem by simply minimizing the Sum of Squared Error (SSE) loss with regularization. The

targets to learn are:

$$
\begin{aligned}
t_x &= (g_x - p_x)/p_w \\
t_y &= (g_y - p_y)/p_h \\
t_w &= \log(g_w/p_w) \\
t_h &= \log(g_h/p_h)
\end{aligned}
$$

Loss Function and Training/Inference Techniques The loss of the SSD model is the same as the one in YOLO, which is the sum of a localization loss and a classification loss, formally:

$$\mathcal{L} = \frac{1}{N}(\mathcal{L}_{\text{cls}} + \alpha\mathcal{L}_{\text{loc}})$$

where N is the number of matched bounding boxes and α balances the weights between two losses, selected from a candidates set in the original paper via cross validation.
The classification loss is a softmax loss over multiple classes:

$$\mathcal{L}_{\text{cls}} = -\sum_{i \in pos} \mathbf{1}_{ij}^{k} \log\left(\hat{c}_i^k\right) - \sum_{i \in neg} \log\left(\hat{c}_i^0\right), \quad \text{where } \hat{c}_i^k = \text{softmax}(c_i^k)$$

where $\mathbf{1}_{ij}^{k}$ is an indicator function that tracks whether the i-th bounding box and the j-th ground truth box match for an object of class k. *pos* and *neg* are, respectively, the set of matched bounding boxes and the set of negative examples.
The localization loss is a smooth L1 loss betwen the predicted bounding box correction and the ground truth values transformed through the coordinate correction explained previously:

$$\mathcal{L}_{\text{loc}} = \sum_{i,j} \sum_{m \in \{x,y,w,h\}} \mathbf{1}_{ij}^{\text{match}}\ L_1^{\text{smooth}}\left(d_m^i - t_m^j\right)^2$$

$$L_1^{\text{smooth}}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases}$$

$$t_x^j = \frac{g_x^j - p_x^i}{p_w^i}$$

$$t_y^j = \frac{g_y^j - p_y^i}{p_h^i}$$

$$t_w^j = \log\left(\frac{g_w^j}{p_w^i}\right)$$

$$t_h^j = \log\left(\frac{g_h^j}{p_h^i}\right)$$

where $\mathbf{1}_{ij}^{\text{match}}$ tracks whether the i-th bounding box with coordinates $(p_x^i, p_y^i, p_w^i, p_h^i)$ matches the j-th ground truth box with coordinates $(g_x^j, g_y^j, g_w^j, g_h^j)$ for any object. d_m^i, with $m \in \{x, y, h, w\}$ being the correction terms predicted by the model.
For the classification loss, SSD uses hard negative mining to select easy misclassified negative examples to construct the neg set. Without this step, the training would be dominated by the large number of easy negatives, which convey little useful gradient information.
The process of hard negative mining works as follows:

1. After computing the confidence scores for all anchors, every negative anchor, i.e., anchors that are not matched to any ground truth box, is assigned an "objectness" score.

2. These negative anchors are then sorted by their confidence loss. Hard negatives are the ones with the highest classification loss.

3. To keep the training balanced, only the top-ranked negatives are selected, ensuring that the ratio of negatives to positives does not exceed a certain ratio. In the original paper, the ratio of neg : pos was chosen to be $3 : 1$.

This strategy not only prevents the loss from being overwhelmed by trivial background examples but it also forces the model to learn from the most informative mistakes.

Figure 2.15: Multiple bounding boxes detect the car in the image. After non-maximum suppression, only the best remains and the rest are ignored as they have large overlaps with the selected one [33].

An important trick used during inference, introduced with the release of the first R-CNN, is Non-Maximum Suppression (NMS). This technique is fundamental in scenarios where the model predicts multiple bounding boxes for the same object. NMS reduces duplicate detections by sorting boxes by confidence, selecting the highest-scoring one, and discarding those that overlap too much (e.g., IoU > 0.45

was used in the original paper) with previously selected boxes. A simple example is shown in Figure 2.15.

Performance Analysis In [29], to evaluate the model, the authors compared SSD with the most performing solutions available at their time. For a fair comparison, all the model used VGG-16 as the base network, with a batch size of 1, were tested on the Pascal VOC2007 test set [34] and run on a NVIDIA GeForce GTX TITAN X. The results can be observed in Figure 2.16. It is important to note that YOLO, SSD300 (input resolution of 300×300) and SSD512 (input resolution of 512×512) are the only one-stage detectors present, while all the other are two-stage detectors based on region proposal approaches.

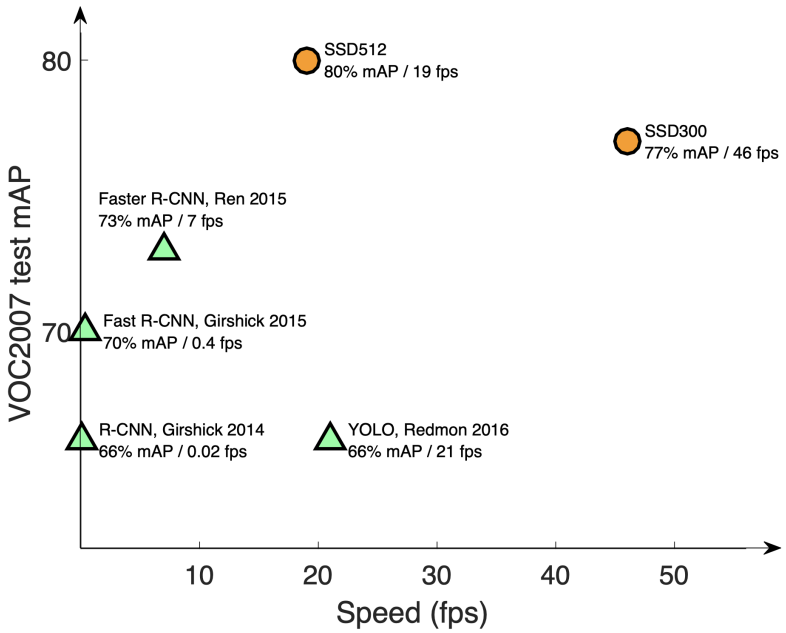


Figure 2.16: SSD performance comparison [29].

In terms of accuracy, SSD300 outperformed YOLOv1 while at the same time being significantly faster with a 25 fps margin and achieves comparable accuracy with a 6.6× faster speed when compared to Faster R-CNN (600×600 input images) while using lower input size of 300×300.

SSD512, on the other hand, performes 13% better than YOLOv1, keeping comparable inference speed, but sacrifing more than half compared to the SSD300.

SSDLite

SSDLite is a variant of the regular SSD which was first briefly introduced on the MobileNetV2 paper [7] and later reused on the MobileNetV3 paper [13]. The main innovation consists in replacing all the regular convolutions in the SSD prediction layer with separable convolution, namely depthwise convolutions followed by 1×1 projection.
When compared to regular SSDs, SSDLite significantly reduces both the parameter count and the computational cost, as it is shown in Table 2.3 taken from the original paper, where all models are trained and tested on the COCO dataset [35].

Network	**mAP**	**Params**	**MAdd**	**CPU**
SSD300 [34]	23.2	36.1M	35.2B	-
SSD512 [34]	26.8	36.1M	99.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
MNet V2 + SSDLite	22.1	**4.3M**	**0.8B**	200ms

Table 2.3: Comparison of SSD and SSDLite variants in terms of accuracy (mAP), parameter count, multiply-add operations (MAdd), and CPU inference latency [7].

SSDLite achieves a drastic reduction in both parameter count and MACs compared to the standard SSD variants. In particular, MobileNetV2 + SSDLite requires only 4.3M parameters and 0.8B operations, making it nearly an order of magnitude smaller and more efficient than SSD300, while running at 200 ms per image on CPU. Despite this efficiency, its accuracy (22.1 mAP) remains competitive with SSD300 (23.2 mAP) and even surpasses MobileNetV1 + SSDLite (22.2 mAP), confirming the advantage of the improved backbone. These results underline the suitability of SSDLite for resource-constrained environments such as mobile and embedded devices, where minimizing computation and memory footprint is often more critical than maximizing raw accuracy.

2.3 Knowledge Distillation

In recent years, DL neural networks have extremely expanded in terms of perfomance, but this also brought with it a significant increase in size and computational demands [36, 37]. Modern models often consist of billions of parameters and require significant computing resources. It was inevitable that, after achieving this incredible capabilities, their complexity and storage requirements would have become some of the most serious challenges for deployment in real-time scenarios and resource-constrained environments. In this context, exploring efficient machine

learning and DL techniques aimed at reducing model complexity and optimizing memory and energy usage is crucial. Well known optimization strategies can be include model compression methods such as quantization and pruning [38], the adoption of efficient neural network layers and computational techniques [39], and the Knowledge Distillation (KD) technique [40].
KD [41] involves a primary deep neural network (teacher) that distils knowledge to a smaller and optimised network (student). The student model can present a different design and architecture in order to substitute specific layers and operations from the original model. The main idea is that, with the teacher's supervision during the training, the student aims to replicate the teacher's model predictions despite the architectural differences.
The classical KD framework comprises the components: Knowledge, Distillation Algorithms, and Teacher-Student architectures. This review comprehends the main aspects of this technique, following the guide proposed by [40] and [42].

2.3.1 Knowledge

For "knowledge" it is intended which kind of information the teacher transmits to the student. It is possible to distinguish different forms of knowledge, the most popular comprehend: response-based, feature-based and relation-based.

Response-based Approach

In the response-based approach, the knowledge used derives from the logits, namely the output of the last layer of the teacher model. The student looks directly at the final prediction and it tries to replicate it. Formally, the objective is to minimize the distance between the logits of the teacher and the ones of the student. Given a vector of logits z as the output of the last Fully-Connected layer, a generalization of the distillation loss ($L_{ResD}(\cdot)$) can be formulated as:

$$L_{ResD}(z_t, z_s) = \mathcal{L}_R(z_t, z_s)$$

where $\mathcal{L}_R(\cdot)$ indicates the divergence loss of teacher z_t and student z_s logits. The Kullback-Leibler divergence loss is often utilised for the distillation loss, penalizing how much the student output differs from the teacher's one. Usually, for classification tasks the logits are intended as the soft targets, i.e. logits after the application of the softmax function, representing the probability that the input belongs to a specific class. Soft targets are computed as:

$$p(z_i, T) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

where z_i is the logit for the i-th class and T is an hyperparameter that controls the importance of each of them.
Distillation loss is most of the times combined with the student loss, i.e. the loss between the ground truth label and the soft logits of the student model, which is often a cross-entropy loss with ground-truth labels. A representation of the response-based approach can be viewed in Figure 2.17.

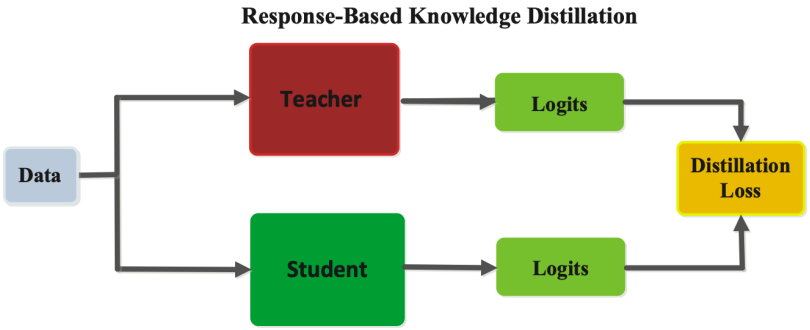


Figure 2.17: Scheme of a generic response-based knowledge distillation approach [40].

Feature-based Approach

As for the feature-based approach, visually represented in Figure 2.18 knowledge is represented by the output of intermediate layers. In this case, the distillation loss is:

$$L_{\text{FeaD}}(f_t(x), f_s(x)) = \mathcal{L}_F(f_t(x), f_s(x))$$

where $f_t(x)$ and $f_s(x)$ are the feature maps of the intermediate layers of the teacher and student respectively, and $\mathcal{L}_F(\cdot)$ indicates the similarity function. As $\mathcal{L}_F(\cdot)$, measures like l_2-norm, l_1-norm and cross entropy are the ones usually used.

Relation-based Approach

This approach focuses on transferring information by modeling the relationships either between layers or between data samples. Instead of considering only single feature maps, the idea is to exploit the correlations that the teacher captures. For instance, given two feature maps from the teacher, $\hat{f}_t$ and $\tilde{f}_t$, and the corresponding ones from the student, $\hat{f}_s$ and $\tilde{f}_s$, the relation loss can be defined as:

$$L_{\text{RelD}}(f_t, f_s) = \mathcal{L}_{R1}\Big(\Psi_t(\hat{f}_t, \tilde{f}_t), \Psi_s(\hat{f}_s, \tilde{f}_s)\Big)$$

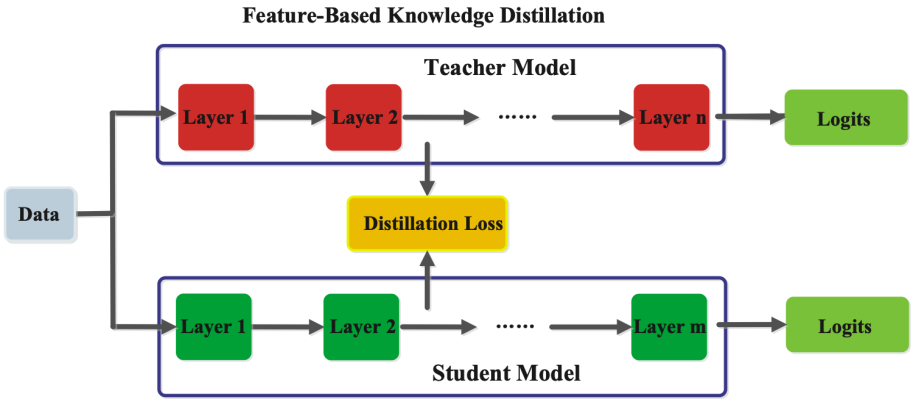


Figure 2.18: Scheme of a generic feature-based knowledge distillation approach [40].

where $\Psi_t(\cdot)$ and $\Psi_s(\cdot)$ are similarity functions, and $\mathcal{L}_{R1}$ denotes the correlation function. This setting allows the student to mimic the teacher's representation of similarities between features.
A similar idea can be extended to pairs of data samples. Let (t_i, t_j) and (s_i, s_j) denote data points in the teacher and student feature spaces, respectively. The relation loss in this case becomes

$$L_{\text{RelD}}(F_t, F_s) = \mathcal{L}_{R2}\Big(\psi_t(t_i, t_j), \psi_s(s_i, s_j)\Big)$$

where $\psi_t(\cdot)$ and $\psi_s(\cdot)$ are similarity functions applied to sample pairs, while $\mathcal{L}_{R2}$ is the function measuring the consistency between the teacher and student correlations.

2.3.2 Distillation

The distillation process defines how the teacher and student models are trained together. Different strategies have been explored in the literature, which can be grouped into three main categories [40].

- **Offline distillation:** In this setting, the teacher is a pre-trained model whose parameters remain fixed. Knowledge is transferred to the student by matching its predictions or feature representations against those of the teacher. Since the teacher is not updated, this strategy is the most widely used and well established.
- **Online distillation:** Unlike the offline case, here teacher and student are optimized simultaneously in an end-to-end fashion. Both models interact

during training, which makes the scheme more flexible but also more demanding in terms of training complexity.

- **Self-distillation:** A particular case in which the same network acts as both teacher and student. Typically, deeper layers supervise shallower ones, transferring internal representations within the same model. Although less common, this approach has shown that even a single network can benefit from distillation.

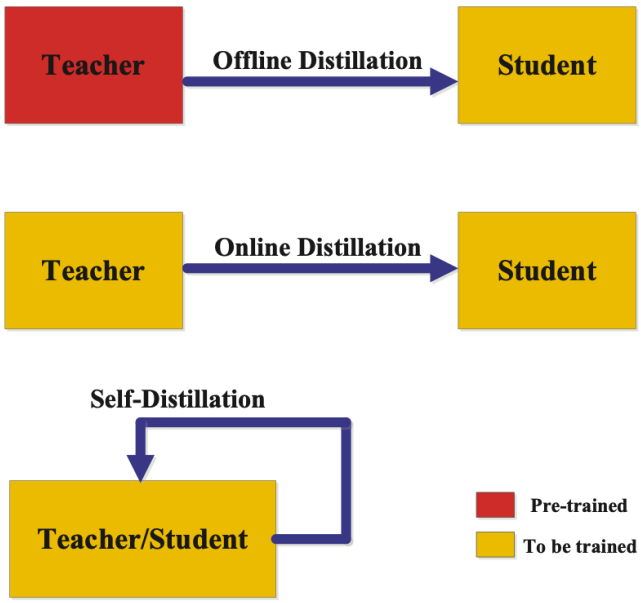


Figure 2.19: Representation of different types of distillations [40].

2.3.3 Teacher-Student architecture

In knowledge distillation, the choice of teacher and student architectures plays a central role in determining the quality of knowledge transfer. The gap in capacity between the two networks affects how well the student can reproduce the teacher's behavior. Typically, the goal is to design a student that is less complex than the teacher, while still able to capture its essential representations.
Different strategies can be adopted when defining the student model:

1. A reduced version of the teacher, obtained by decreasing the number of layers or channels.

2. A quantized variant that maintains the overall structure but lowers memory and computational requirements.

3. A completely different design, possibly based on more efficient operations or on a globally optimized architecture.

Over the years, many methods have been introduced to manage this trade-off between accuracy and efficiency, aiming to build student networks that balance reduced complexity with satisfactory performance [43, 44].

2.4 Slimmable Neural Networks

Slimmable Neural Networks (SNNs) represent a paradigm shift in the design of efficient DL models. Unlike conventional neural networks, which are trained and deployed at a fixed size, SNNs are explicitly constructed to run at multiple capacities within a single set of shared parameters [1]. This design makes them particularly suitable for applications in which computational budgets, memory constraints, or latency requirements may vary dynamically at runtime.
In real-world scenarios, the diversity of hardware devices introduces strong variability in available resources. For example, a high-end smartphone can afford to run a large model with higher accuracy, while a low-power embedded system may only sustain smaller architectures within strict latency budgets. Even within the same class of devices, the performance of a given model can vary considerably from one specific device to another, as illustrated in Table 2.4.

Device	**Runtime**
OnePlus 6	24 ms
Google Pixel	116 ms
LG Nexus 5	332 ms
Samsung Galaxy S3	553 ms
ASUS ZenFone 2	1507 ms

Table 2.4: Runtime of MobileNet v1 for image classification on different devices, as reported in [45].

Traditionally, this challenge has been addressed by training, benchmarking, deploying, and maintaining multiple separate models, each corresponding to a different operating point. This approach is highly inefficient, not only because it require a large offline table to record the allocation of models to different devices according

to their time and energy budgets, but also because it increases training and maintenance costs. Furthermore, even on the same device, the computational budget can fluctuate (e.g., due to background processes reducing available resources), and the energy budget may vary (e.g., when operating in low-power or power-saving modes). Switching between larger and smaller models also introduces overhead, as the time and data required for downloading or offloading models is not negligible. Dynamic neural networks have been introduced as a way to enable selective inference paths by Liu and Deng [6]. Liu and Deng proposed controller modules that determine whether subsequent components should be executed, which leads to low theoretical computational complexity but makes optimization and deployment on mobile devices not trivial, since dynamic execution hinders layer fusion and memory reuse. Huang et al. [46] incorporated early-exit branches connected through dense connectivity, allowing predictions at intermediate layers. Wu et al. [47] and Wang et al. [48] explored selective block execution in deep residual networks, where only a subset of residual blocks is activated during inference. Despite these advances, reducing depth does not lower the memory footprint at inference time, unlike width scaling, which directly reduces the number of channels and is therefore more suitable for mobile environments where memory is a key constraint.
SNNs address these limitations by embedding multiple operating points into a single model. At inference time, the model can dynamically switch configurations, selecting the most suitable one based on device constraints or application requirements. An example can be viewed in Figure 2.20.

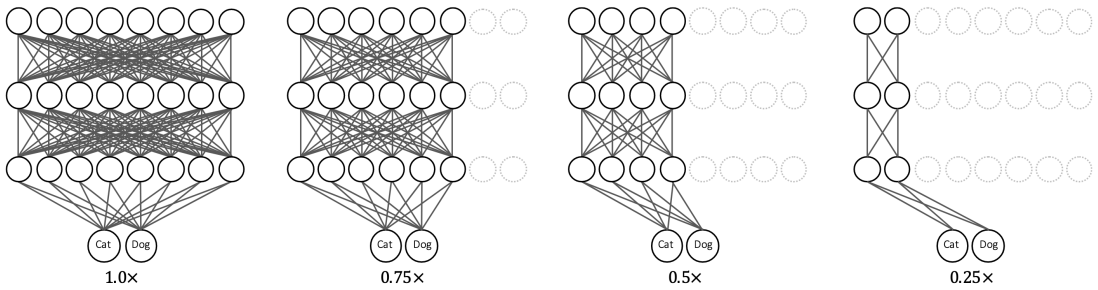


Figure 2.20: Illustration of a SNN. A single model can run at different widths (e.g., 0.25×, 0.5×, 0.75×, 1.0×), enabling adaptive accuracy-efficiency trade-offs [1].

Let $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ denote a set of width multipliers, where each α represents the fraction of channels active in each layer. A slimmable network M can then be expressed as a family of models:

$$M_\alpha : \mathcal{X} \to \mathcal{Y}, \quad \alpha \in \mathcal{A},$$

where all M_α share the same weights, but differ in the subset of channels used in each layer.
For instance, consider a convolutional layer with $C = 64$ channels. With a width multiplier $\alpha = 0.25$, only 16 channels are active, while $\alpha = 1.0$ activates all the 64 channels. Thus, depending on α, the computational complexity of the same network can vary substantially, while still relying on the same parameter set.
The central property of SNNs is the ability to balance accuracy and efficiency on the fly. A smaller width (e.g., 0.25×) typically yields faster inference and reduced memory usage, at the expense of lower accuracy. A larger width (e.g., 1.0×) maximizes accuracy but requires more computational resources.
This trade-off enables a single model to serve multiple deployment scenarios. For example, on a flagship device, the network may run at 1.0× width for maximum accuracy, and on a low-power embedded device, the same network may switch to 0.25× width to reduce latency and memory consumption.
Much more interesting is the case of an adaptive systems, in which the width can change dynamically depending on battery level, user requirements, or real-time constraints. This adaptive behavior is particularly valuable when a single end device must handle diverse operating conditions without switching between entirely different models. For instance, a mobile device may run the network at full width while charging, ensuring maximum accuracy for applications such as augmented reality or high-resolution image processing. When the device enters a low-battery mode, the same model can seamlessly switch to a narrower configuration to prolong autonomy, sacrificing some accuracy for higher efficiency. Similarly, in scenarios with variable input streams, such as real-time video analysis, the network may adjust its width dynamically: larger configurations can be used for complex frames with dense content, while smaller widths suffice for simpler or empty frames, thereby reducing unnecessary computation.

2.4.1 History

As already stated in Chapter 1, the first appearance of SNNs was in 2018, by Yu et al. [1]. In this work, the authors proposed a first approach on how to train one network able to behave well at different widths. Yu et al. observed that naive training of a shared network for multiple widths led to divergent batch normalization statistics and severely degraded accuracy. Their solution was Switchable Batch Normalization (S-BN), which consists of providing separate BatchNorm layers for each width setting, preventing the inter-width interference that arises from using one BN for all sub-networks.
With Switchable BN, the shared models achieved accuracy on par with or better than individually trained models of the same width. Without any task-specific re-tuning, the SNN managed to outperform individually trained backbones at

equivalent widths on different tasks (object detection and instance segmentation), showing how SNNs can be practically useful for deploying adaptable vision systems. Building on this success, Yu et al. introduced, in 2019, Universally Slimmable Networks (US-Nets) [49], addressing two important limitations of the original SNNs. First, SNNs were limited to a few predefined width options. US-Nets instead allow any width between a minimum and maximum, handling cases where every possible channel count (within a range) should perform well, rather than just a fixed set, utilizing techniques like "Sandwich Rule" and "Inplace Distillation". Second, they sought to generalize the approach to architectures without BatchNorm.
One immediate challenge after the introduction of SNNs was automatically finding the optimal widths or architecture for a given efficiency target. Traditional network pruning and NAS methods often require training many candidate models or a large search cost. Slimmable networks, by contrast, provide a trained accuracy predictor for any width configuration. This led to a series of works that combine slimmable networks with NAS or model compression.

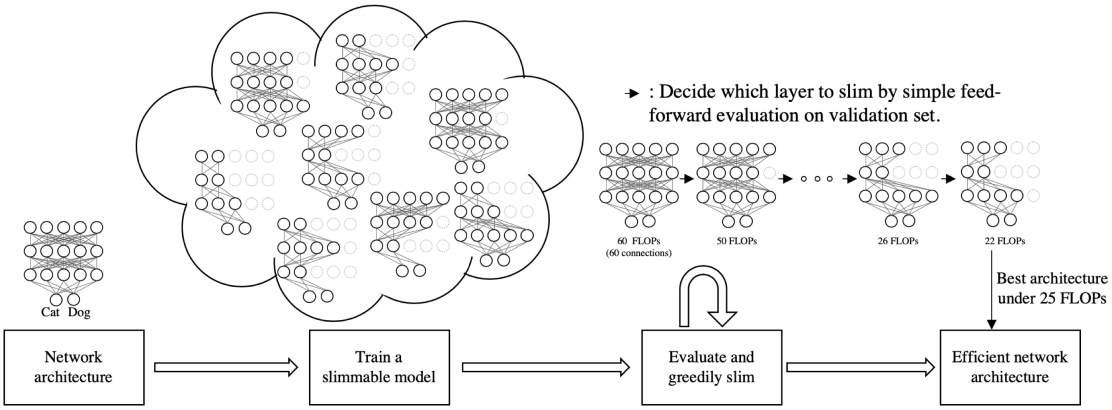


Figure 2.21: Flow diagram of AutoSlim proposed approach [4].

AutoSlim (2019) [50] introduced a method to determine per-layer channel counts under a FLOPs or latency budget, without exhaustive search. Using a slimmable network trained to support multiple widths, the authors treated each layer's width as an independent decision rather than applying a global multiplier. The algorithm works greedily: starting from the full model, it iteratively prunes the layer whose channel reduction causes the smallest accuracy drop, until the target budget is met. Since the slimmable network can be evaluated across many configurations without retraining, this one-shot process is far more efficient than conventional NAS.
AutoSlim optimized channel widths of a given network and paved the way for later approaches that extended weight sharing to a broader range of architectural choices. Once-for-All (OFA) by Cai et al. [2] is a landmark work that pushed slimmable networks into the realm of full-fledged NAS.

Another notable development was BigNAS [3], which argued that a well-trained super-net can eliminate the need for post-search fine-tuning. Unlike earlier weight-sharing methods (e.g., OFA), where sub-networks often required retraining, BigNAS showed that with careful one-stage training, sub-networks could be directly deployed using the shared weights. This demonstrated that a single slimmable model can serve as the final architecture across multiple deployment sizes, further simplifying the NAS pipeline.
To conclude, by 2020 SNNs had evolved into a powerful paradigm for one-shot NAS and model compression. Works like OFA and techniques large super-net training (BigNAS) enabled covering massive architecture spaces with one network.

2.4.2 Benefits and Current Challenges

SNNs can bring many practical benefits that are crucial for low-powered and resource costrained devices:

- **Parameter sharing:** All widths share the same superset of parameters, avoiding the need to train and store multiple models.
- **Flexible deployment:** A single model can be deployed across a wide range of devices with different resource-constraints.
- **Scalable performance:** Inference can be scaled in real time by adjusting the number of active channels, offering a near-continuous trade-off between accuracy and efficiency.
- **General applicability:** The concept of width-scaling applies broadly to convolutional, residual, and other DL architectures.

Despite major progress, several challenges remain for SNNs, citing a few:

- **Training complexity and interference:** Training one network to support many sub-models is difficult, since sub-networks compete for capacity and gradients. Heuristics like the sandwich rule, but scaling to huge search spaces (width, depth, kernel, resolution) is still challenging.
- **Normalization and calibration:** While switchable BN solved discrete widths, continuous configurations or alternative normalizations introduce new stability issues, requiring smarter calibration or adaptive normalization methods.
- **Granularity of slimming:** Most approaches adjust entire layers uniformly, but finer per-layer control could improve trade-offs. Training such networks efficiently, however, remains difficult.

- **Dynamic execution:** Input-dependent slimming promises efficiency, but training gates jointly with the super-net is not trivial, and hardware systems must adapt to support per-input variability.
- **Robustness:** Different widths may behave differently under distribution shifts or adversarial attacks, raising concerns about safety and reliability.

Chapter 3

Related Works

3.1 Slimmable DNNs in Computer Vision

While the early slimmable networks works focused on image classification as a primary benchmark, the applicability of SNNs in other computer vision tasks soon became a subject of research. The already noted work by Yu et al. [1] validated slimmable backbones on object detection and instance segmentation. The slimmable ResNet-50, for example, could power a Mask R-CNN detection pipeline and could obtain a lighter or heavier detector as needed by switching its width. The result was better COCO detection metrics for a given model size compared to training separate detectors for each size.

For semantic segmentation, a well-known computationally intensive task, researchers developed task-specific slimmable strategies. SlimSeg, by Xue et al. [51], is a prime example. One challenge in segmentation is that object boundaries can suffer when the network is heavily compressed, the SlimSeg method introduced a boundary-guided loss to specifically address this issue. Specifically, during training, an auxiliary loss focuses on correctly predicting the segmentation edges for each sub-network, since it was observed that differences between the full and slim models' outputs concentrate near borders. Moreover, SlimSeg used a form of "parametrized channel slimming with stepwise distillation". In practice, they train the full segmentation model, then progressively include narrower versions while using the full model's output as supervision for the sub-networks. With these techniques, SlimSeg achieved a single model that can operate at multiple pixel resolutions and channel widths, adapting to different speed/accuracy needs.

Another interesting vision application is in the domain of generative models, in particular on Generative Adversarial Networks (GANs), a type of generative models based on game theory where ANNs are used to mimic a data distribution [52]. In 2020, Hou at al. conducted an exploratory work on Slimmable GANs [53], where

the generator was made slimmable so that it could produce images at varying computation costs. Although this may seem a less-traditional use-case, it indicated that even in generator–discriminator training, one can train a GAN that functions across multiple model sizes. The slimmable generator was able to switch its width to trade off generation fidelity and speed. While not a core focus of our review, it's worth noting as it expands the horizons of where SNNs can be applied.

3.2 Dynamic Slimmable DNNs

A crucial extension of SNNs in vision came with the notion of dynamic slimmable networks (DS-Net) [54], by Li et al., a work that deeply inspired the model we present. Traditional SNNs assume the width is chosen statically based on a deployment scenario, e.g. a device or a fixed runtime budget, however one can imagine adjusting the width per input, e.g. an easy image might only require a slim sub-network to get right, whereas a harder image might need the full network.

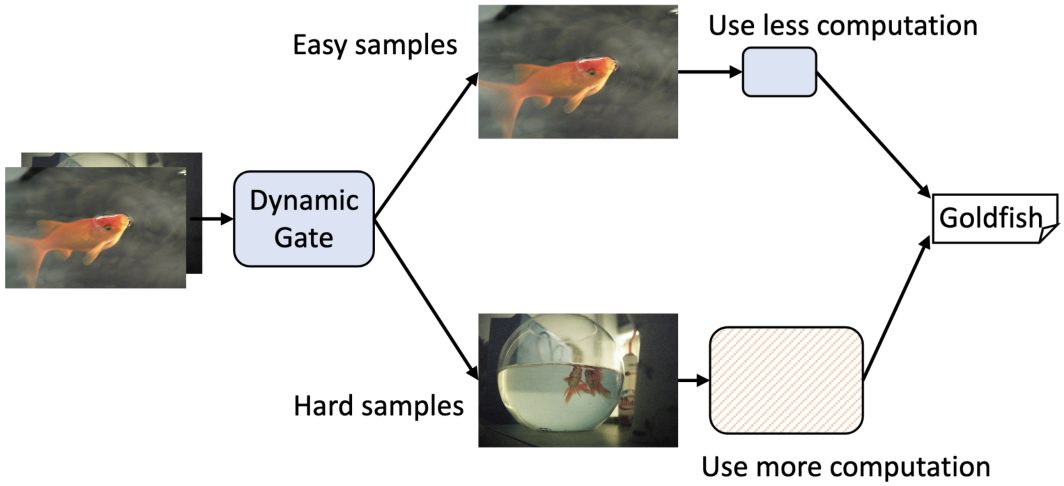


Figure 3.1: Illustration of dynamic networks on efficient inference. Input images are routed to use different architectures regarding their classification difficulty [54].

DS-Net manages to achieve hardwer-efficiency dynamically slicing the number of filters of the DNN with respect to different inputs. This approach not only guarantees consistency and validity across all width configurations, but also preserves hardware efficiency by keeping filters static and contiguous when adjusting the width of the network.
To ensure practical acceleration on hardware, it is important that filters remain contiguous and relatively static during weight selection. Consider a convolutional

layer with N output filters and M input channels, where the weights can be represented as $\mathbf{W} \in \mathbb{R}^{N \times M}$ (omitting the spatial dimension). To make the architecture slice-able at different widths, the authors introduce a routing agent $\mathcal{A}(\theta)$. This agent outputs a *slimming ratio* $\rho \in (0,1]$, which determines the number of active filters, i.e., $\rho \times N$. The resulting slice-able convolution can then be expressed as:

$$\mathcal{Y} = \mathbf{W}[\,:\lfloor \rho \times N \rfloor\,] \; * \mathcal{X}$$

For training, Li et al. propoed an approach similar to the one proposed by Yu et al. [49], comprehending the Sandwich Rule and the In-Place Distillation technique. In the latter, the widest sub-network is used as the target network generating soft labels for other sub-networks. On top of this idea, with the goal of stabilizing the acute fluctuation of the weights during training and mitigating the hardship of convergence, the authors proposed a training scheme named In-place Ensemble Bootstrapping. This consist of using the exponential moving average (EMA) of the model as the target network that generates soft labels. Formally, given θ and θ' the parameters of the online network and the target network, respectively, we have:

$$\theta'_t = p\theta'_{t-1} + (1-p)\theta_t$$

where p is a momentum factor controlling the ratio of the historical parameter and t is a training timestamp which is usually measured by a training iteration.
In addition to that, the training uses different sub-networks as a teacher ensemble when performing distillation, with the purpose of generating more accurate and more general soft labels for distillation training of the student network.
To predictively adjust the network width, in [54], the authors propose a double-headed dynamic gate, composed by an attention head (really similar to other channel attention methods [55, 56]) and a slimming head, integrated into the model with nearly zero cost. The double-headed gate takes as input the feature map and output the slimming ratio ρ using a one-hot design.
The training of the gate is separated and subsequent to the one of the whole network. In order to avoid the collapse of the gate into a static one, the work proposed, together with the addidion of Gumbel noise [57], a technique named Sandwich Gate Sparsification (SGS). SGS is applied by using the slimmest sub-network and the whole network to identify easy and hard samples online and further generate the ground truth slimming factors for the slimming heads of all the dynamic gates. Experiments on ImageNet showed DS-Net could cut computation by $2-4\times$ and achieve $1.6\times$ actual speedup on devices, with minimal accuracy loss, outperforming static models and prior dynamic networks (up to 5.9% higher accuracy than static counterparts under the same cost).
Authors also experimented the DS-Net as a feature extractor in object detection with Feature Fusion Single Shot Multibox Detector (FSSD) [58] and compare it with a MobileNetV1 on the VOC 2007 test set [34]. DS-Net managed to achieve 1.8 mAP improvement with a $1.34\times$ computation reduction.

Chapter 4

Methods

This chapter presents in detail the methodologies employed to develop a slimmable object detection network equipped with a channel gating mechanism. The goal is twofold: first, to design a model capable of achieving competitive results at four different widths (0.25×, 0.5×, 0.75×, and 1.0×) which can be switched on-the-fly during inference; and second, to exploit the channel gating mechanism to identify empty or potentially simple images and trigger an early-exit strategy, thereby reducing computational cost without severely compromising detection performance.
Section 4.1 focuses on the preprocessing steps applied to the Cityscapes dataset [59]. Since Cityscapes was originally designed for semantic segmentation, it was first converted into an object detection format and subsequently divided into a grid of non-overlapping patches. Special care was taken when handling bounding boxes that crossed patch boundaries. Additional data augmentation procedures were applied to increase variability and robustness.
Section 4.2 introduces the model employed for the object detection task. It presents the chosen backbone and the SSDLite detection head configuration. Particular emphasis is placed on the integration of the gating module (Dynamic Slimming Gate), describing both its internal structure and the mechanism by which it selects different network widths.
Finally, section 4.3 outlines the training methodology, which is organized into two complementary phases. *Phase 1* focuses on learning robust detection capabilities across the four predefined widths by applying an adapted version of the In-Place Ensemble Bootstrapping technique proposed by [54]. This stage ensures that all subnetworks, from the smallest to the largest, are jointly optimized. *Phase 2* then targets the training of the gating module, enabling it to discriminate between empty or simple inputs and more complex ones. To achieve this, we leveraged the Sandwich Gate Sparsification strategy, also introduced by [54], which encourages the gate to route trivial samples to the slimmest configuration while reserving wider subnetworks for challenging inputs.

4.1 Dataset Preprocessing

As mentioned in the introduction, Cityscapes was originally designed for semantic and instance segmentation tasks. It is a widely used benchmark in academia, particularly in applications such as pedestrian detection, traffic light recognition, and vehicle tracking, as it provides pixel-precise class annotations from a vehicle's perspective in complex urban environments. To adapt Cityscapes for object detection, we converted its instance segmentation annotations into bounding boxes using a conversion tool [60]. Since the annotations in Cityscapes also consider segmentation instances, each object is defined by a segmentation mask and a unique instance ID. This information can be directly exploited to determine the spatial extent of each object and derive its corresponding 2D bounding box, as shown in Figure 4.1. Overall, after the conversion the dataset covers eight object categories: *Person*, *Car*, *Truck*, *Rider*, *Motorcycle*, *Bicycle*, *Bus* and *Train*. Across the entire data, the distribution of object categories is highly imbalanced. *Car* and *Person* dominate the annotations, together accounting for more than 80% of all bounding boxes. Mid-frequency classes such as *Bicycle* and *Rider* are present in significantly smaller amounts, while *Motorcycle*, *Truck*, *Bus*, and *Train* form a long tail of rare categories, each contributing less than 2% individually. Detailed counts per class and split are reported in Table 4.1.
Although this conversion required additional preprocessing, Cityscapes remains a practical choice thanks to its detailed annotations, urban diversity, and challenging conditions, which make it suitable for evaluating object detection models aimed at real-world deployment on nano-drones.

4.1.1 Tiling Process

Once the conversion was completed, each Cityscapes image (2048×1024 pixels) was divided into a grid of eight non-overlapping patches, an example is illustrated in Figure 4.2. This choice serves a dual purpose. First, it adapts the high-resolution images of the dataset to the 512×512 input size required by the SSDLite detector. Unlike anisotropic resizing, tiling avoids geometric distortions and preserves the original aspect ratios of objects, which has been shown to improve detection accuracy [61]. Second, the partitioning naturally produces a significant number of empty patches, i.e., image crops without bounding boxes, which are later exploited during the training of the adaptive mode.
After the grid partitioning, the final preprocessing step consisted of removing bounding boxes that only slightly extended into adjacent tiles. This was done to avoid generating incomplete or ambiguous annotations, which could negatively impact the detector's training. Empirically, all bounding boxes lying on the border of a tile that had at least one side shorter than 20 pixels were discarded. An

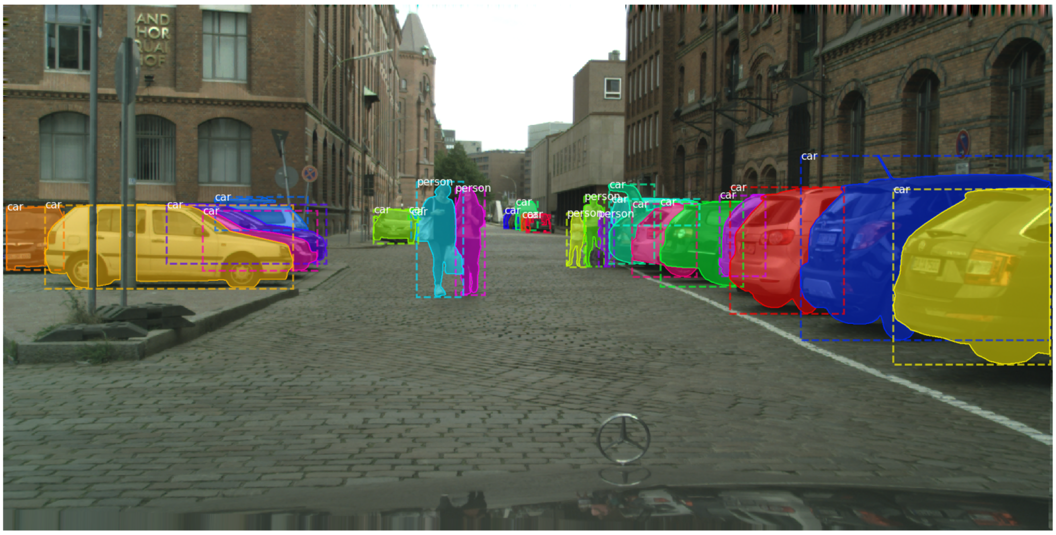
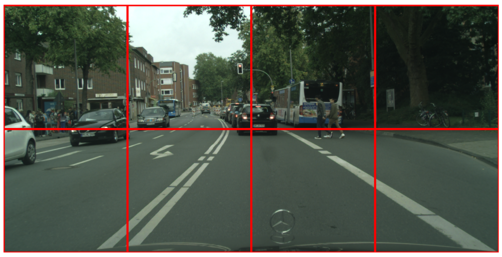


Figure 4.1: Example from the Cityscapes dataset showing instance segmentation masks (colored regions) and the corresponding 2D bounding boxes (dashed rectangles) after conversion to an object detection format.

(a) Example of a raw image from the Cityscapes dataset.



(b) Division into eight non-overlapping 512×512 patches used as inputs.

Figure 4.2: Example of the tiling process applied to Cityscapes images.

example of removed bounding boxes is shown in Figure 4.3.

The final dataset consists of 27,800 non-overlapping 512×512 tiles, split into 19,040 (68.49%) for training, 4,760 (17.12%) for validation, and 4,000 (14.39%) for testing. Among these, 6,883 training samples (∼36%), 1,557 validation samples (∼32%), and 967 test samples (∼24%) contain no objects, providing a substantial pool of empty images for adaptive training.

Across all dataset splits, the distribution of object categories is highly imbalanced. *Car* and *Person* dominate the annotations, together accounting for more than 80% of all bounding boxes. Mid-frequency classes such as *Bicycle* and *Rider* are present in significantly smaller amounts, while *Motorcycle*, *Truck*, *Bus*, and *Train* form a

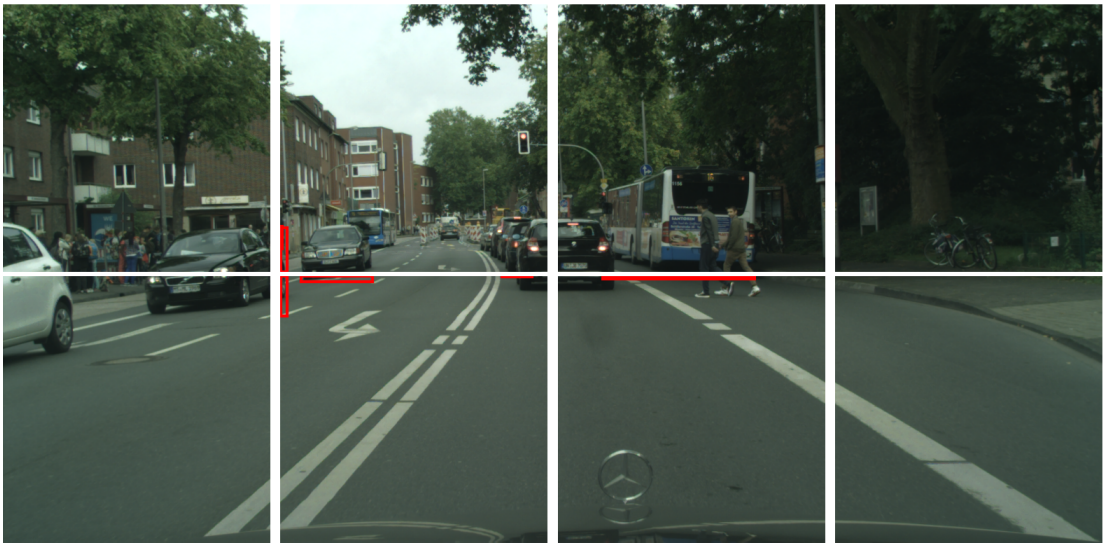


Figure 4.3: Example of bounding boxes removed during preprocessing. Boxes lying on tile borders with at least one side shorter than 20 pixels were discarded to avoid incomplete or misleading annotations in the final dataset.

long tail of rare categories, each contributing less than 2% individually. Detailed counts per class and split are reported in Table 4.1.

Category	**Train**	**Val**	**Test**	**Total**	**% of Total**
Person	21,263	5,202	4,322	30,787	32.3%
Car	37,378	5,954	6,787	50,119	52.6%
Truck	674	38	144	856	0.9%
Rider	2,148	450	719	3,317	3.5%
Motorcycle	1,059	227	199	1,485	1.6%
Bicycle	4,675	1,009	1,763	7,447	7.8%
Bus	531	68	164	763	0.8%
Train	310	100	45	455	0.5%
Total	68,038	13,048	14,143	95,229	100%

Table 4.1: Class distribution of the processed Cityscapes dataset after conversion and tiling. The dataset is strongly imbalanced, with *Car* and *Person* representing the majority of instances, while classes such as *Truck*, *Bus*, and *Train* are rare.

4.1.2 Data Augmentation

Data augmentation is a practical and effective way to increase dataset diversity, reduce overfitting, and improve robustness to real-world variability in illumination, viewpoint, and partial occlusions. Augmentations were implemented with the Albumentations library [62] and are applied only during training; validation and test images undergo normalization and tensor conversion only.
Global appearance is perturbed through random brightness/contrast and hue– saturation adjustments, encouraging invariance to exposure changes, sensor differences, and lighting conditions typical of urban scenes. To increase scale variability without distorting object geometry, a "zoom-out" operation is included: the image is placed on a larger canvas filled with a constant mean color and then resized back to 512×512. This mimics the expand strategy popularized in single-shot detectors and subsequent computer vision works [63, 64], effectively creating wider spatial context and smaller apparent object scales while preserving aspect ratios.
Additional geometric diversity is introduced via small affine transformations (scaling, translation, and limited rotation), simulating viewpoint and platform motion within ranges that keep annotations reliable and free from unrealistic deformations. Random left–right flips capture road topology symmetry and improve generalization across driving directions. Structured occlusions are modeled with coarse dropout ("cutout"), enhancing resilience to missing evidence in crowded scenes.
Bounding boxes are consistently propagated through these transformations, with invalid boxes removed based on minimum area and visibility thresholds. This prevents label noise from tiny or truncated boxes at the borders and ensures reliable supervision for the detector's receptive field and anchor design.
Overall, this augmentation scheme increases appearance, scale, and occlusion diversity without introducing anisotropic resizing artifacts, ultimately leading to more robust detectors under the operating conditions expected for perception. An example of the augmentations proposed in action can be seen in Figure 4.4.

(a) **(b)** **(c)** **(d)**

Figure 4.4: Examples of data augmentations applied to the Cityscapes dataset. **(a)** Original image. **(b)** Photometric distortion (brightness, contrast, hue–saturation shifts). **(c)** Zoom-out operation with padding and resize. **(d)** Affine jitter including small translations, scaling, and rotations.

4.2 Model Architecture

A typical DL–based object detector, as outlined in Section 2.2.3, consists of two main components: a CNN backbone, which extracts hierarchical feature representations from the input image, and a detection head that uses these features to predict object categories and localize them through bounding box offsets.
In this chapter, the architecture adopted in our work is described in detail. Section 4.2.1 focuses on the chosen backbone, a slimmable variant of MobileNetV2 capable of running at four different widths (0.25×, 0.5×, 0.75×, and 1.0×), with

particular attention to the design and integration of the Dynamic Slimming Gate, proposed by [54]. Section 4.2.2 is dedicated to the detection head, where we employ an SSDLite-512 module and discuss its configuration and role within the overall pipeline.

4.2.1 Backbone

As backbone, we adopt MobileNetV2 [7], previously introduced in Section 2.1, and extend it to operate at multiple widths. To achieve this, each convolutional layer is replaced with a *slice-able* variant capable of dynamically selecting the number of active channels. For practical acceleration, the slice-able design ensures that filters remain contiguous and dense during width selection. As a result, the operation reduces to a standard slicing followed by dense matrix multiplication, which is significantly more efficient in real hardware implementations than sparse indexing or irregular weight selection.
Formally, consider a convolutional layer with N filters and M input channels, represented by $\mathbf{W} \in \mathbb{R}^{N \times M}$ (spatial dimensions are omitted for clarity). Given an input activation $\mathcal{X}$ and a slimming ratio $\rho \in \{0.25, 0.5, 0.75, 1.0\}$, the slice-able convolution can be expressed as:

$$\mathcal{Y} = \mathbf{W}[\,:\lfloor \rho \times N \rfloor\,] \; * \mathcal{X}$$

where only the first $\rho \times N$ filters are selected during inference. In this way, the backbone can seamlessly switch between lightweight and full-capacity configurations, either according to external conditions or as dynamically determined by the output of the Dynamic Slimming Gate.

Dynamic Slimming Gate

The Dynamic Slimming Gate, inspired by the work of Li et al. [54], is integrated at the beginning of the i-th block of the backbone. Its role is to transform the input feature map $\mathcal{X}_i$ into a slimming ratio ρ, which determines the number of active channels in the subsequent layers. At inference time, the gate outputs a categorical score over the candidate ratios L_ρ, and the final width is selected by applying an `argmax`, yielding a one-hot vector that directly specifies the active channel configuration. In other words, the gate functions as an *architecture-routing agent* $\mathcal{A}(\mathcal{X}_i)$, which can be formally expressed as

$$\mathcal{A}(\mathcal{X}_i) = \mathcal{F}(\mathcal{E}(\mathcal{X}_i))$$

where $\mathcal{E}$ encodes the feature maps into a compact representation and $\mathcal{F}$ maps this representation to a one-hot vector used for channel slicing.

Let $\mathcal{X}_i \in \mathbb{R}^{C\times H\times W}$ denote the feature map produced by the i-th block of the backbone. The encoder $\mathcal{E}(\cdot)$ reduces it to a vector $\mathcal{X}_{\mathcal{E},i} \in \mathbb{R}^C$, which is then projected by $\mathcal{F}$ into a categorical distribution. After the one-hot decision is taken, the slimming ratio predicted by the gate is obtained as follows:

$$\rho_i = \mathcal{A}(\mathcal{X}_i) \cdot L_\rho$$

where L_ρ is the set of candidate ratios and $\mathcal{A}(\mathcal{X}_i)$ is the one-hot vector selecting the active width.
The encoder $\mathcal{E}(\cdot)$ is instantiated as global average pooling: the input feature map is spatially averaged to produce a C-dimensional channel descriptor. The feature-mapping function $\mathcal{F}(\cdot)$ is implemented as a lightweight two-layer perceptron realized with consecutive 1×1 convolutions and a ReLU nonlinearity in between. The first projection reduces the C-dimensional descriptor to a compact hidden embedding, and the second maps this embedding to $g = |L_\rho|$ logits. During training, gate probabilities are obtained by applying Gumbel - Softmax to these logits [57]; at inference time, a one-hot selection is produced via `argmax`.
In line with [54], the gate comprises two branches that share the same encoder $\mathcal{E}(\cdot)$:

- *Channel-attention* head: maps the shared embedding back to C channels to produce per-channel reweighting coefficients, which are applied multiplicatively to the features. This branch is fully continuous.
- *Slimming* head: maps the same embedding to $g{=}|L_\rho|$ logits (one per candidate width) and selects the width; (Gumbel-)Softmax is used during training, while `argmax` is used at inference to yield a one-hot choice.

Extra Layers

Following prior work on multi-scale object detection, the backbone is augmented with a small set of additional blocks that produce progressively lower-resolution feature maps consumed by the detection head. This design mirrors the extra prediction layers of SSD [29] and is conceptually related to feature-pyramid methods such as FPN [65].
Concretely, one intermediate feature is tapped from the n-th layer, with n set equal to 13 following [7] (prior to the final 1×1 head), and four additional features are produced by stacked depthwise-separable inverted-residual blocks with stride 2, while preserving the slice-able channel design. The result is a six-level fixed pyramid of feature maps (from high to low resolution) that the head uses for classification and bounding-box regression.
To maintain a stable interface with the detection head (and avoid duplicating heads per width), each feature map is zero-padded along the channel dimension up to a predefined reference size before being forwarded to the head. Padding

preserves dense computation and keeps the head configuration constant across widths, simplifying both implementation and deployment.

4.2.2 Detection Head

As the detection head we adopt an SSDLite module, introduced in Section 2.2.4, configured for 512×512 inputs. The head operates on six multi-scale feature maps: one extracted from the 13-th block of the backbone, one from the final MobileNetV2 head, and four from the extra layers (see Section 4.2.1). Following the standard SSD design, depthwise separable prediction layers are employed to reduce computation. At each pyramid level $k \in \{1, \ldots, 6\}$, anchors follow the SSD defaults [29]: scales s_k linearly spaced in [0.1, 0.9] and aspect ratios $\mathcal{A} = \{1, 2, \frac{1}{2}, 3, \frac{1}{3}\}$. For $a \in \mathcal{A}$, an additional square anchor at $s'_k = \sqrt{s_k\, s_{k+1}}$ (for $k < 6$) is included. Hence each spatial location yields 6 anchors per level. The head predicts class logits and box offsets for all anchors.
The following defaults are used:

- Score threshold = 0.1: minimum class confidence required for a prediction to be considered before non-maximum suppression.
- Non Max Suppression (NMS) threshold = 0.55: intersection-over-union (IoU) threshold used by non-maximum suppression to remove highly overlapping detections of the same class.
- Detections per image = 300: maximum number of final detections retained for each image after post-processing.
- Top-k candidates = 300: number of highest-scoring candidate boxes per image considered prior to applying non-maximum suppression.
- Image normalization (mean/std = $(0.5, 0.5, 0.5)$): per-channel normalization applied to inputs, i.e., $(x - \mu)/\sigma$ with $\mu = \sigma = (0.5, 0.5, 0.5)$.

After per-location predictions, top-k candidates are selected and class-wise NMS (see Section 2.2.4) is applied to obtain the final detections.

4.3 Training Procedure

To obtain a dual-purpose network, a two-phase training scheme inspired by Li et al. [54], introduced in Section 3.2, was adopted and adapted to the object detection task. In *Phase 1*, described in Section 4.3.1, the gating mechanism is disabled and the entire network is trained at multiple widths using the In-Place Ensemble Bootstrapping strategy. In *Phase 2*, described in Section 4.3.2, the backbone and

head weights are frozen and the slimming gate is optimized through the Sandwich Gate Sparsification technique, enabling the model to recognize empty or simple images and dynamically route them to lightweight sub-networks.

4.3.1 Phase 1: Static Mode

In earlier works, the training of slimmable networks was typically carried out using In-Place Distillation combined with the Sandwich Rule [49]. The underlying idea of this approach was to employ the widest sub-network as a teacher for the narrower ones by generating soft targets. Although effective in many cases, this strategy often suffered from convergence issues due to large weight fluctuations in the widest sub-network [3], which could lead to instability and loss explosions in the early stages of training.
To address this limitation, we adopt the In-Place Ensemble Bootstrapping (IEB) strategy. Instead of relying solely on the online network, an Exponential Moving Average (EMA) copy of the model, referred to as the *target network*, is maintained and used to generate soft labels. Denoting by θ the parameters of the online network and by θ' those of the target network, the update rule for the EMA is given by

$$\theta'_t = p\theta'_{t-1} + (1-p)\theta_t$$

where $p \in [0,1]$ is the momentum coefficient and t the training step. The use of the EMA provides more stable and accurate targets compared to the fluctuating predictions of the online network, thereby improving convergence stability and the overall quality of the supervision received by the sub-networks.
Furthermore, following the design choices proposed in related works [66, 67], the training of the slimmest sub-network is enhanced by leveraging an ensemble of the soft labels produced by all sub-networks, rather than relying on a single teacher. This ensemble supervision mitigates the risk of underfitting the smallest configuration and encourages consistent performance across the entire width spectrum.
Formally, at each training step, the widest sub-network (L) is supervised directly with ground-truth labels $\mathcal{Y}$, while the target network provides soft predictions at the largest width to guide the n intermediate sub-networks (I_i).
For the slimmest configuration (S), both classification logits and bounding box predictions are averaged across all wider sub-networks to form ensemble targets:

$$\begin{aligned}
\widetilde{\mathbf{Y}}_{L,I}(\theta') &= \frac{1}{n+1}\left(\mathbf{Y}'_L(\theta') + \sum_{i=1}^{n} \mathbf{Y}'_{I_i}(\theta')\right) \\
\widetilde{\mathbf{B}}_{L,I}(\theta') &= \frac{1}{n+1}\left(\mathbf{B}'_L(\theta') + \sum_{i=1}^{n} \mathbf{B}'_{I_i}(\theta')\right)
\end{aligned}$$

where $\mathbf{Y}$ and $\mathbf{B}$ denote classification logits and bounding box deltas, respectively.

The losses for the three groups of sub-networks are defined as:

$$\mathcal{L}_L^{\text{IEB}}(\theta) = \mathcal{L}_{\text{det}}\left(\mathbf{Y}_L(\theta), \mathbf{B}_L(\theta); \mathcal{Y}\right)$$
$$\mathcal{L}_I^{\text{IEB}}(\theta) = \sum_{i=1}^{n}\Big[\mathcal{L}_{\text{KD-cls}}\left(\mathbf{Y}_{I_i}(\theta), \mathbf{Y}'_L(\theta')\right) + \mathcal{L}_{\text{KD-box}}\left(\mathbf{B}_{I_i}(\theta), \mathbf{B}'_L(\theta')\right)\Big]$$
$$\mathcal{L}_S^{\text{IEB}}(\theta) = \mathcal{L}_{\text{KD-cls}}\left(\mathbf{Y}_S(\theta), \widetilde{\mathbf{Y}}_{L,I}(\theta')\right) + \mathcal{L}_{\text{KD-box}}\left(\mathbf{B}_S(\theta), \widetilde{\mathbf{B}}_{L,I}(\theta')\right)$$

with total loss

$$\mathcal{L} = \mathcal{L}_L^{\text{IEB}} + \mathcal{L}_I^{\text{IEB}} + \mathcal{L}_S^{\text{IEB}}$$

We provide the pseudocode of the training procedure in Algorithm 1, and the overall train process is summarized in Figure 4.5.

In object detection, the background class typically dominates, and a failure to correctly distinguish foreground objects from background regions can overwhelm the error signal. To address this, we follow the recommendations of Chen et al. [68] for the classification distillation term ($\mathcal{L}_{\text{KD-cls}}$), employing a class-weighted cross-entropy loss:

$$\mathcal{L}_{\text{KD-cls}} = -\sum_c \omega_c \, P_t^{(c)} \log P_s^{(c)}$$

where P_t and P_s denote the teacher and student probabilities, respectively, and ω_c are per-class weights. This weighting scheme reduces the tendency of the model to misclassify foreground instances as background, which is far more detrimental than confusion among foreground categories.

For the regression distillation term ($\mathcal{L}_{\text{KD-box}}$), the Smooth-$\ell_1$ loss is adopted, consistent with other works [26, 27, 29].

The outcome of *Phase 1* is a single slimmable detector whose parameters are jointly optimized across all candidate widths. Each sub-network learns to mimic the predictions of the largest configuration through distillation. This provides a unified backbone that can later be coupled with the Dynamic Slimming Gate in *Phase 2* to enable adaptive width selection at inference time.

4.3.2 Phase 2: Adaptive Mode

In *Phase 2*, all model parameters are frozen except for the slimming head of the gate. To endow the model with the ability to recognize simple or empty images, training proceeds by optimizing the detector with the standard detection loss ($\mathcal{L}_{\text{det}}$) and, in addition, a complexity penalty ($\mathcal{L}_{\text{cplx}}$) that discourages the selection of computationally expensive widths. Moreover, following Li et al., the Sandwich Gate Sparsification (SGS) technique is adapted to the object detection setting to explicitly guide the gate toward minimal width on easy inputs and maximal width otherwise.

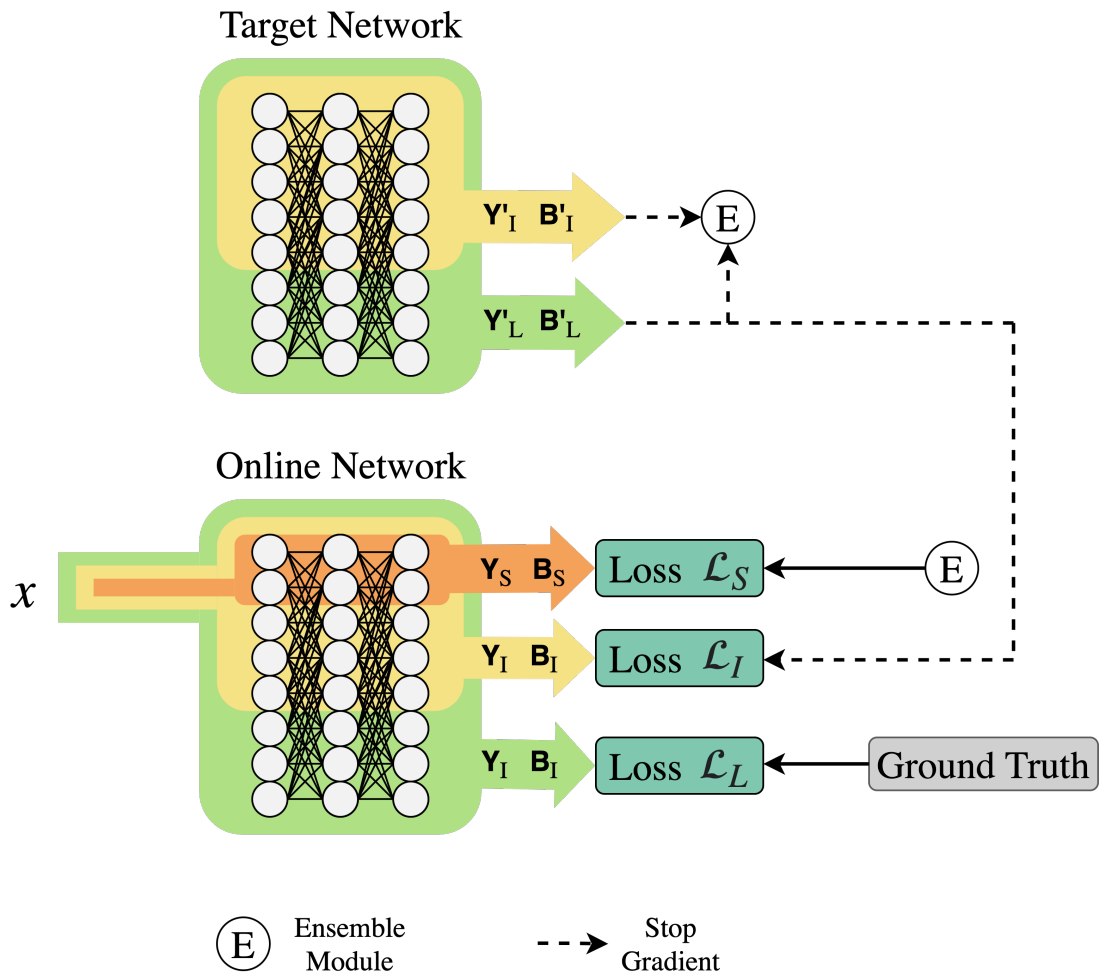


Figure 4.5: Training process of slimmable detector with In-place Ensemble Bootstrapping.

The core idea is to supervise the slimming head directly by comparing its output with a target that depends on the sample type. Dataset samples are partitioned into two groups:

- Labeled samples ($\mathcal{X}_{\text{Labeled}}$), i.e. images containing at least one bounding box.
- Empty samples ($\mathcal{X}_{\text{Empty}}$), i.e. images without any bounding boxes.

To minimize computation, the ground-truth target for the slimming head routes empty (*Easy*) images to the slimmest sub-network:

$$\mathcal{T}(\mathcal{X}_{\text{Empty}}) = [1, 0, \ldots, 0]$$

Algorithm 1 Phase 1: Training with In-Place Ensemble Bootstrapping (IEB)

Require: Model M with width set $\mathcal{W} = \{w_S, w_{I_1}, \ldots, w_L\}$; $\alpha \in [0,1]$.

Initialize online params θ and EMA params $\theta' \leftarrow \theta$.
for each mini-batch $(\mathbf{x}, \mathcal{Y})$ **do**
  Set width w_L; compute detector loss $\mathcal{L}_L^{\text{IEB}} \leftarrow \mathcal{L}_{\text{det}}(M_{w_L}(\mathbf{x}; \theta), \mathcal{Y})$
  Compute EMA logits/boxes with $M(\cdot\,; \theta')$ at w_L and $\{w_{I_i}\}$
  for each w_{I_i} **do**
    Set width w_{I_i}; compute

$$\mathcal{L}_{I_i} = \mathcal{L}_{\text{KD-cls}}\left(\mathbf{Y}_{I_i}(\theta), \mathbf{Y}'_L(\theta')\right) + \mathcal{L}_{\text{KD-box}}\left(\mathbf{B}_{I_i}(\theta), \mathbf{B}'_L(\theta')\right)$$

  end for
  Form ensemble targets from EMA predictions $\widetilde{\mathbf{Y}}_{L,I}$ and $\widetilde{\mathbf{B}}_{L,I}$
  Set width w_S; compute

$$\mathcal{L}_S = \mathcal{L}_{\text{KD-cls}}\left(\mathbf{Y}_S(\theta), \widetilde{\mathbf{Y}}_{L,I}\right) + \mathcal{L}_{\text{KD-box}}\left(\mathbf{B}_S(\theta), \widetilde{\mathbf{B}}_{L,I}\right)$$

  Compute total loss $\mathcal{L} = \mathcal{L}_L^{\text{IEB}} + \sum_i \mathcal{L}_{I_i} + \mathcal{L}_S$.
  Update online params $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$
  EMA update: $\theta' \leftarrow \alpha\,\theta' + (1-\alpha)\,\theta$
end for

Conversely, labeled images are encouraged to use the widest sub-network:

$$\mathcal{T}(\mathcal{X}_{\text{Labeled}}) = [0,\, 0,\, \ldots,\, 1]$$

The final SGS loss is defined as

$$\begin{aligned}\mathcal{L}_{\text{SGS}}(\mathcal{X}) = {} & \mathbb{T}_{\text{Is-Empty}}(\mathcal{X}) \cdot \mathcal{L}_{\text{CE}}\left(\mathcal{X}, \mathcal{T}(\mathcal{X}_{\text{Empty}})\right) \\ & + \left(1 - \mathbb{T}_{\text{Is-Empty}}(\mathcal{X})\right) \cdot \mathcal{L}_{\text{CE}}\left(\mathcal{X}, \mathcal{T}(\mathcal{X}_{\text{Labeled}})\right)\end{aligned}$$

where $\mathbb{T}_{\text{Is-Empty}}(\mathcal{X}) \in \{0,1\}$ indicates whether $\mathcal{X}$ is an empty image, and

$$\mathcal{L}_{\text{CE}}(\mathcal{X}, \mathcal{T}) = -\sum_c \mathcal{T}^{(c)} \log\left(\mathcal{P}^{(c)}(\mathcal{X})\right)$$

is the cross-entropy between the gate's softmax scores $\mathcal{P}(\mathcal{X})$ and the one-hot target $\mathcal{T}$. The overall objective for *Phase 2* is then

$$\mathcal{L} = \mathcal{L}_{\text{det}} + \alpha\,\mathcal{L}_{\text{cplx}} + \beta\,\mathcal{L}_{\text{SGS}}$$

with $\alpha = 4$ and $\beta = 2$ in our experiments.

To optimize the non-differentiable slimming head of the dynamic gate within the object detection framework, the Gumbel–Softmax reparameterization [57] is adopted, replacing the discrete `argmax` with a differentiable softmax during backpropagation. In addition, following [54], Gumbel noise is injected during training to prevent the gate from collapsing into a static routing policy.
The outcome of *Phase 2* is a fully trained Dynamic Slimming Gate capable of adaptively routing each input to either the slimmest or the widest sub-network at inference time, depending on its estimated difficulty. Whereas *Phase 1* produces a single detector that can operate at multiple widths with consistent predictions, *Phase 2* equips this model with a learned policy that trades off accuracy and computational cost on a per-image basis, typically steering easy inputs toward the minimal width and hard inputs toward the maximal width. In practice, this policy can be deployed in different ways: for example, it may use the slimmest subnet to refine potentially misclassified *Easy* frames or, more aggressively, skip such frames altogether to maximise computational savings. As a result, the same backbone can be run either statically at a fixed width or dynamically under a computational budget, depending on the operational constraints.

Chapter 5

Results

This chapter presents the experimental results and analysis of the proposed slimmable detection network.
In Section 5.1, we examine the outcomes of *Phase 1* training with In-Place Ensemble Bootstrapping (see Section 4.3.1). Results are analyzed both qualitatively and quantitatively, with a focus on the performance variation across the four sub-models and the potential energy savings achieved by deploying narrower configurations.
Section 5.2 reports the results obtained after *Phase 2* training of the Dynamic Slimming Gate (see Section 4.3.2). Here, we evaluate the effectiveness of the adaptive mode, highlighting how the gate autonomously balances accuracy and efficiency depending on the input.
Finally, in Section 5.3, we provide a direct comparison between the two modes, contrasting their detection accuracy, computational cost, and energy footprint.

5.1 Static Mode

For our Slimmable MobileNetV2 backbone, as described in Section 4.2.1, the slimming gate is placed after the fifth depthwise separable convolution block. All preceding layers operate with a fixed slimming ratio $\rho = 0.50$, while the subsequent layers are controlled by the gate with candidate ratios $\rho \in \{0.25, 0.50, 0.75, 1.0\}$. The model is initialized with COCO-pretrained weights [35], as this has proven to be an effective strategy for transfer learning on urban datasets such as Cityscapes [69].

5.1.1 Training Details

The supernet was trained for a total of 150 epochs with a batch size of 32. We used SDG optimizer with learning rate of 0.02, momentum of 0.9, and weight decay

of 5×10^{-4}. A cosine annealing learning rate schedule was adopted, following [54], with a minimum learning rate set to 1% of the initial value. The momentum coefficient for the EMA model was set to $p = 0.997$.
For the classification loss, per-class weights were used to mitigate foreground – background imbalance, as adviced by [68]: the background class was up-weighted ($\omega_0 = 1.5$), while all foreground classes were assigned $\omega_c = 1$.

5.1.2 Results

To the best of our knowledge, no prior work has reported object detection experiments on a dataset of this kind, making it necessary to establish a reference point. For benchmarking, we train a non-slimmable MobileNetV2 SSDLite-512 baseline, which retains the same gating structure as our slimmable model but is fixed to the maximum width, effectively operating as a single-width static network. This baseline therefore serves as a proxy for a conventional detector without width adaptivity, against which we can assess the benefits of our approach. It achieves an mAP of 21.57% with a per-pass cost of 1.76 GMACs.
Table 5.1 reports the performance of our slimmable MobileNetV2 SSDLite-512 across its four candidate widths. As expected, accuracy increases with width, reaching 21.96% mAP at full capacity (1.0×), but at the highest computational cost of 1.44 GMACs per pass. Reducing the width yields substantial savings: the 0.25× configuration requires only 0.38 GMACs (a 73.6% reduction) but drops to 3.76% mAP, showing that extremely narrow models lose much of the representational capacity. Intermediate settings provide more favorable trade-offs: 0.50× reduces compute by 59.0% while retaining 41.2% of the full-width accuracy, and 0.75× achieves 13.44% mAP with a 34.7% cost reduction (retaining 61.2% of the full-width accuracy).

Width	**mAP (%)**	**GMACs/forward**	**Saving vs SNN 1.00× (%)**
0.25×	2.09	0.38	73.6
0.50×	9.04	0.59	59.0
0.75×	14.12	0.94	34.7
1.00×	21.96	1.44	—
Static 1.00×	21.57	1.76	—

Table 5.1: Slimmable MobileNetV2 SSDLite-512 performance at different widths. Savings are computed with respect to the slimmable 1.00× path (1.44 GMACs). The static baseline retains the same gating structure but is fixed to the maximum width, acting as a conventional single-width detector.

The slight gain at 1.0× over the single-width baseline is consistent with training

under IEB, where an EMA teacher provides temporally smoothed supervision. EMA-based teachers are indeed known to stabilize optimization and can yield modest generalization improvements [70].

Class	**Test %**	**0.25×**	**0.5×**	**0.75×**	**1.0×**
Person	30.6%	0.0617	0.1216	0.1716	0.2482
Car	48.0%	0.1850	0.2919	0.3546	0.4510
Truck	1.0%	0.0117	0.0276	0.0587	0.1141
Rider	5.1%	0.0126	0.0524	0.1282	0.2235
Motorcycle	1.4%	0.0022	0.0184	0.0506	0.1153
Bicycle	12.5%	0.0066	0.0538	0.0929	0.1839
Bus	1.2%	0.0206	0.1355	0.1751	0.3047
Train	0.3%	0.0004	0.0220	0.0437	0.1159

Table 5.2: Per-class mAP for the Slimmable MobileNetV2 SSDLite-512 at different widths, alongside the proportion of each class in the test set. Classes in the long tail (Truck, Bus, Train, Motorcycle) show markedly lower mAP, reflecting both model capacity and data scarcity.

Table 5.2 further breaks down the results by category, showing the per-class mAP for each candidate width. Frequent categories such as *Car* and *Person* consistently dominate detection performance at all widths, reaching 0.4510 and 0.2482 mAP respectively at full capacity. In contrast, rare or small-scale categories like *Train*, *Motorcycle*, and *Truck* achieve much lower scores, particularly at narrower widths, reflecting the compounded effect of data scarcity and reduced representational capacity. Intermediate widths (0.5× and 0.75×) offer a balanced compromise: they already surpass the narrowest setting by a large margin on most classes, and begin to recover performance on mid-frequency categories such as *Bicycle* and *Rider*. This breakdown highlights how width scaling disproportionately affects rare or fine-grained categories, while high-frequency classes remain relatively robust even in slimmest configurations.

From a qualitative perspective, the predictions of the different sub-networks exhibit consistent patterns depending on the input characteristics. In simple and well-defined scenes containing only clearly delineated foreground objects, the performance across all widths is often comparable, if not almost identical, which supports the use of the slimmest configuration for such cases (Figure 5.1(a)). Under good lighting conditions, the intermediate 0.75× sub-network frequently serves as an excellent substitute for the full 1.0× model, achieving nearly identical results while reducing the computational cost by 34.7% (Figure 5.1(b)). Conversely, as expected, the slimmest configuration shows a marked performance drop in crowded

or visually complex environments, which highlights its reduced representational capacity and reveals the limitations of aggressive channel reduction when fine-grained discrimination is required (Figure 5.1(c)).
This trend is particularly evident for rare classes such as *Bus*, *Truck* and *Train*, where the widest configuration maintains a clear advantage (Figure 5.2(b)). Moreover, all sub-networks except the widest struggle on two specific scenarios: poorly lit images and the detection of small background objects (Figure 5.2(a–c)). Together, these observations highlight the importance of dynamically adapting width to the input to balance efficiency and accuracy.

Figure 5.1: Qualitative predictions of the slimmable model in *Static Mode* at different slimming ratios $\rho \in \{0.25, 0.50, 0.75, 1.0\}$. Each column shows a representative image with ground truth (GT) on the first row and model predictions below. (a) Example with a single, clearly visible object. (b) Moderately dense and well lit scene. (c) Crowded scene with multiple objects.

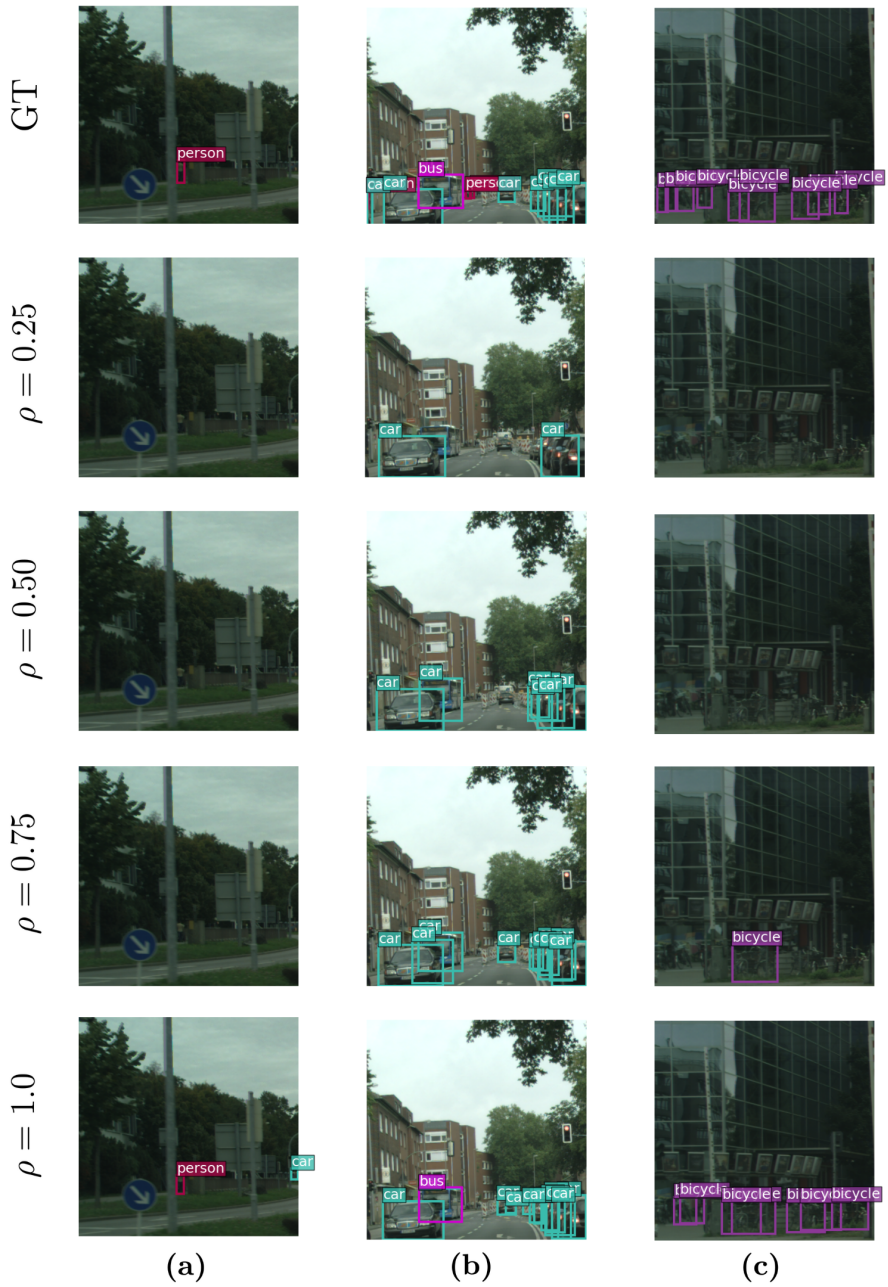


Figure 5.2: Qualitative predictions of the slimmable model in *Static Mode* at different slimming ratios ρ. The layout follows Figure 5.1. (a) Small object scenario. (b) Urban scene with multiple cars and one highly infrequent object class. (c) Poorly lit image with reduced contrast.

5.2 Adaptive Mode

In contrast to the static evaluation of fixed-width subnetworks, the *Adaptive Mode* enables the model to dynamically select between the widest and the slimmest width on a per-image basis. Instead of executing a predetermined configuration for all inputs, a lightweight gate predicts the most appropriate subnetwork according to the estimated difficulty or content of each image. This mechanism allows the system to allocate computational resources adaptively: trivial or empty frames can be processed by the slimmest network, while more complex scenes exploit the full capacity of the model. The following section reports and analyses the results obtained after training this gating mechanism (*Phase 2*), evaluating both its impact on detection accuracy and the achieved computational savings.

5.2.1 Training Details

For the second training stage (Adaptive Mode), we start from the weights of the supernet trained in *Phase 1* and freeze all parameters except those of the slimming gate. This allows the gate to learn how to route each input to the most appropriate sub-network without altering the feature extractors.
Training is performed until convergence with a batch size of 1, as each image is processed individually to determine its routing difficulty. To stabilise the training and approximate a larger batch size, gradients are accumulated over 512 images before each optimisation step. Following [54], we optimize only the gate parameters using stochastic gradient descent with a learning rate of 0.05 (with a momentum of 0.9) which decays to 0.9× of its value in every epoch. In our experiments, the gate typically converged within approximately 15 epochs, after which both the routing policy and detection performance stabilized.
As explained in Section 4.3.2, the objective combines three terms: the standard detection loss $\mathcal{L}_{\text{det}}$, the complexity penalty $\mathcal{L}_{\text{cplx}}$ (which discourages the selection of computationally expensive widths), and the Sandwich Gate Sparsification loss $\mathcal{L}_{\text{SGS}}$, which supervises the gate to favour the slimmest path for empty or trivial images and the widest path otherwise. In our experiments, the loss weights are set to $\alpha = 2.0$ for $\mathcal{L}_{\text{cplx}}$ and $\beta = 4.0$ for $\mathcal{L}_{\text{SGS}}$. The computational cost of each width configuration is normalised using a pre-computed GFLOPs table.
During training, the gate output is computed using the differentiable Gumbel–Softmax reparameterization [57], allowing gradients to flow through the categorical selection. To identify easy samples, an image is labeled as *Easy* not simply because it contains no ground-truth objects, but only if the slimmest sub-network correctly predicts it as empty (i.e., with no false positives), as in Figure 5.3. In this way, the gate is guided by reliable targets derived from the smallest configuration's behaviour.

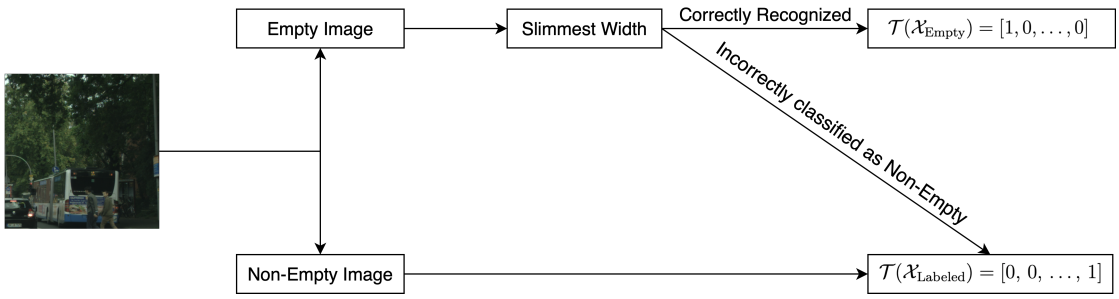


Figure 5.3: Routing policy used to train the Dynamic Slimming Gate. Empty images correctly recognized by the slimmest sub-network are assigned the minimal-width target $\mathcal{T}(\mathcal{X}_{\text{Empty}})$, whereas non-empty or misclassified ones are assigned the maximal-width target $\mathcal{T}(\mathcal{X}_{\text{Labeled}})$.

5.2.2 Results

As in Section 5.1, we also apply the *Phase 2* training procedure to the static network for benchmarking purposes. In this setting, the adaptive version attempts to route empty images to the slimmest configuration, thereby reducing the average computational cost to 1.59 GMACs (−9.7%) with a slight decrease in accuracy to 21.18% mAP (−1.8% vs the 21.57% of the Static 1.00× baseline). This baseline policy classifies about ≈11% of test images as empty, correctly identifying only ≈21% of the truly empty ones.

These results are unsurprisingly suboptimal: since the static model has never been trained at the smallest width, it lacks the capacity to fully converge in this configuration. Moreover, the complete absence of a detection-loss component supervising the smallest path further hinders stable convergence and weakens the model's ability to learn an effective gating policy.

Our slimmable model on the other hand, by leveraging the trained slimmest subnet, routes 1 884/4 000 images (47.1%) through the minimal-width path, thereby lowering computational cost and correctly identifying approximately 57% of the empty images.

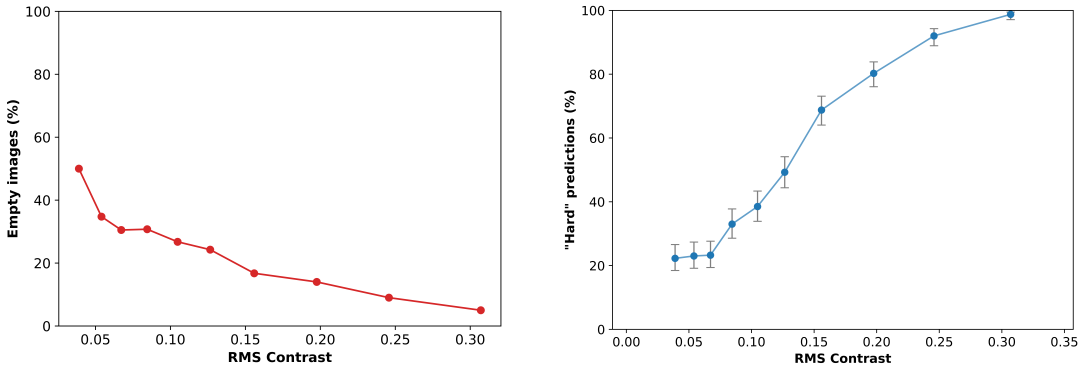
In this context, the adaptive mode can be operated in two distinct ways. The default and less aggressive configuration, which we term *Try-Best*, is the one for which the gate has been trained: images classified as *Easy* are still processed by the slimmest sub-network rather than being discarded, allowing the system to save computation while retaining the possibility to recover detection errors on non-trivial cases. A second, more aggressive configuration, which we term *Early-Exit*, treats images classified as *Easy* by the gate as effectively empty or uninformative, bypassing further detection altogether. This policy maximizes computational savings but may sacrifice accuracy on borderline cases.

Table 5.3 summarises the results obtained under both configurations. Under *Try-Best*, accuracy is slightly higher at 16.90% mAP (a 23.0% drop vs. the slimmable 1.0×), with a compute cost of 0.93 GMACs per image (a 35.5% reduction vs. 1.44 GMACs). Under *Early-Exit*, the model attains an mAP of 16.64% (a 24.2% drop relative to the 1.0× slimmable setting at 21.96), while reducing the average compute to 0.85 GMACs per image, a 41.0% saving with respect to the 1.44 GMACs full-width run.

Configuration	Try-Best	Early-Exit	mAP	GMAC/img
1.0× Static		✓	21.18	1.59
Slimmable	✓		16.90	0.93
Slimmable		✓	16.64	0.85

Table 5.3: Adaptive-mode results under the two routing policies. The 1.0× static benchmark run is included for reference.

From a qualitative point of view, combining gate decisions with the Static Mode predictions reveals consistent patterns. Scenes with low structural complexity or homogeneous appearance (e.g., mostly asphalt or mostly vegetation) are frequently routed as *Easy* by the gate (Figure 5.5(a–b)), where skipping or using the slimmest subnet has limited impact on accuracy. Conversely, heterogeneous urban scenes (mixed road, street furniture, foliage, facades) tend to be handled conservatively and are more often routed as *Hard* (Figure 5.6(a–b)), in line with the observation that wider subnets better handle small, crowded, or poorly lit objects.



(a) **(b)**

Figure 5.4: Relationship between RMS Contrast, which goes from 0 to 0.5, and gate behavior. (a) Percentage of empty images across RMS contrast bins, showing that low-contrast scenes are more likely to be visually empty. (b) Percentage of samples classified as *Hard* by the gate, which increases with contrast.

A possible explanation for this behaviour lies in the architecture of the Dynamic Slimming Gate itself. Because it aggregates spatial information via global average pooling to form a compact channel descriptor, its internal statistics primarily capture global activation patterns (e.g., texture richness and contrast diversity), rather than precise spatial structure [55, 71]. Consequently, low heterogeneity can serve as a plausible, though imperfect, proxy for trivial or empty frames. This mechanism can be indeed prone to misrouting: visually uniform but semantically rich images (e.g. low-contrast backgrounds with small targets) or cluttered yet empty scenes may be wrongly classified, with empty images marked as *Hard* (Figure 5.6(c)) and non-empty ones sent to *Easy* (Figure 5.5(c)).
To further investigate this hypothesis, we analyzed the relationship between the gate's predictions and the average image contrast, measured in terms of RMS contrast. As shown in Figure 5.4, the percentage of empty images decreases steadily as contrast increases, while the proportion of samples classified as *Hard* follows the opposite trend. This correlation strongly suggests that the gate implicitly relies on global contrast statistics when estimating image difficulty and the presence of objects in it. In practice, this behaviour is also consistent with dataset characteristics: low-contrast scenes (e.g., uniform asphalt or dense vegetation) are often truly empty, whereas high-contrast ones tend to correspond to structured urban views containing objects and edges. Therefore, while contrast is not an explicit supervision signal, it emerges as a strong latent cue guiding the gate's routing decisions.

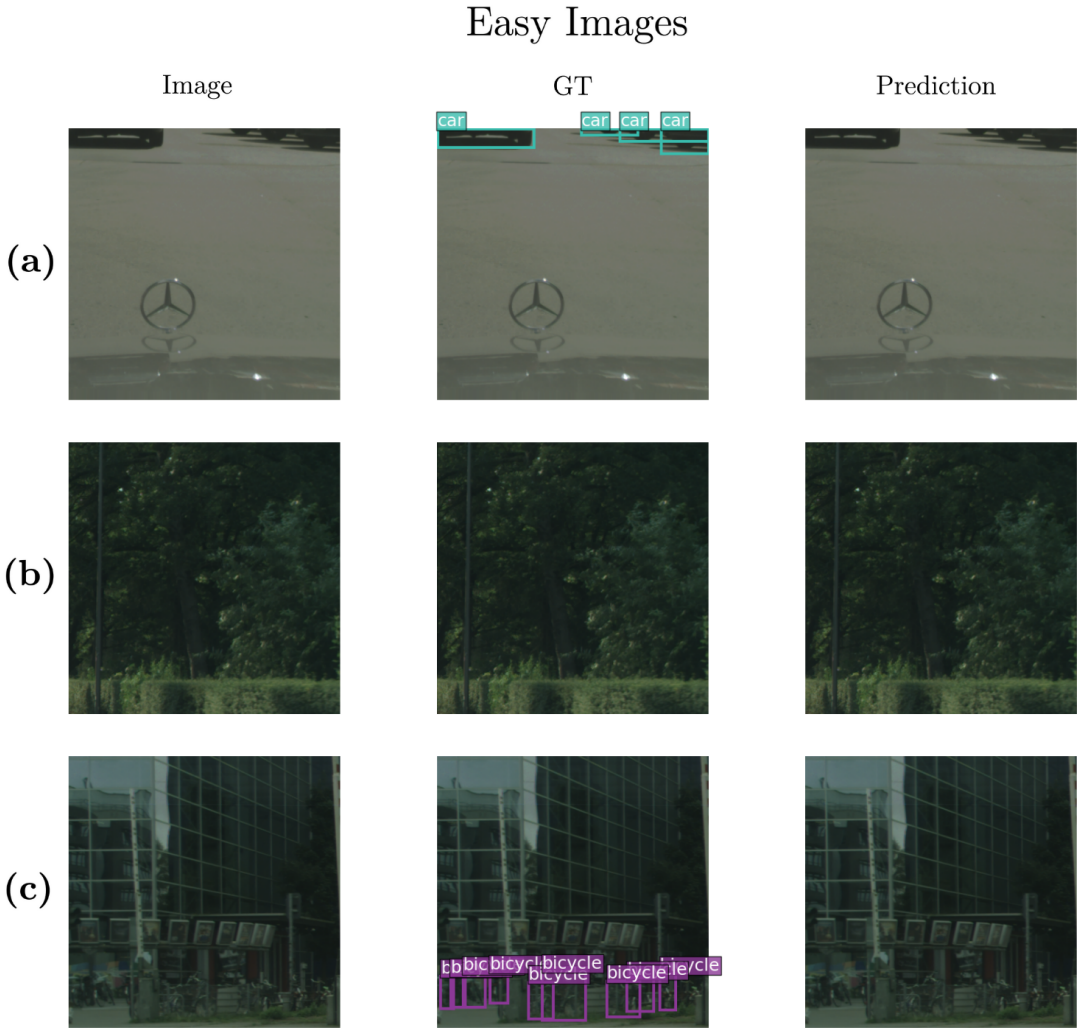


Figure 5.5: Qualitative examples of images routed as *Easy* by the gate. (a) Predominantly asphalt scene with few salient features. (b) Homogeneous vegetation background with no foreground targets. (c) Visually uniform yet semantically non-empty frame, containing small and infrequent objects, misclassified as *Easy*.

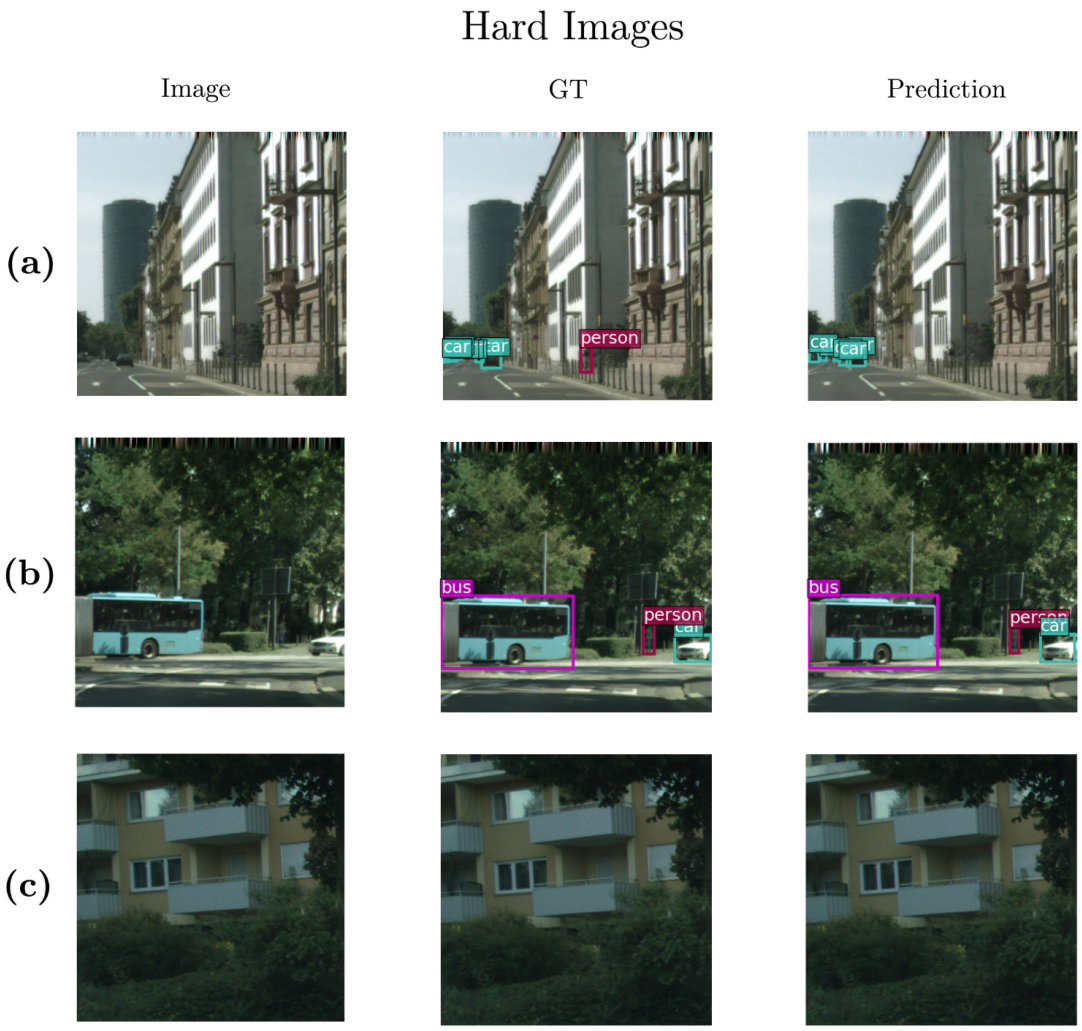


Figure 5.6: Qualitative examples of images routed as *Hard* by the gate. (a) Urban street with mixed structures (road, facades, street furniture) and multiple small objects. (b) Scene containing a large infrequent class (*Bus*) alongside pedestrians and cars. (c) Cluttered yet empty background in low-light conditions, wrongly classified as *Hard*.

5.3 Comparative Results

Figure 5.7 summarizes the accuracy–compute trade–off (mAP vs. average GMACs per forward) for the *Slimmable* model in both Static and Adaptive modes, alongside the non–slimmable *Baseline*. The static slimmable runs (circles) delineate a clean operating frontier: as width increases, mAP improves with diminishing returns and a knee near the 0.75× setting. Placing the adaptive policies (squares) on the same axes shows that input–conditioned routing can yield more favorable points at comparable budgets: by sending easy frames to cheaper subnets and reserving wide subnets for difficult scenes, both *Early-Exit* and *Try-Best* achieve higher average accuracy than fixed widths around the $0.85-0.95$ GMAC/img range. The non–slimmable baseline (diamonds) exhibits limited movement when attempting to skip empty images, confirming that without narrower subnets there is little headroom for computational savings.

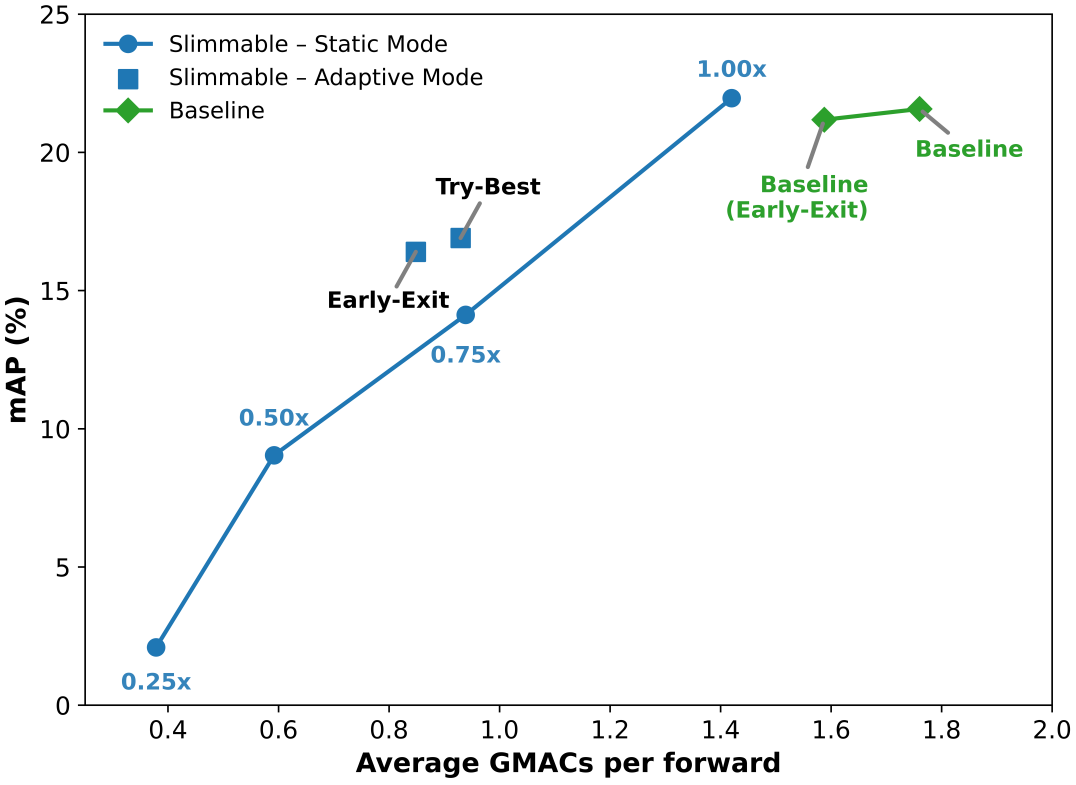


Figure 5.7: Accuracy–compute trade–off for Static and Adaptive modes.

The two modes are complementary and can coexist in a single system: Static Mode is preferable when the budget is fixed and predictable (e.g., a known operation

time), and we can assume a stable input distribution (e.g., a specific patrol route or inspection profile). It offers deterministic latency, simpler validation, and straightforward worst–case guarantees—properties that can be important for certification and mission planning. Adaptive Mode, on the other hand, is advantageous when the scene difficulty varies across time or routes, and when average rather than worst–case compute matters. In these settings, gating converts heterogeneity in the stream into compute savings and/or accuracy gains at the same average cost. Between the two policies, *Early-Exit* favors efficiency (lower GMAC/img) with a small mAP penalty, whereas *Try-Best* spends slightly more compute to recover additional detections.
In other words, if the deployment can rely on a fixed budget and a well–characterized input profile, a static width is adequate and easier to certify. If the operational conditions are variable or uncertain, enabling the gate provides a strictly better average accuracy–efficiency trade–off, effectively shifting the operating point above the static frontier.

Chapter 6

Conclusions

Designing DNNs for mobile and embedded devices remains a challenging task. Practical constraints such as limited compute capacity, strict power budgets, and heterogeneous hardware architectures often require developing separate models for each platform or usage scenario. This fragmentation increases deployment complexity and memory footprint, particularly when multiple performance–energy trade-offs are required on the same device.

This work tries to address these issues by designing and validating a *slimmable* approach to object detection tailored for medium–low altitude drone applications. By integrating techniques such as In-Place Ensemble Bootstrapping, knowledge distillation, and channel-wise slicing, we developed a MobileNetV2–SSDLite512 model capable of operating efficiently across four predefined widths using a single set of shared parameters. Each configuration, from the slimmest to the full-capacity network, was trained to produce consistent predictions, effectively allowing the convolutional weights to be "sliced" on demand to match different computational budgets. In the static mode, the detector achieved mAP scores of 2.09%, 9.04%, 14.12%, and 21.96% across the 0.25×, 0.50×, 0.75×, and 1.00× widths, respectively, with corresponding computational costs of 0.38, 0.59, 0.94, and 1.44 GMACs per image. This demonstrates a clear trade-off between accuracy and efficiency, confirming that meaningful predictions can be retained even under substantial computational reduction.

Beyond static scalability, we further extended the model with a *Dynamic Slimming Gate*, enabling adaptive inference. This module learns to analyze incoming feature maps and route each input either through the widest or the slimmest subnet, depending on its estimated difficulty. In practice, this mechanism allows the detector to process visually complex scenes with full capacity, while reducing computation on non-informative images. The resulting framework therefore supports both a static deployment at a fixed cost and a dynamic one, where accuracy and efficiency are jointly optimized on a per-image basis.

In adaptive mode, the network can operate under two different configurations: the standard *Try-Best*, which processes *Easy* images through the slimmest sub-network, and the more aggressive *Early-Exit*, which omits them entirely to maximize efficiency. Under *Try-Best*, the model achieves 16.90% mAP (a 23.0% decrease relative to the slimmable 1.00× setting at 21.96%) with an average cost of 0.93 GMACs/img (−35.5%). Conversely, *Early-Exit* attains 16.64% mAP (−24.2% vs. 1.00×) with just 0.85 GMACs/img, corresponding to a 41.0% reduction in compute.

Such flexibility opens multiple operational modes, depending on both external and internal factors. Exogenous conditions such as remaining battery life, mission duration, or communication bandwidth may dictate a preferred energy profile, while endogenous conditions, determined by the input itself, can trigger automatic adaptation to preserve efficiency without compromising accuracy. This dual adaptivity makes the approach well suited for resource-constrained, autonomous platforms where computation must be allocated intelligently and in real time.

Overall, the proposed methodology demonstrates that width-adaptive object detection can provide a unified, lightweight, and versatile alternative to maintaining multiple specialized models. By exploiting a single shared backbone capable of scaling its complexity dynamically, we enable flexible deployment across a wide range of operating conditions without sacrificing performance consistency.

Further research could focus on enhancing the per-width representational quality and designing a more robust gating mechanism, less sensitive to activation statistics and more attuned to spatial context. Exploring mixed-precision kernels, quantization-aware training, and joint optimization of energy–latency objectives could also extend the model's applicability to ultra-low-power devices and real-time drone missions.

Bibliography

[1] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas S. Huang. «Slimmable Neural Networks». In: *CoRR* abs/1812.08928 (2018). arXiv: 1812.08928. URL: http://arxiv.org/abs/1812.08928 (cit. on pp. 1, 28–30, 34).

[2] Peng Wang et al. «Unifying Architectures, Tasks, and Modalities Through a Simple Sequence-to-Sequence Learning Framework». In: *CoRR* abs/2202.03052 (2022). arXiv: 2202.03052. URL: https://arxiv.org/abs/2202.03052 (cit. on pp. 1, 31).

[3] Jiahui Yu et al. «BigNAS: Scaling up Neural Architecture Search with Big Single-Stage Models». In: *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII*. Glasgow, United Kingdom: Springer-Verlag, 2020, pp. 702–717. ISBN: 978-3-030-58570-9. DOI: 10.1007/978-3-030-58571-6_41. URL: https://doi.org/10.1007/978-3-030-58571-6_41 (cit. on pp. 1, 32, 46).

[4] Xin He, Kaiyong Zhao, and Xiaowen Chu. «AutoML: A Survey of the State-of-the-Art». In: *CoRR* abs/1908.00709 (2019). arXiv: 1908.00709. URL: http://arxiv.org/abs/1908.00709 (cit. on pp. 1, 31).

[5] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929 (cit. on p. 1).

[6] Lanlan Liu and Jia Deng. «Dynamic Deep Neural Networks: Optimizing Accuracy-Efficiency Trade-offs by Selective Execution». In: *CoRR* abs/1701.00299 (2017). arXiv: 1701.00299. URL: http://arxiv.org/abs/1701.00299 (cit. on pp. 2, 29).

[7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV]. URL: https://arxiv.org/abs/1801.04381 (cit. on pp. 2, 3, 6–9, 23, 43, 44).

[8] Muhammet Taha Topalli, Mehmet Yilmaz, and Muhammed Fatih Çorapsiz. «Real Time Implementation of Drone Detection using TensorFlow and MobileNetV2-SSD». In: *2021 7th International Conference on Electrical, Electronics and Information Engineering (ICEEIE)*. 2021, pp. 436–439. DOI: 10.1109/ICEEIE52663.2021.9616846 (cit. on p. 3).

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications». In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861 (cit. on pp. 3, 8).

[10] Francois Chollet. «Xception: Deep Learning With Depthwise Separable Convolutions». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017 (cit. on p. 3).

[11] Ramesh G, Jeswin Y, Divith R Rao, B.R Suhaag, Daksh Uppoor, and Kiran Raj K M. «Real Time Object Detection and Tracking Using SSD Mobilenetv2 on Jetbot GPU». In: *2024 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*. 2024, pp. 255–260. DOI: 10.1109/DISCOVER62353.2024.10750771 (cit. on p. 3).

[12] Anusree Kanadath, J. Angel Arul Jothi, and Siddhaling Urolagin. «Histopathology Image Segmentation Using MobileNetV2 based U-net Model». In: *2021 International Conference on Intelligent Technologies (CONIT)*. 2021, pp. 1–8. DOI: 10.1109/CONIT51480.2021.9498341 (cit. on p. 3).

[13] Andrew Howard et al. *Searching for MobileNetV3*. 2019. arXiv: 1905.02244 [cs.CV]. URL: https://arxiv.org/abs/1905.02244 (cit. on pp. 3, 23).

[14] Danfeng Qin et al. *MobileNetV4 – Universal Models for the Mobile Ecosystem*. 2024. arXiv: 2404.10518 [cs.CV]. URL: https://arxiv.org/abs/2404.10518 (cit. on p. 3).

[15] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. *NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications*. 2018. arXiv: 1804.03230 [cs.CV]. URL: https://arxiv.org/abs/1804.03230 (cit. on p. 3).

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. «Attention is all you need». In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964 (cit. on p. 3).

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385 (cit. on pp. 4, 7).

[18] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. 2019. arXiv: 1803.08375 [cs.NE]. URL: https://arxiv.org/abs/1803.08375 (cit. on p. 5).

[19] Rui Kong, Yuanchun Li, Yizhen Yuan, and Linghe Kong. «ConvReLU++: Reference-based Lossless Acceleration of Conv-ReLU Operations on Mobile CPU». In: *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. MobiSys '23. Helsinki, Finland: Association for Computing Machinery, 2023, pp. 503–515. ISBN: 9798400701108. DOI: 10.1145/3581791.3596831. URL: https://doi.org/10.1145/3581791.3596831 (cit. on p. 6).

[20] Baoye Song, Jianyu Chen, Weibo Liu, Jiangzhong Fang, Yani Xue, and Xiaohui Liu. «YOLO-ELWNet: A lightweight object detection network». In: *Neurocomputing* 636 (Mar. 2025), p. 129904. DOI: 10.1016/j.neucom.2025.129904 (cit. on p. 6).

[21] Tony Lindeberg. «Scale Invariant Feature Transform». In: vol. 7. May 2012. DOI: 10.4249/scholarpedia.10491 (cit. on p. 12).

[22] 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. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177 (cit. on p. 13).

[23] 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*. Vol. 1. 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517 (cit. on p. 13).

[24] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV]. URL: https://arxiv.org/abs/1311.2524 (cit. on p. 13).

[25] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, and Brian Lee. *A Survey of Modern Deep Learning based Object Detection Models*. 2021. arXiv: 2104.11892 [cs.CV]. URL: https://arxiv.org/abs/2104.11892 (cit. on pp. 14, 15).

[26] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV]. URL: https://arxiv.org/abs/1504.08083 (cit. on pp. 13, 47).

[27] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV]. URL: https://arxiv.org/abs/1506.01497 (cit. on pp. 14, 47).

[28] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640 (cit. on pp. 15, 16).

[29] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. «SSD: Single Shot MultiBox Detector». In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 21–37. ISBN: 9783319464480. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2 (cit. on pp. 16, 17, 22, 44, 45, 47).

[30] Lilian Weng. «Object Detection Part 4: Fast Detection Models». In: *lilianweng.github.io* (2018). URL: https://lilianweng.github.io/posts/2018-12-27-object-recognition-part-4/ (cit. on pp. 16, 17, 19).

[31] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556 (cit. on p. 16).

[32] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV]. URL: https://arxiv.org/abs/1409.0575 (cit. on p. 16).

[33] Pedro F. Felzenszwalb, David McAllester, Deva Ramanan, and Ross B. Girshick. « Object Detection with Discriminatively Trained Part-Based Models ». In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 32.09 (Sept. 2010), pp. 1627–1645. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2009.167. URL: https://doi.ieeecomputersociety.org/10.1109/TPAMI.2009.167 (cit. on p. 21).

[34] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. «The PASCAL Visual Object Classes (VOC) Challenge». In: *International Journal of Computer Vision* 88 (2010), pp. 303–338 (cit. on pp. 22, 36).

[35] Tsung-Yi Lin et al. «Microsoft COCO: Common Objects in Context». In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312 (cit. on pp. 23, 51).

[36] Tom B. Brown et al. «Language models are few-shot learners». In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS '20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546 (cit. on p. 23).

[37] Gaurav Menghani. «Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better». In: *ACM Comput. Surv.* 55.12 (Mar. 2023). ISSN: 0360-0300. DOI: 10.1145/3578938. URL: https://doi.org/10.1145/3578938 (cit. on p. 23).

[38] Tailin Liang, John Glossner, Lei Wang, and Shaobo Shi. «Pruning and Quantization for Deep Neural Network Acceleration: A Survey». In: *CoRR* abs/2101.09671 (2021). arXiv: 2101.09671. URL: https://arxiv.org/abs/2101.09671 (cit. on p. 24).

[39] Babak Rokh, Ali Azarpeyvand, and Alireza Khanteymoori. «A Comprehensive Survey on Model Quantization for Deep Neural Networks in Image Classification». In: *ACM Transactions on Intelligent Systems and Technology* 14.6 (Nov. 2023), pp. 1–50. ISSN: 2157-6912. DOI: 10.1145/3623402. URL: http://dx.doi.org/10.1145/3623402 (cit. on p. 24).

[40] Jianping Gou, Baosheng Yu, Stephen John Maybank, and Dacheng Tao. «Knowledge Distillation: A Survey». In: *CoRR* abs/2006.05525 (2020). arXiv: 2006.05525. URL: https://arxiv.org/abs/2006.05525 (cit. on pp. 24–27).

[41] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML]. URL: https://arxiv.org/abs/1503.02531 (cit. on p. 24).

[42] Gousia Habib, Tausifa jan Saleem, Sheikh Musa Kaleem, Tufail Rouf, and Brejesh Lall. *A Comprehensive Review of Knowledge Distillation in Computer Vision*. 2024. arXiv: 2404.00936 [cs.CV]. URL: https://arxiv.org/abs/2404.00936 (cit. on p. 24).

[43] Amir M. Mansourian et al. *A Comprehensive Survey on Knowledge Distillation*. 2025. arXiv: 2503.12067 [cs.CV]. URL: https://arxiv.org/abs/2503.12067 (cit. on p. 28).

[44] Amir Moslemi, Anna Briskina, Zubeka Dang, and Jason Li. «A survey on knowledge distillation: Recent advancements». In: *Machine Learning with Applications* 18 (2024), p. 100605. ISSN: 2666-8270. DOI: 10.1016/j.mlwa.2024.100605. URL: https://www.sciencedirect.com/science/article/pii/S2666827024000811 (cit. on p. 28).

[45] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. *AI Benchmark: Running Deep Neural Networks on Android Smartphones*. 2018. arXiv: 1810.01109 [cs.AI]. URL: https://arxiv.org/abs/1810.01109 (cit. on p. 28).

[46] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. *Multi-Scale Dense Networks for Resource Efficient Image Classification*. 2018. arXiv: 1703.09844 [cs.LG]. URL: https://arxiv.org/abs/1703.09844 (cit. on p. 29).

[47] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. *BlockDrop: Dynamic Inference Paths in Residual Networks*. 2019. arXiv: 1711.08393 [cs.CV]. URL: https://arxiv.org/abs/1711.08393 (cit. on p. 29).

[48] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. *SkipNet: Learning Dynamic Routing in Convolutional Networks*. 2018. arXiv: 1711.09485 [cs.CV]. URL: https://arxiv.org/abs/1711.09485 (cit. on p. 29).

[49] Jiahui Yu and Thomas Huang. *Universally Slimmable Networks and Improved Training Techniques*. 2019. arXiv: 1903.05134 [cs.CV]. URL: https://arxiv.org/abs/1903.05134 (cit. on pp. 31, 36, 46).

[50] Jiahui Yu and Thomas Huang. *AutoSlim: Towards One-Shot Architecture Search for Channel Numbers*. 2019. arXiv: 1903.11728 [cs.CV]. URL: https://arxiv.org/abs/1903.11728 (cit. on p. 31).

[51] Danna Xue, Fei Yang, Pei Wang, Luis Herranz, Jinqiu Sun, Yu Zhu, and Yanning Zhang. *SlimSeg: Slimmable Semantic Segmentation with Boundary Supervision*. 2023. arXiv: 2207.06242 [cs.CV]. URL: https://arxiv.org/abs/2207.06242 (cit. on p. 34).

[52] Guillermo Iglesias, Edgar Talavera, and Alberto Díaz-Álvarez. «A survey on GANs for computer vision: Recent research, analysis and taxonomy». In: *Computer Science Review* 48 (May 2023), p. 100553. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2023.100553. URL: http://dx.doi.org/10.1016/j.cosrev.2023.100553 (cit. on p. 34).

[53] Liang Hou, Zehuan Yuan, Lei Huang, Huawei Shen, Xueqi Cheng, and Changhu Wang. *Slimmable Generative Adversarial Networks*. 2021. arXiv: 2012.05660 [cs.LG]. URL: https://arxiv.org/abs/2012.05660 (cit. on p. 34).

[54] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. *Dynamic Slimmable Network*. 2021. arXiv: 2103.13258 [cs.CV]. URL: https://arxiv.org/abs/2103.13258 (cit. on pp. 35–37, 43–45, 50, 52, 57).

[55] Jie Hu, Li Shen, and Gang Sun. «Squeeze-and-Excitation Networks». In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7132–7141. DOI: 10.1109/CVPR.2018.00745 (cit. on pp. 36, 60).

[56] Zongxin Yang, Linchao Zhu, Yu Wu, and Yi Yang. *Gated Channel Transformation for Visual Recognition*. 2020. arXiv: 1909.11519 [cs.CV]. URL: https://arxiv.org/abs/1909.11519 (cit. on p. 36).

[57] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax*. 2017. arXiv: 1611.01144 [stat.ML]. URL: https://arxiv.org/abs/1611.01144 (cit. on pp. 36, 44, 50, 57).

[58] Zuoxin Li, Lu Yang, and Fuqiang Zhou. *FSSD: Feature Fusion Single Shot Multibox Detector*. 2024. arXiv: 1712.00960 [cs.CV]. URL: https://arxiv.org/abs/1712.00960 (cit. on p. 36).

[59] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. *The Cityscapes Dataset for Semantic Urban Scene Understanding*. 2016. arXiv: 1604.01685 [cs.CV]. URL: https://arxiv.org/abs/1604.01685 (cit. on p. 37).

[60] Till Beemelmanns. *How to convert Cityscapes dataset to COCO dataset format*. 2020. URL: https://tillbeemelmanns.github.io/2020/10/10/convert-cityscapes-to-coco-dataset-format.html (visited on 09/14/2025) (cit. on p. 38).

[61] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV]. URL: https://arxiv.org/abs/1804.02767 (cit. on p. 38).

[62] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. «Albumentations: Fast and Flexible Image Augmentations». In: *Information* 11.2 (2020). ISSN: 2078-2489. DOI: 10.3390/info11020125. URL: https://www.mdpi.com/2078-2489/11/2/125 (cit. on p. 41).

[63] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. *SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers*. 2021. arXiv: 2105.15203 [cs.CV]. URL: https://arxiv.org/abs/2105.15203 (cit. on p. 41).

[64] Hanchao Li, Pengfei Xiong, Jie An, and Lingxue Wang. *Pyramid Attention Network for Semantic Segmentation*. 2018. arXiv: 1805.10180 [cs.CV]. URL: https://arxiv.org/abs/1805.10180 (cit. on p. 41).

[65] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. «Feature Pyramid Networks for Object Detection». In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106 (cit. on p. 44).

[66] Zhiqiang Shen, Zhankui He, and Xiangyang Xue. «MEAL: Multi-Model Ensemble via Adversarial Learning». In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. ISBN: 978-1-57735-809-1. DOI: 10.1609/aaai.v33i01.33014886. URL: https://doi.org/10.1609/aaai.v33i01.33014886 (cit. on p. 46).

[67] Zhiqiang Shen and Marios Savvides. *MEAL V2: Boosting Vanilla ResNet-50 to 80%+ Top-1 Accuracy on ImageNet without Tricks*. 2021. arXiv: 2009.08453 [cs.CV]. URL: https://arxiv.org/abs/2009.08453 (cit. on p. 46).

[68] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. «Learning efficient object detection models with knowledge distillation». In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 742–751. ISBN: 9781510860964 (cit. on pp. 47, 52).

[69] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. «Mask R-CNN». In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322 (cit. on p. 51).

[70] Antti Tarvainen and Harri Valpola. «Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results». In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1195–1204. ISBN: 9781510860964 (cit. on p. 53).

[71] Yong Xu, Feng Li, Zhile Chen, Jinxiu Liang, and Yuhui Quan. «Encoding spatial distribution of convolutional features for texture representation». In: *Proceedings of the 35th International Conference on Neural Information Processing Systems*. NIPS '21. Red Hook, NY, USA: Curran Associates Inc., 2021. ISBN: 9781713845393 (cit. on p. 60).