Linköping University
Department of Science and Technology
Master's Thesis, 30hp | Applied Mathematics
2026 | LiU-ITN-TEK-A–26/035–SE

# Neural Network Pruning for Efficient Inference

Logan Eriksson

Supervisors: Lukás Malý
Examiner: George Baravdish
External supervisors: Ludvig Gustavsson Roth & Simon Wold

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00, www.liu.se

# Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Logan Eriksson

**Abstract**

Neural network models used for computer vision have achieved remarkable predictive performance in recent years, but these improvements have often come at the cost of increased computational complexity and inference latency. This creates challenges when deploying models on resource-constrained hardware, where efficient inference is critical. Neural network pruning is a model compression technique that aims to reduce computational requirements by removing parameters or structural components from trained networks while preserving predictive performance.

This thesis investigates the effects of neural network pruning on predictive performance and inference latency for object detection models. A modular pruning framework was developed in Python using PyTorch and evaluated on the YOLOv8m and RT-DETR object detection architectures using the VisDrone dataset. The framework supports unstructured pruning, structured pruning, and N:M sparsity pruning together with multiple importance criteria, including magnitude-based pruning, first-order Taylor pruning, and activation-based pruning.

A series of pruning experiments were conducted across multiple pruning ratios. For each configuration, models were pruned, fine-tuned, and evaluated using mAP50:95. Inference latency was measured to assess the practical deployment benefits of different pruning strategies.

The results show that predictive performance generally decreases as pruning ratios increase, although substantial parameter reductions can often be achieved before severe degradation occurs. For both YOLOv8m and RT-DETR, unstructured pruning preserved predictive performance more effectively than structured pruning at comparable pruning levels. Magnitude-based pruning consistently matched or outperformed the more computationally expensive first-order Taylor criterion across most experiments. N:M sparsity pruning retained much of the predictive performance achieved by unstructured pruning while enforcing hardware-oriented sparsity patterns.

The latency experiments revealed a clear distinction between sparsity-based and structure-based pruning approaches. Unstructured pruning and N:M sparsity produced only modest reductions in inference latency despite achieving substantial parameter sparsity. In contrast, structured pruning substantially reduced inference latency by physically removing channels and filters from the network architecture. These findings suggest that parameter sparsity alone is insufficient to guarantee practical runtime acceleration on conventional hardware.

Overall, the results demonstrate that pruning remains an effective approach for improving deployment efficiency in object detection systems, while highlighting the importance of selecting pruning strategies according to deployment objectives and hardware constraints.

# Acknowledgments

I would like to express my sincere gratitude to my supervisor, Lukáš Malý, for his guidance, support, and valuable feedback throughout this thesis. I would also like to thank my examiner, George Baravdish, for his feedback and contributions to the work.

Finally, I would like to express my gratitude to Saab Dynamics for providing me with the opportunity to carry out this thesis project and for supplying the resources necessary for its completion. I would especially like to thank my Saab supervisors, Ludvig Gustavsson Roth and Simon Wold, for their valuable time, guidance, and insightful discussions throughout the project.

# Contents

**Abstract** . . . . . **III**

**Acknowledgments** . . . . . **IV**

**Contents** . . . . . **V**

**1 Introduction** . . . . . **1**
1.1 Background . . . . . 1
1.2 Motivation . . . . . 1
1.3 Aim . . . . . 2
1.4 Proposed Research Questions . . . . . 2
1.5 Scope . . . . . 3
1.6 Related Work . . . . . 3

**2 Theoretical Background** . . . . . **5**
2.1 Object Detection and Classification . . . . . 5
2.2 Convolutional Neural Networks . . . . . 6
2.2.1 Convolution Operation . . . . . 6
2.3 Vision Transformers . . . . . 7
2.3.1 Self-Attention . . . . . 7
2.4 Neural Network Pruning . . . . . 9
2.4.1 Pruning Methods . . . . . 9
2.4.2 Pruning Criteria . . . . . 11
2.5 Evaluation . . . . . 12
2.5.1 Intersection over Union (IoU) . . . . . 13
2.5.2 Terminology . . . . . 13
2.5.3 Precision . . . . . 13
2.5.4 Recall . . . . . 14
2.5.5 Average Precision (AP) . . . . . 14
2.5.6 Mean Average Precision (mAP) . . . . . 14
2.5.7 mAP50:95 . . . . . 14

**3 Method** . . . . . **15**
3.1 Experimental Overview . . . . . 15
3.2 Pruning Framework . . . . . 15
3.2.1 One-Shot Pruning . . . . . 16
3.3 Pruning Schemes . . . . . 17
3.3.1 Unstructured Pruning . . . . . 17
3.3.2 Structured Pruning . . . . . 17
3.3.3 $N$:$M$ Sparsity Pruning . . . . . 18

3.4 Importance Criteria . . . 18
3.4.1 Magnitude Importance . . . 18
3.4.2 First-Order Taylor Importance . . . 18
3.4.3 Activation Importance . . . 19
3.5 Pruning Ratios . . . 19
3.6 Models . . . 19
3.6.1 YOLOv8 . . . 19
3.6.2 RT-DETR . . . 21
3.7 Dataset . . . 23
3.8 Fine-Tuning . . . 27
3.8.1 Fine-Tuning Sparse Models . . . 27
3.8.2 Fine-Tuning Structured Models . . . 28
3.9 Predictive Evaluation . . . 28
3.10 Inference Latency Measurement . . . 28

**4 Results . . . 30**
4.1 YOLOv8m Results . . . 30
4.2 RT-DETR Results . . . 32

**5 Discussion . . . 35**
5.1 YOLOv8m Predictive Performance . . . 35
5.2 YOLOv8m Inference Latency . . . 36
5.3 RT-DETR . . . 37
5.4 Overall Comparison and Implications . . . 38

**6 Conclusions . . . 40**
6.1 Limitations . . . 43
6.2 Further work . . . 44

# 1 Introduction

Deep learning has become a dominant approach for solving complex problems in computer vision [3]. Architectures such as Convolutional Neural Networks (CNNs) and more recently Vision Transformers (ViTs) have achieved remarkable performance in tasks including image classification and object detection [3, 1]. However, these performance improvements have been accompanied by increasingly large and computationally demanding models [1].

The growing size of modern neural networks poses challenges when deploying models in real-world environments, particularly on hardware with limited computational resources. This has motivated a large body of research on model compression and efficiency techniques that aim to reduce the computational cost of neural networks without significantly degrading predictive performance [4].

One of the most widely studied approaches for improving inference efficiency is neural network pruning [5]. Pruning methods remove parameters or structural components from a trained network, producing a smaller and more efficient model. This thesis investigates pruning techniques and their impact on both predictive performance and inference latency for computer vision models.

## 1.1 Background

While modern neural networks have achieved incredible performance across many tasks, this performance has come at the cost of increasingly large and complex models. It is well documented that many modern neural networks are over-parameterized, meaning that many of the networks parameters contribute very little to the models predictive performance [5]. Neural Network Pruning is a collective term for methods that aim to exploit this redundancy by strategically removing structural components of the neural network [8, 5].

## 1.2 Motivation

Efficient inference is a critical requirement when deploying neural networks in practical systems. Many real-world applications need to run models on resource-constrained hardware such as mobile devices, embedded systems, or edge computing platforms. In these environments, limitations in memory, processing power, and energy consumption make it difficult to deploy large neural networks [4].

Model compression techniques provide a potential solution to this problem by reducing the computational and memory requirements of neural networks. Among

these techniques, pruning has gained significant attention due to its ability to reduce model complexity while preserving much of the original predictive performance [5].

Understanding how different pruning strategies affect model accuracy and inference latency is therefore important for enabling efficient deployment of deep learning models.

## 1.3 Aim

The aim of this thesis is to investigate, implement, and evaluate several neural network pruning techniques for computer vision models. The study focuses on analyzing how different pruning strategies affect predictive performance and inference efficiency.

In particular, this work will experimentally evaluate pruning methods on representative Convolutional Neural Network (CNN) and Vision Transformer (ViT) architectures using selected benchmark datasets. The results will be used to analyze the trade-offs between model sparsity, predictive accuracy, and inference latency.

## 1.4 Proposed Research Questions

Proposed research questions:

- How does predictive performance change as the degree of pruning increases for different pruning methods?
- How does inference latency relate to different pruning strategies and pruning degrees?

## 1.5 Scope

This thesis is limited to *Convolutional Neural Networks* (CNN) and *Vision Transformers* (ViT). This thesis will specifically use *YOLOv8* [18] and *RT-DETR* [10] type models on the *VisDrone* [21] dataset. The YOLOv8 architecture comes in several different model sizes of which the medium sized model, YOLOv8m was selected as a representative CNN-based detector due to its widespread use in real-time object detection. RT-DETR was selected as a representative transformer-based detector because it combines transformer architectures with competitive detection performance.

This thesis is limited to exploring the effects of pruning a dense model and comparing to the original model. This thesis will not, however, compare pruned dense models to non-pruned models of similar size.

Iterative pruning will not be investigated in this thesis, but Algorithm 2 shows how this can be done.

This thesis is intended for *Computer Vision* tasks, specifically for *Image Detection and Classification*. This mostly applies to the scope of the theoretical background, as many of the methods described in this work are not task specific.

## 1.6 Related Work

Neural network pruning has been widely studied as a method for reducing model size and computational cost while preserving predictive performance. Early work demonstrated that large neural networks often contain substantial parameter redundancy [5, 8], making it possible to remove parameters with minimal impact on accuracy.

One of the earliest influential approaches is Optimal Brain Damage [8], which proposed removing parameters based on a second-order approximation of the loss function. By estimating the Hessian diagonal, the method evaluates the increase in loss caused by removing each parameter. Optimal Brain Surgeon [6] later extended this idea by incorporating a more accurate second-order analysis using the inverse Hessian. While theoretically well motivated, these approaches are often computationally expensive for modern deep networks.

In practice, simpler criteria have proven highly effective. Magnitude-based pruning removes weights with the smallest absolute values and has become one of the most widely used pruning strategies due to its simplicity and strong empirical performance. Han et al. [5] demonstrated that iterative magnitude pruning combined with retraining can significantly compress deep networks with little loss in

accuracy.

Gradient-based pruning methods attempt to better approximate the effect of removing parameters on the loss function. Molchanov et al. [12] proposed a pruning criterion based on the first-order Taylor expansion of the loss with respect to network parameters. Their method estimates the change in loss when removing a filter or weight, enabling efficient structured pruning in convolutional neural networks. Such Taylor-based approaches form a practical compromise between simple magnitude pruning and computationally expensive second-order methods.

Structured pruning has received considerable attention because it removes entire filters, channels, or layers rather than individual weights, allowing direct reductions in inference latency on modern hardware. Li et al. [9] introduced filter pruning for convolutional networks based on weight magnitudes, while later work, eg. He et al. [7], explored channel pruning and layer pruning strategies to produce hardware-efficient sparse models.

More recently, semi-structured sparsity patterns have been explored to better align pruning with hardware acceleration. N:M sparsity enforces a fixed number of non-zero parameters within small groups of weights, enabling efficient execution on modern accelerators. Mishra et al. [11] and related work demonstrated that structured patterns such as 2:4 sparsity can provide significant speedups on specialized hardware while maintaining competitive accuracy.

Transformer pruning methods often target attention heads, tokens, or hidden dimensions to reduce computational complexity. For example, Zhu et al. [20] proposed pruning strategies specifically designed for transformer architectures.

Overall, existing pruning approaches can be broadly categorized according to their importance criteria (e.g., magnitude-based, first-order, second-order, or activation-based methods) and their sparsity structure (unstructured, structured, or semi-structured). These categories directly motivate the theoretical framework for pruning criteria introduced in Section 2.4.

# 2 Theoretical Background

This section presents the theoretical background required for the thesis. First, object detection and image classification are introduced as computer vision tasks. This is followed by a description of Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), which represent two common families of neural network architectures for vision tasks. Finally, the section introduces neural network pruning, including the pruning structures and importance criteria used in this work.

## 2.1 Object Detection and Classification

Computer vision models are commonly used for object detection and classification tasks. In image classification, the objective is to assign a class label to an image from some set of possible object classes. Object detection extends this task by additionally estimating the spatial locations of individual objects within the image [14]. Given an input image

$$X \in \mathbb{R}^{H \times W \times C},$$

where $H$ and $W$ denote image height and width (in pixels) and $C$ denotes the number of input channels, an object detector produces a set of candidate detections. For RGB images, which is by far the most commonly used input channel dimension, $C = 3$ is used.

A common representation of the detector output is

$$Y \in \mathbb{R}^{N \times (4+K)},$$

where $K$ is the number of object classes and $N$ is the number of candidate detections predicted by the model before post-processing. For each candidate detection, the model predicts

$$(x, y, w, h, p_1, \ldots, p_K),$$

where $(x, y)$ are the bounding box center coordinates, $(w, h)$ are the bounding box width and height and $(p_1, \ldots, p_K)$ are class probabilities or class confidence values.

The value of $N$ should be interpreted as the number of candidate predictions produced by the model architecture before post-processing. In many modern object detectors this number is fixed by the architecture and input resolution, while the final number of reported detections can vary between images due to post-processing steps like confidence thresholding and non-maximum suppression [14].

The class with the highest confidence score is then selected for each candidate bounding box. Candidate boxes with confidence below a chosen threshold are

typically discarded. In addition, object detectors typically apply non-maximum suppression or a related post-processing method to remove duplicate or heavily overlapping detections [14].

## 2.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a neural network which uses convolutional kernels, often refered to as filters, to extract information from the input. A neural network used for object detection and classification as described above, can be viewed as a parameterized function

$$f_\theta : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{N \times (4+K)},$$

where $\theta$ denotes the learnable parameters of the network. For CNNs, many of these parameters are kernel or filter weights.

The notation above uses the image convention $H \times W \times C$ for readability. In many deep learning frameworks, including PyTorch, tensors are internally represented in channel-first format $C \times H \times W$ for individual images and $B \times C \times H \times W$ for batches, where $B$ denotes the batch size [13].

### 2.2.1 Convolution Operation

Let $x \in \mathbb{R}^{H \times W \times C}$ denote an input feature map. A convolutional layer with $C'$ output channels applies a set of kernels

$$W = \{W^{(k)} \in \mathbb{R}^{C \times r \times r}\}_{k=1}^{C'}$$

and biases $\{b^{(k)}\}_{k=1}^{C'}$. The $k$-th output feature map can be written as

$$z^{(k)} = \sigma\left(W^{(k)} * x + b^{(k)}\right) \in \mathbb{R}^{H' \times W'},$$

where $*$ denotes the convolution operation as commonly implemented in CNNs (technically cross-correlation), $b^{(k)} \in \mathbb{R}$ is a bias term, and $\sigma(\cdot)$ is a nonlinear activation function such as ReLU (Rectified Linear Unit) or SiLU (Sigmoid Linear Unit). The spatial dimensions $H'$ and $W'$ of the output $z^{(k)}$ depend on implementation choices such as kernel size, stride, and padding [3]. The output of the entire convolutional layer is $z \in \mathbb{R}^{H' \times W' \times C'}$.

Stacking convolutional layers produces a hierarchical feature representation. Early layers tend to capture local low-level patterns such as edges and textures, while deeper layers capture increasingly abstract semantic features [3]. This property makes CNNs well suited to image recognition and object detection tasks. The

layered structure of CNNs is also particularly relevant for pruning, since filters and channels form natural structural units that can be removed during structured pruning.

## 2.3 Vision Transformers

Vision Transformers (ViTs) adapt the transformer architecture to image data. Unlike CNNs, which use convolutional operations with strong locality assumptions, ViTs process images as sequences of patches and use self-attention mechanisms to model relationships between image regions [1]. The locality assumption in CNNs arises from the use of convolutional kernels with limited receptive fields. Each convolution operates only on a small spatial neighborhood, meaning that information from distant regions of an image can interact only after passing through multiple layers. While convolutional layers primarily operate on local neighborhoods, self-attention allows information to be exchanged directly between all image patches simultaneously.

An input image

$$X \in \mathbb{R}^{H \times W \times C}$$

is divided into $N$ non-overlapping patches of size $P \times P$. Patch sizes are typically fixed, meaning the number of patches depends on the input resolution. Each patch is flattened into a vector

$$x_i \in \mathbb{R}^{P^2 C}$$

and linearly projected into a $D$-dimensional embedding vector

$$e_i = W_E x_i + b,$$

where $W_E$ and $b$ contain learnable parameters. The specific ordering of pixel values from the patch to the flattened vector is arbitrary, as long as it is consistent. The resulting embedding vectors can be collected into a matrix

$$E \in \mathbb{R}^{N \times D}.$$

Positional information is typically added to the patch embeddings so that the model can distinguish between patches at different spatial locations [1].

### 2.3.1 Self-Attention

Self-attention constructs three learned representations for each embedding vector: queries, keys, and values. Queries determine what information an embedding

vector seeks, keys determine what information an embedding vector contains, and values represent the information that may be aggregated from other embedding vectors. Given embeddings

$$E \in \mathbb{R}^{N \times D},$$

a self-attention layer computes query, key, and value matrices

$$Q = EW_Q, \qquad K = EW_K, \qquad V = EW_V,$$

where

$$W_Q, W_K, W_V \in \mathbb{R}^{D \times d_{head}}$$

are learnable projection matrices and $d_{head}$ denotes the fixed dimension of the projected attention representations. This gives

$$Q, K, V \in \mathbb{R}^{N \times d_{head}}.$$

Scaled attention is defined as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_{head}}}\right)V,$$

where

$$\text{softmax}(Z)_{ij} = \frac{\exp(Z_{ij})}{\sum_{k=1}^{n} Z_{ik}}, \quad Z \in \mathbb{R}^{m \times n}$$

Softmax is applied row-wise, such that each row forms a probability distribution over all keys for a given query. The matrix

$$QK^\top$$

has dimensions $N \times N$ and produces pairwise attention scores that quantify how strongly different embedding vectors should interact. The softmax operation normalizes these scores into attention weights used to combine information from the value representations. The scaling factor $\sqrt{d_{head}}$ prevents large dot-product values from producing excessively peaked softmax distributions, improving training stability [19].

The above describes a single attention head. Multi-head attention performs several attention operations in parallel and concatenates the resulting outputs before applying a final linear projection. One attention block may have $h$ different projection matrices

$$W_Q^{(i)}, W_K^{(i)}, W_V^{(i)}$$

giving

$$A = [A_1, \ldots, A_h],$$

where

$$A_i = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_{head}}}\right) V_i, \quad i = 1, \ldots, h.$$

In addition to self-attention, transformer blocks typically include layer normalization, residual connections, and feed-forward networks. A natural pruning target for attention blocks is the number of attention heads $h$.

## 2.4 Neural Network Pruning

Neural network pruning aims to reduce the size or computational cost of a trained model while preserving as much predictive performance as possible. Let $\theta \in \mathbb{R}^d$ denote the full parameter vector of a neural network and let $\mathcal{L}(\theta)$ denote the training loss. Pruning can be represented by introducing a binary keep-mask

$$m \in \{0, 1\}^d,$$

where $m_i = 1$ means that parameter $\theta_i$ is retained and $m_i = 0$ means that it is pruned. The pruned parameter vector is then

$$\tilde{\theta} = m \odot \theta,$$

where $\odot$ denotes element-wise multiplication.

An ideal pruning objective can be written as

$$\arg\min_{m \in \{0,1\}^d} \mathcal{L}(m \odot \theta) \quad \text{s.t.} \quad \|m\|_0 \leq B,$$

where $B$ is the maximum number of parameters allowed to remain non-zero. Equivalently, if the goal is to prune exactly $k$ parameters, the constraint can be written as

$$\|m\|_0 = d - k.$$

Directly solving this optimization problem is generally infeasible for modern neural networks because evaluating many possible masks would require repeated training or fine-tuning. In practice, pruning methods therefore use an importance score $\mathcal{S}(\theta)$ to estimate which parameters or structures are least important and should be removed [8, 12].

### 2.4.1 Pruning Methods

A pruning method defines how parameters are allowed to be pruned. The pruning methods that will be used are *unstructured*, *structured*, and *N:M sparsity*, and are described in this section.

**Unstructured Pruning** Unstructured pruning removes individual parameters independently. Unstructured pruning can be formulated as

$$\arg\min_{m \in \{0,1\}^d} \mathcal{S}(m \odot \theta) \quad \text{s.t.} \quad \|m\|_0 = d - k,$$

where $\mathcal{S}$ is some metric and $k$ is the number of parameters we wish to remove.

**Structured Pruning** Structured pruning removes entire parameter groups such as convolutional filters, output channels, neurons, or attention dimensions. Let

$$\{g_0, g_1, \ldots, g_G\}$$

be non-overlapping groups of parameter indices, where each group $g_j \subset \{1, \ldots, d\}$ corresponds to a prunable structure, and $g_0$ corresponds to the remaining non-prunable parameters. A binary group keep-mask

$$z \in \{0, 1\}^G$$

can be used to define the full parameter mask as

$$m(z)_i = \begin{cases} z_j & \text{if } i \in g_j \text{ and } j = 1, \ldots, G \\ 1 & \text{if } i \in g_0 \end{cases}, \quad i = 1, \ldots, d$$

Here $z_j = 1$ means that group $g_j$ is retained and $z_j = 0$ means that it is pruned. Structured pruning can then be formulated as

$$\arg\min_{z \in \{0,1\}^G} \mathcal{S}(m(z) \odot \theta) \quad \text{s.t.} \quad \|z\|_0 = G - k,$$

where $k$ in this case is the number of groups we wish to remove.

Structured pruning changes the architecture of the network by removing complete structures. This can reduce the number of floating-point operations and can improve inference latency on ordinary dense hardware, because the resulting model has smaller dense layers rather than sparse tensors of the original size.

$N$:$M$ **Sparsity Pruning** $N$:$M$ sparsity pruning is a semi-structured pruning method where weights are zeroed out such that at most $N$ out of every $M$ parameters in a weight matrix $W$ remain non-zero. For simplicity, assume $M$ divides the number of columns in $W$. Let

$$\{\theta^{(1)}, \ldots, \theta^{(G)}\}$$

be a partition of of the parameters in $W$ such that each group contains $M$ consecutive weights in $W$. $N$:$M$ sparsity pruning can be formulated as

$$\underset{m_i \in \{0,1\}^M}{\arg\min} \mathcal{S}(m_i \odot \theta^{(i)}) \quad \text{s.t.} \quad \|m_i\|_0 = M - N, \quad i = 1, \ldots, G.$$

The values retained are those with the highest importance scores within the group. For example, in 2:4 sparsity, two weights are retained and two weights are pruned in each group of four.

This pruning structure is less flexible than unstructured pruning, but more hardware-friendly. It is designed to align with specialized acceleration support for semi-structured sparse matrices.

### 2.4.2 Pruning Criteria

A pruning criterion defines how parameter importance is estimated. This thesis uses *magnitude importance*, *first order importance* and *activation importance*, and are described in this section.

**Magnitude-Based Pruning** Magnitude-based pruning ranks parameters or parameter groups according to their absolute values. Magnitude-based pruning assumes that parameters with smaller magnitudes have less influence on model behavior and can therefore be removed with lower expected impact.

For an individual parameter $\theta_i$, the importance score is

$$\mathcal{S}(\theta_i) = |\theta_i|.$$

For a group of parameters $\theta_g$, such as a convolutional filter, a group score can be computed using a norm:

$$\mathcal{S}(\theta_g) = \|\theta_g\|_p.$$

Parameters or groups with the smallest scores are pruned first. In practice, $p \in \{1, 2\}$ are common choices.

**First-Order Taylor Pruning** First-order Taylor pruning estimates how much the loss would change if a parameter or group of parameters were removed. Using a first-order Taylor approximation,

$$\mathcal{L}(\theta + \delta\theta) \approx \mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^\top \delta\theta.$$

denote the gradient of the loss with respect to the parameters. If parameter $\theta_i$ is removed, then $\delta\theta_i = -\theta_i$. The approximate change in loss for that parameter is

$$\Delta\mathcal{L}_i \approx -\nabla\mathcal{L}(\theta)_i\theta_i.$$

Just like the magnitude criteria, we prune the weights with the least expected impact on the model. This naturally gives the importance scores

$$\mathcal{S}(\theta_i) = |\Delta\mathcal{L}_i| = |\nabla\mathcal{L}(\theta)_i\theta_i|.$$

For a parameter group $\theta_g$ with indices $I_g$, the scores of individual parameters can be aggregated, for example by summation:

$$\mathcal{S}(\theta_g) = (\sum_{i\in I_g} |\nabla\mathcal{L}(\theta)_i\theta_i|^p)^{\frac{1}{p}}.$$

**Activation-Based Pruning** Activation-based pruning estimates the importance of a structure from the activations it produces when data is passed through the network. The intuition is that channels or neurons with consistently small activations contribute less to later computations and are therefore candidates for pruning.

Let $a_j(x)$ denote the output activation tensor produced by a candidate structure $j$ when processing input $x$. Depending on the pruning target, $a_j(x)$ may represent, for example, the feature map of a convolutional channel or the output of an attention head. Let $\mathcal{D}_{\text{cal}}$ denote a calibration data batch. An activation-based importance score can be written as

$$\mathcal{S}_j = \frac{1}{|\mathcal{D}_{\text{cal}}|} \sum_{x\in\mathcal{D}_{\text{cal}}} \|a_j(x)\|_p,$$

Structures with the smallest activation scores are pruned first.

Activation-based pruning is naturally suited to structured pruning because activations are associated with output channels, features, or neurons rather than individual scalar weights. In the implemented framework, activation-based pruning is therefore used strictly as a structured pruning criterion.

## 2.5 Evaluation

To produce and analyze results in a meaningful way, we need well defined evaluation metrics. This section aims to define the metrics that will be used.

### 2.5.1 Intersection over Union (IoU)

To evaluate the spatial accuracy of predicted bounding boxes, the *Intersection over Union* (IoU) metric is used. Given a ground truth bounding box $B_{gt}$ and a predicted bounding box $B_p$, the *Intersection over Union* (IoU) is an accuracy measure of the predicted bounding box and is given by

$$\text{IoU}(B_{gt}, B_p) = \frac{A(B_{gt} \cap B_p)}{A(B_{gt} \cup B_p)},$$

where $A(B)$ is the area of a set $B$. IoU $\in [0, 1]$ with higher values indicating better prediction.

### 2.5.2 Terminology

Model predictions are categorized according to the following definitions:

- **True Positives (TP):** Objects correctly detected and classified by the model.
- **False Positives (FP):** Incorrect detections where the model predicts an object that is not present, or incorrectly classifies an object.
- **True Negatives (TN):** Correct rejections where no object is predicted and none is present. Note that true negatives are not practically quantifiable as there is no meaningful way to count the number of objects not present in an image.
- **False Negatives (FN):** Missed detections where an object is present but not identified by the model.

Here, a correct detection is one where the predicted bounding box meets a sufficient IoU threshold with a ground truth box.

### 2.5.3 Precision

*Precision* ($P$) measures the proportion of predicted objects that are correctly identified and is given by

$$P = \frac{TP}{TP + FP}.$$

The metric answers the question:
*"Of all the detected objects, how many were actually present?"*

### 2.5.4 Recall

*Recall* ($R$) measures the proportion of present objects that are correctly identified and is given by

$$R = \frac{TP}{TP + FN}$$

The metric answers the question:
*"Of all the present objects, how many were actually detected?"*

### 2.5.5 Average Precision (AP)

The average precision also combines precision and recall, but does so at many different confidence levels. Assume we have $N_i$ predictions of class $i$, sorted in descending order by confidence scores. Let a prediction be a true positive (TP), only if it is correctly identified and has an IoU greater than some threshold $\tau$. Let $P_k(\tau)$ and $R_k(\tau)$ be the precision and recall scores when considering the top $k$ predictions by confidence. The average precision, for that class, at that threshold is

$$\mathrm{AP}_i(\tau) = \sum_{n=1}^{N_i} (R_n(\tau) - R_{n-1}(\tau)) P_n(\tau),$$

with $R_0(\tau) = 0$.

### 2.5.6 Mean Average Precision (mAP)

The mean average precision is just the AP averaged over all classes in the data set and is given by

$$\mathrm{mAP}(\tau) = \frac{1}{C} \sum_{i=1}^{C} \mathrm{AP}_i(\tau),$$

where $C$ is the number of classes.

### 2.5.7 mAP50:95

mAP50:95 averages the mAP over different IoU thresholds. Let

$$\mathcal{T} = \{0.5, 0.55, \ldots, 0.9, 0.95\}.$$

mAP50:95 is given by

$$\text{mAP50:95} = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \mathrm{mAP}(\tau).$$

# 3 Method

This chapter describes the experimental methodology used to investigate the effects of neural network pruning on object detection models. The study focuses on how different pruning strategies influence predictive performance, model sparsity, and inference efficiency. To ensure fair comparison, all pruning configurations are evaluated using the same baseline model, dataset, image size, fine-tuning settings, and evaluation procedure.

## 3.1 Experimental Overview

A pruning configuration consists mainly of the pruning method, the pruning criteria or importance score, and the pruning ratio. The pruning ratio is the ratio of parameters that are removed. For example a pruning ratio of 0.1 removes 10% of the parameters. Each pruning configuration is evaluated independently. For every pruning ratio, a new copy of the same dense baseline checkpoint is loaded before pruning. This prevents information from previous pruning runs from influencing later experiments. The experimental pipeline follows the steps shown in Figure 1.

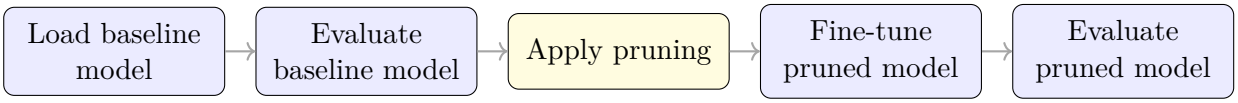



Figure 1: High-level overview of the experimental pipeline.

## 3.2 Pruning Framework

A modular pruning framework was implemented in Python using PyTorch [13]. The pruning process is controlled through a configuration object that specifies the pruning criterion, pruning scheme, pruning ratio, device, calibration settings, and module exclusion rules. A pruning engine then applies the selected importance criterion together with the selected pruning scheme. Algorithm 1 shows a high-level overview of how the framework is used.

The framework uses a configurable module selector to determine which model components are eligible for pruning. By default, supported weight-bearing modules include convolutional and linear layers, while normalization layers and detection heads are excluded. For structured pruning, Torch-Pruning [2] is used to propagate architectural changes through dependent modules.

**Algorithm 1** High-level pruning workflow using the framework

Create pruning configuration
    $cfg \leftarrow$ `PruningConfig`($scheme$, $criterion$, $ratio$)
Initialize pruning engine
    $engine \leftarrow$ `PruningEngine`($model$, $cfg$)
**if** *criterion* is `"taylor"` or `"activation"` **then**
    Build *dataloader*
    Build *batch_adapter*
    **if** *criterion* is `"taylor"` **then**
        Build *loss_adapter*
    **end if**
**end if**
**if** *scheme* is `"structured"` **then**
    Create example input tensor *example_input*
**end if**
Apply pruning
    *engine*.`prune`(*dataloader*, *batch_adapter*, *loss_adapter*, *example_input*)

### 3.2.1 One-Shot Pruning

This thesis uses one-shot pruning for all experiments. In one-shot pruning, the target pruning ratio is applied in a single step, often followed by fine-tuning. This differs from iterative pruning, where pruning is done in small steps with intermediate fine-tuning.

One-shot pruning was selected because it gives a controlled comparison between pruning configurations. Each experiment begins from the same dense baseline checkpoint and applies a single pruning ratio. This makes it easier to compare how pruning scheme, importance criterion, and pruning ratio affect the final model.

It is of course possible to use iterative pruning with the framework by repeated one-shot pruning and fine-tuning. Algorithm 2 shows how this can be done.

**Algorithm 2** Iterative pruning example

```
total_pruning_ratio ← 0
while total_pruning_ratio < target_ratio do
    Set Δr
    Apply Algorithm 1
     using pruning ratio Δr
    Fine-tune the pruned model
    Update total_pruning_ratio
end while
```

## 3.3 Pruning Schemes

The framework supports three pruning schemes: unstructured pruning, structured pruning, and $N$:$M$ sparsity pruning.

### 3.3.1 Unstructured Pruning

Unstructured pruning removes individual scalar weights. In the implementation, binary masks are applied to weight tensors. The original tensor shapes are preserved, but selected weights are set to zero.

For unstructured pruning, both per-module and global pruning are supported by the framework. In the experiments, pruning is performed locally per module. This means that the same pruning ratio is applied independently to each selected layer. Because unstructured pruning keeps the dense layer shapes unchanged, it mainly reduces the number of non-zero parameters.

### 3.3.2 Structured Pruning

Structured pruning removes entire output channels or features from supported layers. For convolutional layers, this corresponds to pruning output channels or filters. For linear layers, this corresponds to pruning output features or neurons.

Structured pruning changes the architecture of the model. This directly reduces the actual number of model parameters.

The implementation uses the Torch-Pruning library for dependency-aware structured pruning [2]. Dependency-aware pruning is necessary because removing an output channel from one layer can affect the expected input channels of later layers. Torch-Pruning traces these dependencies and applies compatible structural

changes across connected layers.

### 3.3.3 $N$:$M$ Sparsity Pruning

$N$:$M$ sparsity is implemented as a semi-structured mask-based pruning scheme. The framework reshapes supported weight tensors into a matrix view and applies the $N$:$M$ constraint row-wise. For each group of $M$ consecutive weights, only the $N$ weights with the highest importance scores are retained.

In the current implementation, incomplete trailing groups that do not contain exactly $M$ elements are left unchanged. This avoids introducing irregular behavior at tensor boundaries. The default configuration uses 2:4 sparsity.

## 3.4 Importance Criteria

The framework supports three importance criteria: magnitude-based importance, first-order Taylor importance, and activation-based importance.

### 3.4.1 Magnitude Importance

Magnitude pruning uses the absolute value of each weight as its importance score. For unstructured and $N$:$M$ pruning, this score is computed element-wise. For structured pruning, channel or filter importance is computed from grouped weight magnitudes.

### 3.4.2 First-Order Taylor Importance

Taylor pruning estimates the effect of removing weights using gradients of the loss function. The implementation accumulates gradients over calibration inputs and computes element-wise scores. For the results in this thesis, one batch of size 16 is used to calculate the gradients.

For structured pruning, element-wise Taylor scores are aggregated over candidate output channels or features. Taylor pruning requires both calibration data and a task-specific loss function. In this work, an Ultralytics-compatible detection loss adapter is used to compute the loss during calibration [18].

### 3.4.3 Activation Importance

Activation-based pruning estimates the importance of structures using activation statistics collected during forward passes. The implementation registers forward hooks on prunable modules and records activation magnitudes during calibration inputs. For the results in this thesis, a batch of size 16 is used to record the activations. Activation pruning is used only for structured pruning, since activations naturally correspond to channels or features rather than individual scalar weights.

## 3.5 Pruning Ratios

The pruning ratio provided to the framework acts only as a target ratio for the chosen modules. The actual pruning ratio may differ from the target ratio. For unstructured pruning, the actual ratio will often be lower than the target due to the fact that not all modules are prunable. For $N : M$ pruning, the ratio is fixed at $\frac{N}{M}$ per prunable module meaning the actual ratio will often be slightly lower than the target ratio. For structured pruning, dependent modules are also affected meaning the actual ratio can be higher than the target ratio.

## 3.6 Models

The experiments are conducted using the YOLOv8 model and the RT-DETR model provided by Ultralytics [18, 10]. These models act as representative models for CNN and ViT type architectures.

### 3.6.1 YOLOv8

YOLOv8 is a convolutional object detector designed for real-time detection.

The model used in the experiments is YOLOv8m. The architecture is illustrated in Figure 2. The model contains convolutional blocks, C2f modules, an SPPF module, feature aggregation layers, and a detection head. For unstructured pruning, all weights in Conv2d are targets for pruning. This includes internal Conv2d modules in Conv, C2f, SPPF and Bottleneck modules. For structured pruning, the prunable groups are the filters or output channels of the Conv2d modules. The first number in the Conv modules in Figure 2 corresponds to the number of output channels. Looking back to the mathematical formulation in Section 2.2.1, structured pruning for convolutional blocks is done by reducing the output dimension $C'$. The detection head also contains convolutional blocks, but is

not modified by pruning in the experiments in this thesis. As seen in Section 2.2.1, the kernel or filter size must match the input channel dimension $C$ for the convolution to be well defined. The output channel dimension from one convolutional block is often the input channel dimension of a later block, as seen in Figure 2. Reducing the number of output channels of a convolutional block can therefore also force a reduction in later convolutional filters channel dimension.

**Example** As an example, consider the first two Conv modules in the figure. The first block produces an output with channel dimension 48, meaning the filters in the second Conv module are of size $48 \times 3 \times 3$. Now imagine removing 24 of the filters in the first Conv. This not only removes the weights from these filters, but also removes the weights in the 24 corresponding "slices" of each filter in the second Conv. Note that the number of output channels in the second Conv is not affected by this change, only the dimensions of each individual filter.

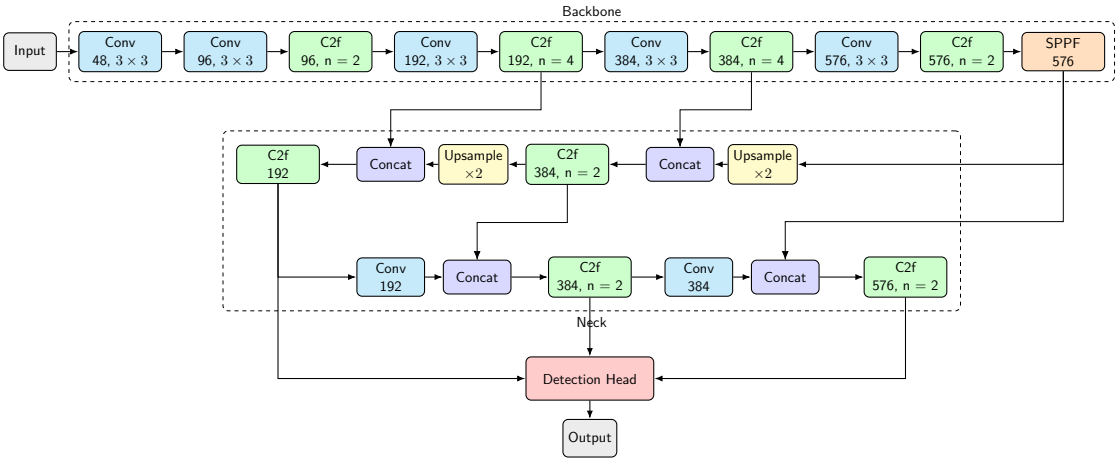



Figure 2: Ultralytics YOLOv8m detection architecture [18].

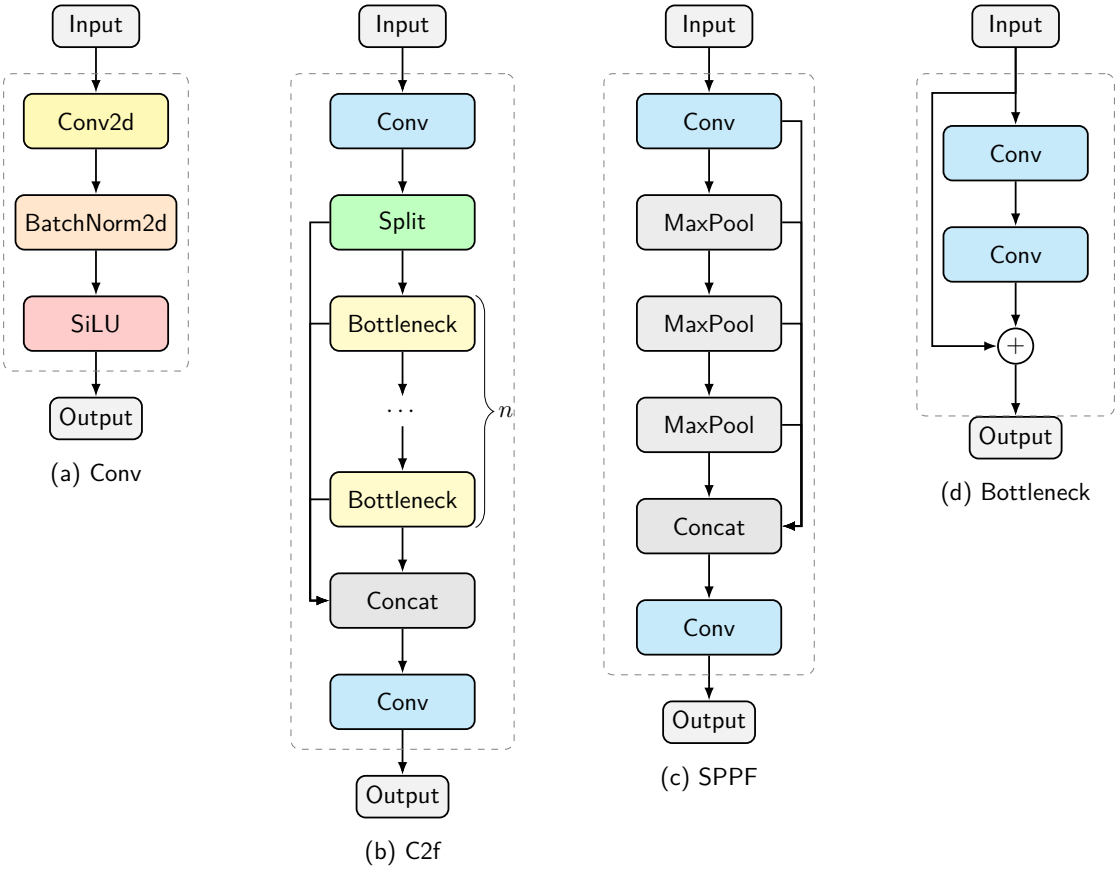



Figure 3: Internal structures of YOLOv8 modules [16].

### 3.6.2 RT-DETR

For RT-DETR, structured pruning targets the linear projection layers used in attention and feed-forward modules. Looking back to the formulation in Section 2.3.1, multi-head attention consists of several parallel attention heads with projection matrices $W_Q^{(i)}, W_K^{(i)}$, and $W_V^{(i)}$. Structured pruning can therefore remove complete attention heads or reduce hidden dimensions by removing corresponding projection parameters. Similar to channel pruning in CNNs, removing attention structures may propagate architectural changes to connected layers.

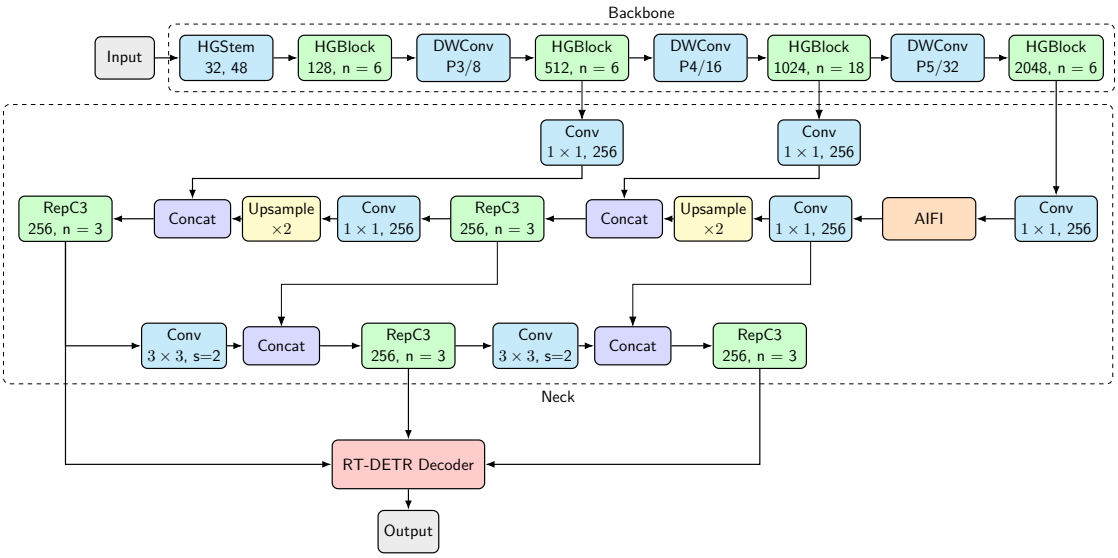



Figure 4: Ultralytics RT-DETR-L detection architecture [15].

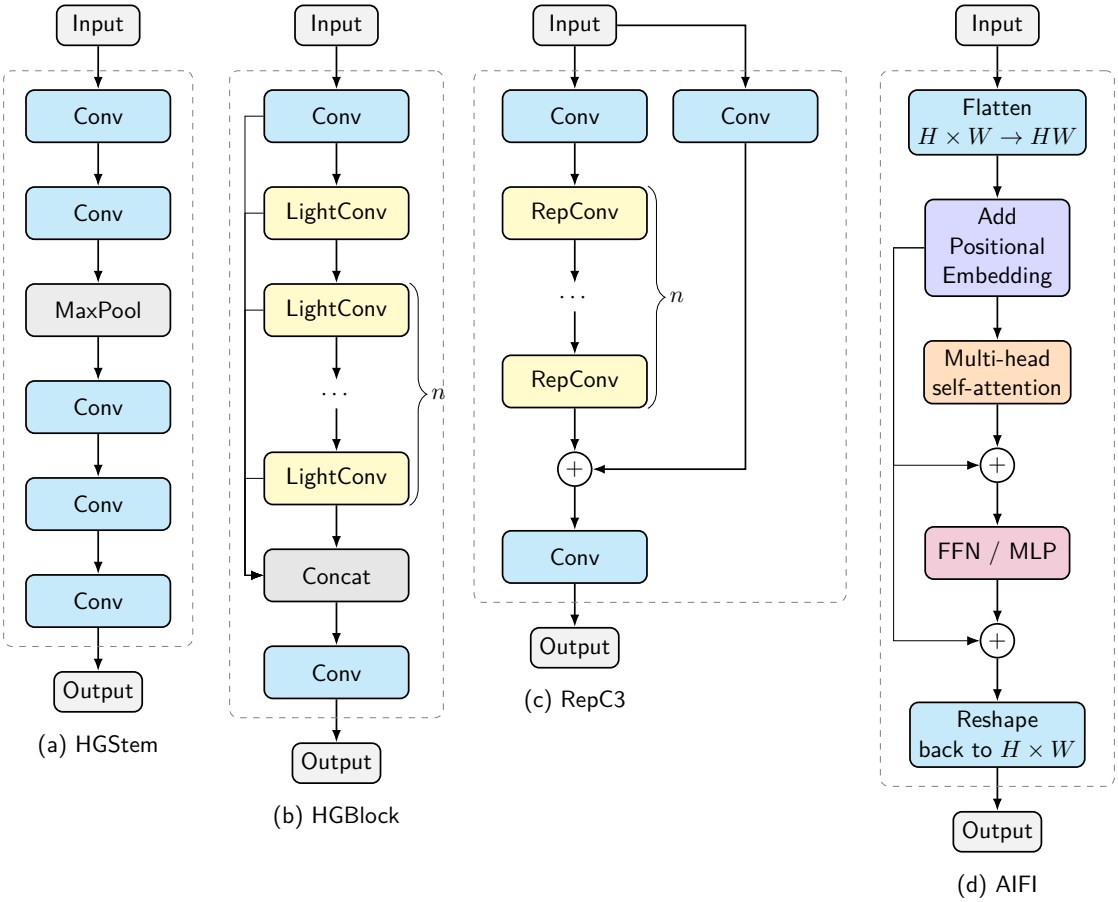



Figure 5: Internal structures of common RT-DETR modules [16, 17].

## 3.7 Dataset

The experiments are conducted using the VisDrone [21] dataset, a benchmark dataset for object detection in drone imagery. The dataset contains aerial images with annotated objects such as pedestrians, vehicles and bicycles. Table 1 shows all the classes and their class-id. Drone imagery is challenging because objects often appear at small scales, from varying viewpoints, and in crowded scenes.

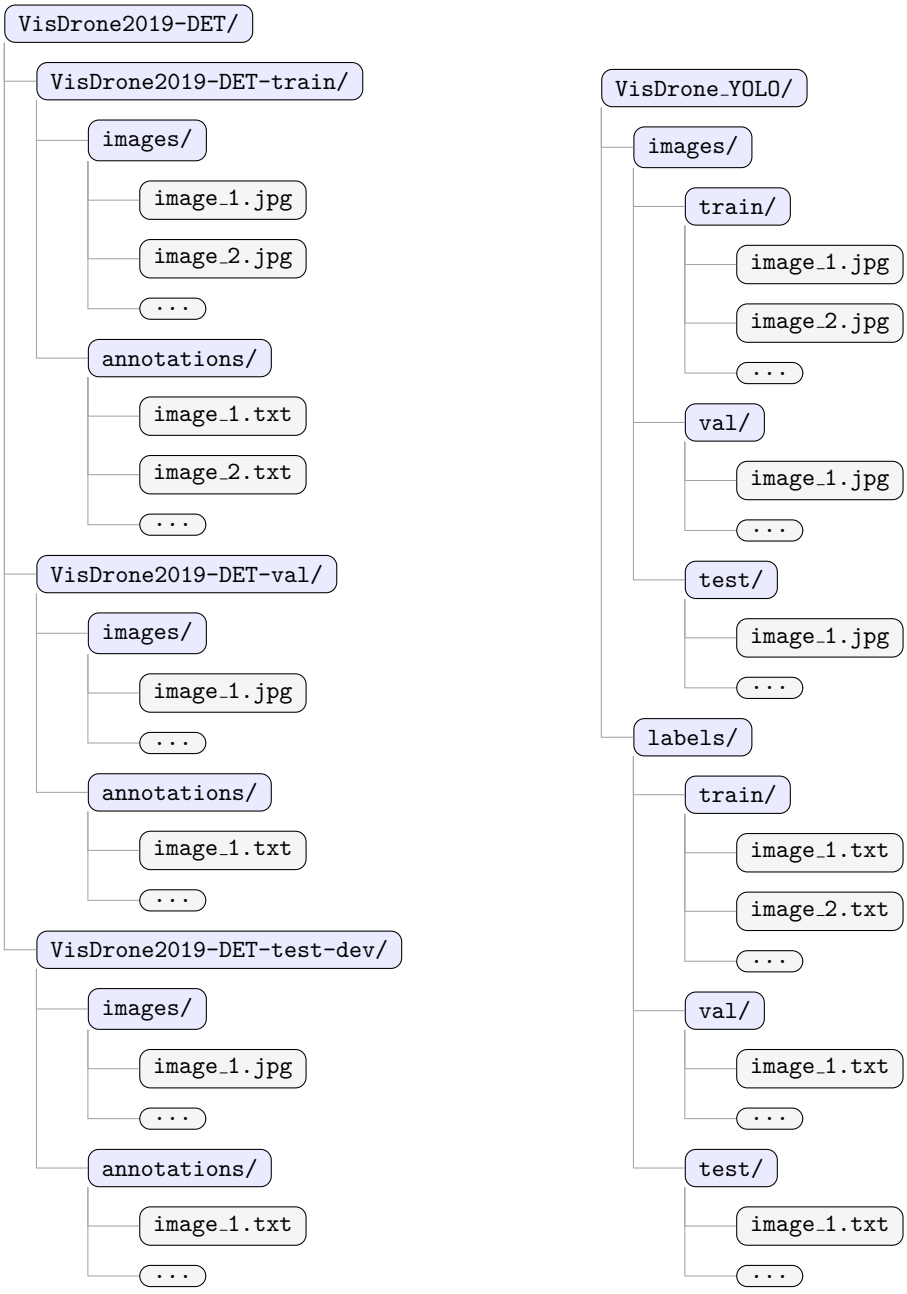
Table 1: VisDrone Object Classes

| **Class ID** | **Class Name** |
|---|---|
| 0 | ignored regions |
| 1 | pedestrian |
| 2 | people |
| 3 | bicycle |
| 4 | car |
| 5 | van |
| 6 | truck |
| 7 | tricycle |
| 8 | awning-tricycle |
| 9 | bus |
| 10 | motor |

Table 2: VisDrone Classes in YOLO Format

| **YOLO ID** | **Class Name** |
|---|---|
| 0 | pedestrian |
| 1 | people |
| 2 | bicycle |
| 3 | car |
| 4 | van |
| 5 | truck |
| 6 | tricycle |
| 7 | awning-tricycle |
| 8 | bus |
| 9 | motor |

The dataset is converted to YOLO format. In the YOLO format, images and labels are stored in separate directories for training, validation, and testing. Each annotation file contains one row per object. The formatting is included because this specific file structure is expected in the Ultralytics training setup.




(a) Original VisDrone dataset structure (b) YOLO-formatted dataset structure

Figure 6: Comparison between the original VisDrone dataset and the YOLO dataset format.

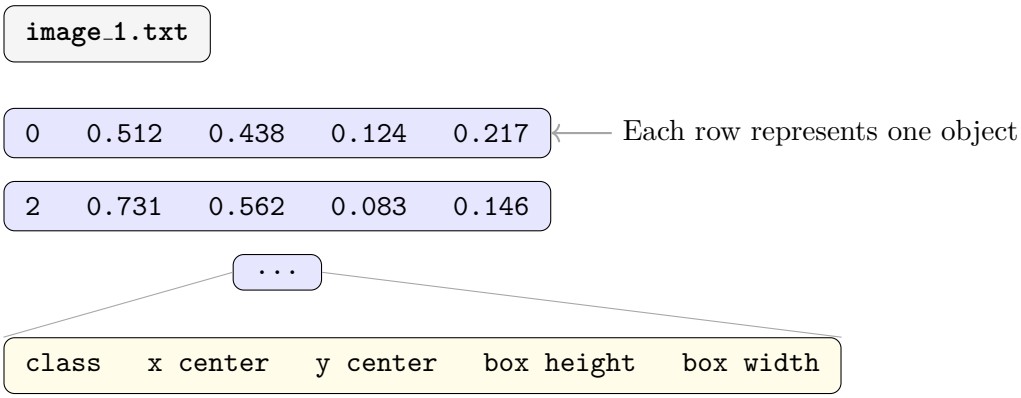
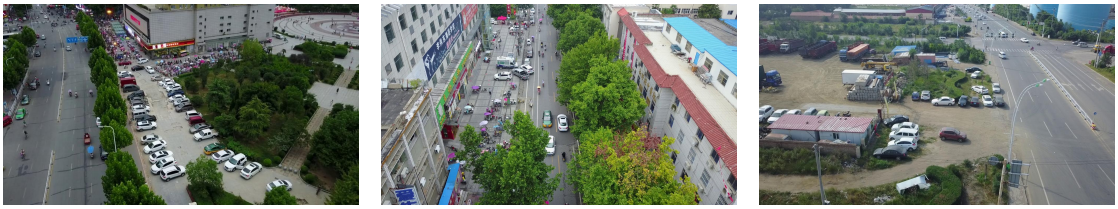



Figure 7: Internal structure of a YOLO annotation text file

The coordinates are normalized relative to the image width and height.

The training split is used for fine-tuning pruned models. The test split is used for final evaluation. All images are resized to $640 \times 640$ pixels during both fine-tuning and evaluation.

The original and converted dataset structures are shown in Figure 6, and examples of non-annotated and annotated images are shown in Figure 8 and Figure 9.



(a) Train (b) Validation (c) Test

Figure 8: Example non-annotated images from each dataset split

(a) Train (b) Validation (c) Test

Figure 9: Annotated example images from each dataset split

## 3.8 Fine-Tuning

After pruning, each model is fine-tuned to recover predictive performance. Fine-tuning is performed using the Ultralytics training pipeline with the same dataset, image size, batch size, and deterministic settings across pruning configurations. Table 3 summarizes the parameters used for fine-tuning.

Table 3: Fine-Tuning settings

| Parameter | Value |
|---|---|
| Dataset | VisDrone [21] |
| Image size | 640×640 |
| Training Batch size | 16 |
| Maximum Fine-tuning epochs | 150 |
| Early topping Patience | 10 epochs |
| Random seed | 2026 |

Data augmentation is disabled during fine-tuning in order to reduce uncontrolled variation between pruning experiments.

### 3.8.1 Fine-Tuning Sparse Models

For mask-based pruning methods such as unstructured pruning and $N$:$M$ sparsity, sparse masks must be preserved during fine-tuning. Otherwise, pruned weights can become non-zero again due to gradient updates or optimizer state. To prevent this, the implementation applies three safeguards during sparse fine-tuning:

1. gradients corresponding to pruned weights are masked before optimizer updates,
2. optimizer state tensors corresponding to pruned weights are also masked,
3. pruning masks are re-applied after optimizer steps.

This ensures that pruned weights remain zero throughout fine-tuning. After fine-tuning, masks are finalized so that the exported model contains the effective sparse weights.

### 3.8.2 Fine-Tuning Structured Models

For structured pruning, the model architecture is physically modified before fine-tuning. Since removed channels or filters no longer exist in the model, no sparse mask enforcement is required. The structurally pruned model is fine-tuned directly using the same Ultralytics training procedure.

## 3.9 Predictive Evaluation

Predictive performance is evaluated on the test split using standard object detection metrics. The chosen metric for predictive performance is mAP50:95.

The baseline model is evaluated once before pruning. Each pruned model is evaluated after fine-tuning. The same confidence threshold, IoU threshold, image size, and deterministic settings are used for all evaluations. The confidence threshold is intentionally set low to include most candidate predictions for calculating metrics. Overlapping predictions are removed before metric calculations using *NMS* (Non-Maximum Suppression) with IoU threshold 0.7. This IoU threshold is not to be confused with the IoU threshold sweep used in the mAP50:95 calculation. Non-Maximum Suppression can be though of as discarding bounding boxes that are similar enough to likely be trying to predict the same object. The *NMS* threshold is used to compare predicted boxes to each other and remove predictions with, in this case, more than 70% overlap to avoid duplicate predictions. The mAP50:95 thresholds are used to decide how much a prediction must overlap with the true object to be considered a true positive (**TP**)

The evaluation uses

- confidence threshold 0.01
- NMS IoU 0.7
- image size $640 \times 640$

## 3.10 Inference Latency Measurement

Inference latency is measured to evaluate the practical computational impact of different pruning configurations. All latency measurements are performed using the same hardware and software environment to ensure consistent comparisons between experiments. The exported PyTorch models are first converted to TensorRT

engine files before benchmarking. The TensorRT engines are exported with image size $640 \times 640$, batch size 1, and FP16 precision.

The benchmarks are performed on an NVIDIA Jetson device and use synthetic input data rather than images loaded from disk. A fixed NumPy random seed is used to generate one synthetic RGB image of size $640 \times 640$. This avoids measuring file I/O overhead and ensures that the latency measurements focus on model inference and Ultralytics prediction overhead.

For each model, 20 warm-up iterations are executed before measurement. These warm-up iterations reduce initialization overhead and allow TensorRT, CUDA memory allocation, and hardware-specific optimizations to reach a steady state. After warm-up, latency is measured over 100 inference iterations. CUDA synchronization is performed before and after each measured inference call to ensure that asynchronous GPU execution is accounted for correctly.

Each measured iteration runs the Ultralytics `predict` function with image size $640 \times 640$, batch size 1, device 0, and all saving and display options disabled. The reported timings therefore represent end-to-end Ultralytics prediction latency, including preprocessing, model execution, and postprocessing.

Latency for a model is estimated as

$$T = \frac{1}{M} \sum_{i=1}^{M} t_i,$$

where $t_i$ denotes the measured inference time for iteration $i$, and $M = 100$ is the number of measured benchmark iterations after warm-up has been discarded.

# 4 Results

The results presented in this section evaluate the effects of different pruning configurations on predictive performance and inference latency for YOLOv8m and RT-DETR. For each pruning configuration, the achieved pruning ratio, mAP50:95, and inference latency are reported. The results are presented separately for each model architecture and pruning scheme.

## 4.1 YOLOv8m Results

For the YOLOv8 model, unstructured and structured pruning was performed. Results for N:M pruning on YOLOv8m are not considered due to tensor dimensions in most of the architecture not being suitable for the pruning scheme. Specifically, the YOLOv8 architecture, including YOLOv8m, contains mostly convolutional filters with dimesnions $3 \times 3 \times C$ where $C$ is the channel depth for the specific layer. The resulting matrix multiplications performed are therefore mostly ones with only 3 columns in at least one of the matrices. Since 2:4 is the ratio used for N:M pruning in this thesis, it is not suitable for these tensors. Tables 4 to 7 summarize the predictive performance and inference latency obtained for YOLOv8m under different pruning configurations. Figure 10 provides a visual comparison of the observed trends.

Table 4: Predictive performance (mAP50:95) with unstructured pruning on yolov8m.

| **Target Ratio** | **Orig.** | **0.05** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Actual Ratio** | - | 0.043 | 0.085 | 0.171 | 0.256 | 0.341 | 0.426 | 0.512 | 0.597 | 0.682 |
| Magnitude | 0.2124 | 0.2193 | 0.2184 | 0.2177 | 0.2162 | 0.2127 | 0.2088 | 0.1957 | 0.1818 | 0.1727 |
| First Order | 0.2124 | 0.2173 | 0.2137 | 0.2062 | 0.2039 | 0.1942 | 0.1846 | 0.1823 | 0.1652 | 0.1507 |

Table 5: Predictive performance (mAP50:95) with structured pruning on yolov8m.

| **Target Ratio** | **Orig.** | **0.05** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Actual Ratio** | - | 0.091 | 0.175 | 0.330 | 0.468 | 0.589 | 0.691 | 0.780 | 0.850 | 0.902 |
| Magnitude | 0.2124 | 0.2077 | 0.1968 | 0.1774 | 0.1709 | 0.1558 | 0.1356 | 0.1199 | 0.0999 | 0.0812 |
| First Order | 0.2124 | 0.2038 | 0.1889 | 0.1740 | 0.1583 | 0.1524 | 0.1382 | 0.1171 | 0.1026 | 0.0848 |
| Activation | 0.2124 | 0.2072 | 0.1915 | 0.1691 | 0.1628 | 0.1449 | 0.1328 | 0.1114 | 0.0975 | 0.0818 |

For unstructured pruning, predictive performance remained relatively stable at low pruning ratios for both evaluated importance criteria. Small improvements

over the baseline model were observed for several pruning configurations. As pruning ratios increased, mAP50:95 gradually decreased, with larger reductions occurring at the highest pruning levels.

For structured pruning, predictive performance decreased more rapidly as pruning increased. All evaluated importance criteria exhibited a monotonic reduction in mAP50:95, with the largest performance degradation occurring at the highest pruning ratios. Magnitude-based pruning achieved the highest mAP50:95 among the structured pruning methods across most pruning levels.

Table 6: Inference latency (ms) with unstructured pruning on yolov8m.

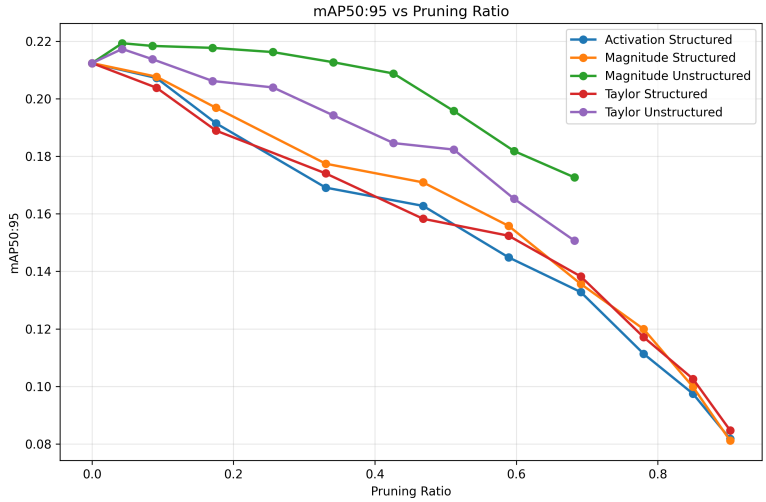
| **Target Ratio** | **Orig.** | **0.05** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Actual Ratio** | - | 0.043 | 0.085 | 0.171 | 0.256 | 0.341 | 0.426 | 0.512 | 0.597 | 0.682 |
| Magnitude | 17.980 | 18.221 | 18.117 | 18.068 | 18.030 | 17.983 | 18.074 | 18.061 | 18.014 | 18.027 |
| First Order | 17.980 | 18.106 | 18.060 | 18.178 | 18.031 | 18.105 | 17.984 | 18.151 | 17.977 | 17.966 |

Table 7: Inference latency (ms) with structured pruning on yolov8m.

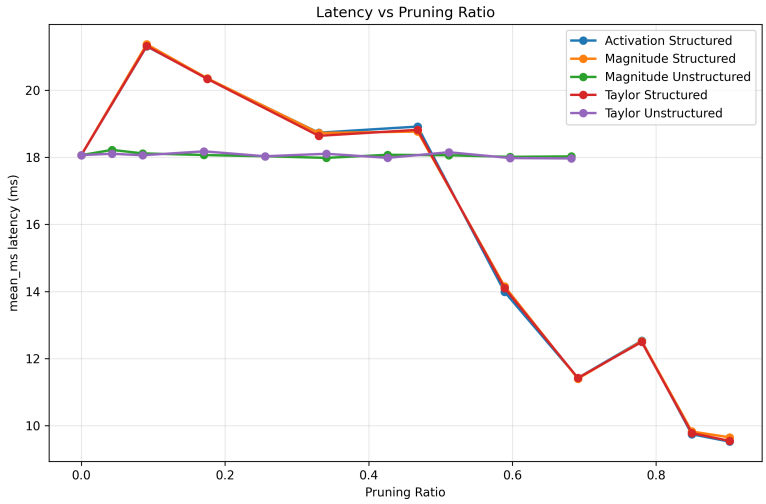
| **Target Ratio** | **Orig.** | **0.05** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Actual Ratio** | - | 0.091 | 0.175 | 0.330 | 0.468 | 0.589 | 0.691 | 0.780 | 0.850 | 0.902 |
| Magnitude | 17.980 | 21.371 | 20.357 | 18.726 | 18.767 | 14.155 | 11.398 | 12.516 | 9.823 | 9.650 |
| First Order | 17.980 | 21.318 | 20.338 | 18.640 | 18.819 | 14.100 | 11.419 | 12.494 | 9.777 | 9.544 |
| Activation | 17.980 | 21.308 | 20.350 | 18.733 | 18.916 | 13.988 | 11.420 | 12.530 | 9.742 | 9.522 |

The latency measurements show different behavior for the two pruning schemes. For unstructured pruning, inference latency remained close to the baseline value across all pruning ratios. In contrast, structured pruning produced substantial latency reductions at higher pruning levels. The largest latency improvements were observed for pruning ratios above 0.5, where inference times decreased to approximately half of the baseline latency.

Figure 10 visualizes the relationship between pruning ratio, predictive performance, and inference latency for YOLOv8m. The figure highlights the different trade-offs achieved by the evaluated pruning schemes and importance criteria.




(a) mAP50:95 versus pruning ratio.




(b) Inference latency versus pruning ratio.

Figure 10: Predictive performance and inference latency as a function of pruning ratio for YOLOv8m across all evaluated pruning methods.

## 4.2 RT-DETR Results

Tables 8 to 11 summarize the results obtained for RT-DETR. Unlike YOLOv8m, structured pruning could not be evaluated due to limitations encountered during dependency-aware graph tracing. Specifically, the graph-tracing was unable to trace the model graph and produce eligible pruning groups within reasonable time

constraints. Consequently, only unstructured pruning and N:M sparsity pruning are reported for RT-DETR.

Table 8: Predictive performance (mAP50:95) with unstructured pruning on RT-DETR.

| **Target Ratio** | **Orig.** | **0.05** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Actual Ratio** | - | 0.046 | 0.093 | 0.185 | 0.278 | 0.370 | 0.463 | 0.556 | 0.648 | 0.741 |
| Magnitude | 0.1576 | 0.1620 | 0.1597 | 0.1606 | 0.1593 | 0.1611 | 0.1623 | 0.1570 | 0.1532 | 0.1349 |
| First Order | 0.1576 | 0.1584 | 0.1577 | 0.1605 | 0.1569 | 0.1585 | 0.1562 | 0.1486 | 0.1356 | 0.1147 |

Table 9: Predictive performance (mAP50:95) with 2:4 pruning on RT-DETR.

| **Target Ratio** | **Orig.** | **0.5** |
|---|---|---|
| **Actual Ratio** | - | 0.461 |
| Magnitude | 0.1576 | 0.1541 |
| First Order | 0.1576 | 0.1480 |

The predictive performance results for RT-DETR show a gradual decline in mAP50:95 as pruning ratios increase. Similar to YOLOv8m, magnitude-based pruning generally achieved higher predictive performance than first-order Taylor pruning across most pruning ratios. The N:M sparsity experiments retained a large proportion of the baseline predictive performance despite removing approximately half of the eligible parameters.

Table 10: Inference latency (ms) with unstructured pruning on RT-DETR.

| **Target Ratio** | **Orig.** | **0.05** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Actual Ratio** | - | 0.046 | 0.093 | 0.185 | 0.278 | 0.370 | 0.463 | 0.556 | 0.648 | 0.741 |
| Magnitude | 34.484 | 34.039 | 34.149 | 34.015 | 33.659 | 33.481 | 33.610 | 32.906 | 32.662 | 32.900 |
| First Order | 34.484 | 34.304 | 34.265 | 34.219 | 33.789 | 33.744 | 33.574 | 33.202 | 32.684 | 32.800 |

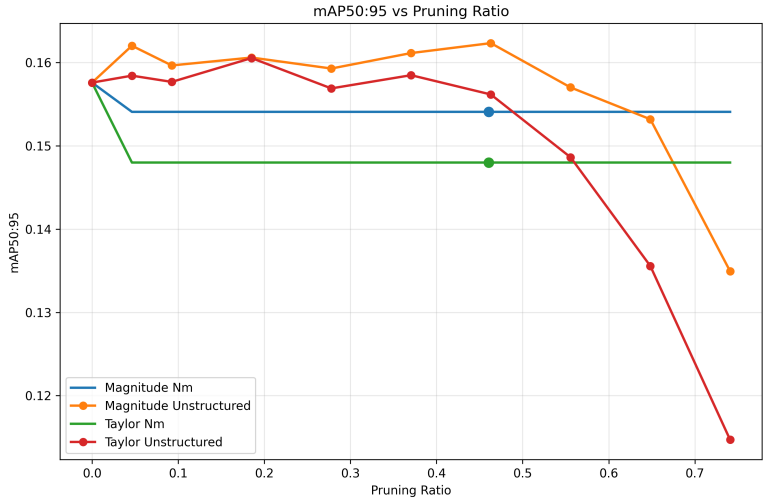
Table 11: Inference latency (ms) with 2:4 pruning on RT-DETR.

| **Target Ratio** | **Orig.** | **0.5** |
|---|---|---|
| **Actual Ratio** | - | 0.461 |
| Magnitude | 34.484 | 33.390 |
| First Order | 34.484 | 33.572 |

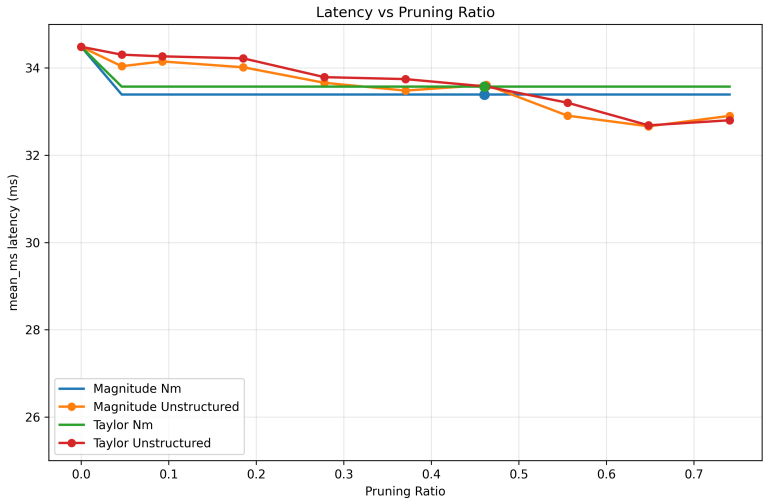
Inference latency for RT-DETR exhibited only modest changes across the evaluated pruning configurations. Unstructured pruning resulted in small latency reductions relative to the baseline model. The 2:4 sparsity configurations produced

similar latency behavior, with measured inference times remaining close to those of the original model.

Figure 11 presents the predictive performance and latency trends observed for RT-DETR. The figure allows direct comparison between the evaluated pruning methods and highlights how predictive performance and inference latency evolve as pruning increases.




(a) mAP50:95 versus pruning ratio.




(b) Inference latency versus pruning ratio.

Figure 11: Predictive performance and inference latency as a function of pruning ratio for RT-DETR across all evaluated pruning methods.

# 5 Discussion

The results demonstrate clear differences between unstructured and structured pruning in terms of both predictive performance and inference latency. Overall, unstructured pruning preserved predictive performance significantly better than structured pruning, while structured pruning produced substantially larger reductions in inference latency.

## 5.1 YOLOv8m Predictive Performance

The unstructured pruning results in Table 4 show that predictive performance remained relatively stable across moderate pruning ratios. In fact, small improvements in mAP50:95 were observed at low pruning ratios for both magnitude-based and first-order pruning. For example, magnitude-based unstructured pruning achieved a mAP50:95 of 0.2193 at target ratio 0.05 compared to the baseline value of 0.2124. Similar behavior can be observed for first-order pruning. This effect is consistent with previous observations that mild pruning can act as a form of regularization by removing redundant or noisy parameters. However, as pruning ratios increased, predictive performance gradually deteriorated. At high pruning levels, particularly above actual pruning ratios of approximately 0.5, the decrease became more pronounced.

The unstructured pruning results also show that magnitude-based pruning consistently outperformed first-order Taylor pruning in terms of predictive performance. Although the differences are relatively small at low pruning ratios, the gap widens at higher sparsity levels. At the highest evaluated pruning ratio, magnitude pruning retained a mAP50:95 of 0.1727 compared to 0.1507 for first-order pruning. This suggests that the simpler magnitude criterion was more robust for preserving predictive performance in the evaluated YOLOv8 architecture.

The structured pruning results in Table 5 show a much steeper decline in predictive performance as pruning increased. Unlike unstructured pruning, structured pruning physically removes filters and channels from the architecture, directly reducing representational capacity. This effect becomes increasingly severe at high pruning ratios. At the largest evaluated structured pruning ratio, all methods dropped below 0.10 mAP50:95.

Among the structured pruning criteria, magnitude-based pruning again achieved the strongest predictive performance overall, although activation-based pruning performed similarly at lower pruning ratios. First-order pruning generally produced the weakest results at larger pruning ratios. Figure 10a further illustrates

that structured pruning methods exhibit a nearly monotonic decrease in predictive performance as pruning increases, whereas unstructured methods maintain substantially higher mAP values across the same sparsity range.

## 5.2 YOLOv8m Inference Latency

The latency measurements reveal a strong contrast between unstructured and structured pruning. As shown in Table 6, unstructured pruning produced almost no measurable reduction in inference latency. Across all pruning ratios, latency remained close to the baseline value of approximately 18 ms. This behavior is expected because unstructured pruning only introduces sparse weight tensors while preserving the original dense tensor shapes. Standard inference hardware and libraries are typically optimized for dense matrix operations and therefore cannot efficiently exploit arbitrary sparsity patterns.

Structured pruning, however, produced substantial latency reductions, as shown in Table 7. At moderate and high pruning ratios, latency improvements became significant. For example, magnitude-based structured pruning reduced latency from approximately 18 ms to below 10 ms at the highest pruning levels. This demonstrates one of the primary advantages of structured pruning: because channels and filters are physically removed, the resulting network becomes smaller and computationally cheaper in a way that standard dense hardware can efficiently utilize.

Interestingly, the latency improvements for structured pruning were not perfectly monotonic. Some pruning configurations produced slightly worse latency than neighboring pruning ratios. For example, latency increased slightly between target ratios 0.5 and 0.6 for several structured methods. This may be explained by implementation details in TensorRT optimization, memory alignment effects, or changes in layer dimensions that interact differently with the underlying hardware kernels. One possible explanation for the inference being lower at target ratio 0.5 than surrounding points, could be that tensors often multiples two, or multiples of powers of two. When reducing one of the tensor dimensions by 50%, the resulting dimension may often still be a multiple of two, which typically aligns better with hardware. These effects highlight that theoretical parameter reduction does not always translate into practical runtime improvements.

Another important observation is that low levels of structured pruning initially increased latency compared to the original model. For example, the structured pruning configurations at ratios 0.05 and 0.1 produced inference times above 20 ms, exceeding the baseline latency of 17.98 ms. This likely occurs because small structural modifications may disrupt highly optimized execution paths without yet

reducing computation enough to offset the overhead. Only after larger architectural reductions do clear latency gains emerge.

Taken together, the results demonstrate a clear trade-off between predictive performance and inference efficiency. Unstructured pruning is more effective for preserving accuracy but does not significantly improve practical inference latency on ordinary dense hardware. Structured pruning, on the other hand, provides substantial latency reductions but causes a larger degradation in predictive performance. This trade-off is particularly visible in Figure 10b, where the structured pruning curves decline more rapidly than the unstructured pruning curves as pruning increases.

The results also suggest that magnitude-based pruning is a strong baseline across both pruning schemes. Despite its simplicity, it consistently matched or outperformed the more computationally expensive first-order criterion. This is consistent with prior work showing that simple magnitude heuristics can be highly competitive in practice.

## 5.3 RT-DETR

The RT-DETR experiments exhibited several trends that were consistent with the YOLOv8m results. As shown in Table 8 and Figure 11a, predictive performance gradually decreased as pruning increased, and magnitude-based pruning generally achieved equal or better performance than first-order Taylor pruning across the evaluated pruning ratios.

One important observation is that the predictive performance was worse across the board for RT-DETR. While it has been observed in previous work that RT-DETR can beat YOLOv8 in object detection [10], the shortcomings in this work may be explained by the different architectures' abilities to adapt to the new dataset. It is possible that RT-DETR, a larger and significantly more complex model, could struggle to adapt to the new dataset using the implemented training procedure. With additional training optimizations, RT-DETR could potentially perform equally or even surpass YOLOv8 on VisDrone. The results presented in this thesis should not, however, be viewed as a comparison between YOLOv8 and RT-DETR comparison, as the aim is to investigate the effects of pruning on a given starting model.

The unstructured pruning results in Table 8 indicate that RT-DETR is capable of tolerating substantial parameter removal before experiencing severe degradation in predictive performance. At moderate pruning ratios, both magnitude-based and first-order pruning retained a large proportion of the baseline mAP50:95. This sug-

gests that RT-DETR may contain a considerable degree of parameter redundancy that can be removed without immediately harming detection performance.

As with YOLOv8m, magnitude-based pruning consistently outperformed the first-order criterion, as shown in Table 8. This result is somewhat surprising given that Taylor-based methods attempt to explicitly estimate the impact of pruning on the loss function. One possible explanation is that the Taylor importance estimates were calculated using a single calibration batch, introducing noise into the importance estimates. Under these conditions, the simpler magnitude criterion may provide a more stable and robust measure of parameter importance.

The N:M sparsity experiments shown in Table 9 further demonstrate that RT-DETR can maintain a large proportion of its baseline predictive performance despite removing nearly half of the eligible parameters. Compared to unstructured pruning, the N:M constraints introduce additional restrictions on which parameters may be retained, yet the observed reduction in mAP50:95 remained relatively modest. This suggests that semi-structured sparsity may provide a practical compromise between flexible unstructured pruning and hardware-oriented sparsity patterns.

The latency measurements for RT-DETR shown in Table 10 and Table 11 differ somewhat from those observed for YOLOv8m. While pruning produced measurable reductions in inference latency, the improvements were relatively modest compared to the gains achieved through structured pruning in YOLOv8m. The best RT-DETR configurations reduced latency by only a few milliseconds relative to the baseline model. This is expected, since both unstructured pruning and N:M sparsity preserve the overall architecture and tensor dimensions of the network. Consequently, much of the computational graph remains unchanged, limiting the ability of standard inference engines to translate parameter sparsity into runtime acceleration.

These trends can be seen clearly in Figure 11b, where latency decreases only gradually as pruning increases. In contrast to the structured YOLOv8m results shown in Figure 10b, no comparable reduction in runtime is observed despite substantial parameter sparsity.

## 5.4 Overall Comparison and Implications

Comparing the two architectures therefore suggests that the relationship between parameter sparsity and practical inference efficiency is highly dependent on the pruning structure rather than the underlying network architecture. Both YOLOv8m and RT-DETR preserved predictive performance reasonably well under unstruc-

tured pruning, as shown in Figure 10a and Figure 11a, but only the structured pruning experiments on YOLOv8m produced substantial latency reductions. This highlights the importance of considering deployment constraints when selecting a pruning strategy, as high parameter sparsity alone does not necessarily translate into faster inference.

Several limitations should also be acknowledged. First, the experiments were conducted only on YOLOv8m and RT-DETR using the VisDrone dataset. Different architectures or datasets may exhibit different pruning behavior. Second, only one-shot pruning was evaluated. Iterative pruning combined with repeated fine-tuning could potentially recover additional predictive performance, especially at high sparsity levels. Finally, the latency measurements were performed on a specific TensorRT and Nvidia Jetson hardware configuration, meaning the exact runtime behavior may differ on other accelerators or inference frameworks.

Overall, the experiments show that the choice of pruning strategy depends strongly on the deployment objective. If preserving predictive performance is the primary goal, unstructured pruning appears preferable. If practical inference acceleration on standard hardware is required, structured pruning provides substantially larger benefits despite the greater loss in predictive performance.

# 6 Conclusions

This thesis investigated the effects of neural network pruning on predictive performance and inference latency for modern object detection models. A modular pruning framework supporting unstructured pruning, structured pruning, and N:M sparsity pruning was implemented and evaluated using the YOLOv8m and RT-DETR object detectors on the VisDrone dataset. The objective was to analyze how different pruning strategies influence the trade-off between predictive performance and inference efficiency.

The first research question concerned how predictive performance changes as the degree of pruning increases for different pruning methods. The experimental results demonstrated that predictive performance generally decreases as pruning ratios increase, regardless of pruning strategy. However, the rate of degradation differed substantially between pruning schemes and importance criteria. For YOLOv8m, unstructured pruning consistently preserved predictive performance better than structured pruning at comparable pruning levels. At low pruning ratios, unstructured pruning occasionally produced small improvements in mAP50:95 compared to the baseline model. One possible explanation is that mild pruning acts as a form of regularization by removing redundant parameters. Structured pruning resulted in a more rapid decline in predictive performance, particularly at high pruning ratios where a large portion of the model capacity was removed.

The RT-DETR experiments exhibited similar trends. Unstructured pruning maintained predictive performance across a wide range of pruning ratios, while N:M sparsity pruning preserved a large proportion of baseline performance despite removing nearly half of the eligible parameters. Although predictive performance gradually decreased as pruning increased, RT-DETR demonstrated substantial robustness to parameter removal, indicating that significant redundancy exists even within transformer-based object detection architectures.

Among the evaluated importance criteria, magnitude-based pruning consistently achieved the strongest predictive performance across most experiments. Although first-order Taylor pruning and activation-based pruning produced competitive results at lower pruning ratios, their performance generally deteriorated more rapidly as pruning increased. These findings suggest that simple magnitude-based importance measures remain highly effective despite their relatively low computational complexity. Perhaps the most surprising result was that magnitude-based pruning consistently matched or outperformed first-order Taylor pruning across both YOLOv8m and RT-DETR. This finding suggests that the additional computational cost required to estimate gradient-based importance may not necessarily

translate into superior pruning decisions for the evaluated object detection models.

The second research question concerned the relationship between pruning and inference latency. The results showed a clear distinction between unstructured and structured pruning. Unstructured pruning produced little or no measurable reduction in inference latency despite achieving substantial parameter sparsity. This outcome is expected because the underlying dense tensor structures remain unchanged and modern hardware is typically optimized for dense computations. Structured pruning, in contrast, produced significant latency reductions by physically removing channels and filters from the network architecture. At higher pruning ratios, inference latency was reduced by nearly fifty percent relative to the baseline model. These results demonstrate that structured pruning is considerably more effective for practical inference acceleration on conventional hardware platforms.

The N:M sparsity experiments demonstrated that semi-structured pruning can preserve much of the predictive performance achieved by unstructured pruning while enforcing sparsity patterns that are more compatible with specialized hardware acceleration. However, the evaluated deployment platform was unable to fully exploit these sparsity patterns, resulting in only modest latency improvements. Upon inspection, the accelerator did in fact identify several tensors eligible for 2:4 acceleration, but chose to use standard dense kernels for the computation. Why the sparsity was not exploited remains unclear. This observation highlights the importance of considering hardware support when evaluating sparsity-based acceleration techniques.

The experiments also highlight the importance of considering deployment objectives when selecting a pruning strategy. If maintaining predictive performance is the primary objective, unstructured pruning provides the most favorable trade-off. If reducing inference latency is the primary objective, structured pruning offers substantially greater practical benefits despite a larger reduction in predictive performance. N:M sparsity pruning occupies an intermediate position by introducing hardware-friendly sparsity patterns while retaining many of the advantages of sparse pruning.

The experiments also demonstrated that target pruning ratios do not necessarily correspond to achieved parameter reductions. Actual pruning ratios varied depending on the pruning scheme, particularly for structured pruning where dependency propagation caused additional parameters to be removed. This highlights the importance of reporting achieved sparsity rather than relying solely on target pruning ratios when comparing pruning methods.

Overall, the results demonstrate that neural network pruning remains a viable approach for improving deployment efficiency in object detection systems. The

findings contribute to a better understanding of how different pruning schemes affect modern computer vision models and provide practical guidance for selecting pruning strategies based on application requirements and hardware constraints.

## 6.1 Limitations

Several limitations should be considered when interpreting the results presented in this thesis.

First, the experiments were limited to the VisDrone dataset and two object detection architectures, YOLOv8m and RT-DETR. Although these models represent both convolutional and transformer-based approaches to object detection, the observed pruning behavior may not generalize directly to other architectures, datasets, or computer vision tasks.

Second, all experiments employed one-shot pruning followed by fine-tuning. While this provides a controlled experimental setup and simplifies comparisons between pruning configurations, iterative pruning strategies may produce different trade-offs between compression and predictive performance. Previous research has shown that iterative pruning can often recover more performance at high sparsity levels.

Third, structured pruning experiments could only be completed for YOLOv8m. Although the implemented framework supports dependency-aware structured pruning through graph tracing, the tracing process was not sufficiently robust for the RT-DETR architecture. Transformer-based detection models contain complex dependencies between attention modules, projection layers, and decoder components that were not fully supported by the pruning framework. Consequently, RT-DETR was evaluated only using unstructured pruning and N:M sparsity pruning.

Fourth, latency measurements were conducted using a single hardware and software configuration. While this provides a consistent basis for comparison between pruning methods, inference behavior may differ on other accelerators, inference engines, or deployment environments. In particular, hardware specifically designed for sparse computations may achieve larger benefits from sparse pruning methods than those observed in this study.

Finally, the evaluation focused primarily on predictive performance and inference latency. Other deployment-related metrics such as memory usage, energy consumption, throughput, and model storage size were not investigated and may provide additional insight into the practical benefits of pruning.

## 6.2 Further work

Several directions for future work emerge from the findings of this thesis.

One natural extension is to evaluate the pruning framework on a wider range of computer vision architectures. While this study considered both CNN-based and transformer-based object detectors, additional architectures would provide a broader understanding of how pruning behavior varies across model families. In particular, future work could investigate whether the observed superiority of magnitude-based pruning generalizes across different architectures and datasets.

Another promising direction is the investigation of iterative pruning strategies. Rather than applying the desired pruning ratio in a single step, iterative pruning gradually increases sparsity while repeatedly fine-tuning the model. This approach may improve predictive performance at higher pruning ratios and could potentially reduce the performance gap observed between structured and unstructured pruning.

Future work should also investigate structured pruning for transformer-based object detectors such as RT-DETR. The inability to reliably perform dependency-aware graph tracing for RT-DETR highlights the need for improved pruning techniques and tooling for modern transformer architectures. Architecture-specific pruning strategies for attention layers, feed-forward networks, and decoder modules may enable more effective structured compression of transformer-based detectors.

Another interesting direction is the combination of pruning with complementary model compression techniques. Quantization, knowledge distillation, low-rank decomposition, and neural architecture search could potentially be combined with pruning to achieve greater reductions in model size and computational cost while preserving predictive performance.

Further research could also further investigate hardware-aware pruning strategies. N:M sparsity patterns are specifically designed to align with sparse acceleration support available in modern hardware. While the hardware used in this work managed to find eligible tensors for 2:4 acceleration, no such sparse acceleration was actually observed. Investigating the specific hardware and software configurations required to translate such sparsity into measurable computational speedups would be valuable. This could help bridge the gap between theoretical efficiency gains and actual deployment performance.

Finally, future studies could expand the evaluation beyond predictive performance and latency by incorporating additional deployment metrics such as energy consumption, memory footprint, throughput, and thermal efficiency. Such metrics

are increasingly important for edge computing, autonomous systems, and other resource-constrained deployment scenarios where efficiency is often as important as predictive performance.

# References


[1] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*. 2021.

[2] Gongfan Fang et al. “DepGraph: Towards Any Structural Pruning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.

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

[4] Song Han, Huizi Mao, and William J Dally. “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”. In: *International Conference on Learning Representations*. 2016.

[5] Song Han et al. “Learning Both Weights and Connections for Efficient Neural Networks”. In: *Advances in Neural Information Processing Systems*. 2015.

[6] Babak Hassibi and David Stork. “Second Order Derivatives for Network Pruning: Optimal Brain Surgeon”. In: *Advances in Neural Information Processing Systems*. 1993.

[7] Yihui He, Xiangyu Zhang, and Jian Sun. “Channel Pruning for Accelerating Very Deep Neural Networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017.

[8] Yann LeCun, John Denker, and Sara Solla. “Optimal Brain Damage”. In: *Advances in Neural Information Processing Systems*. 1990.

[9] Hao Li et al. “Pruning Filters for Efficient ConvNets”. In: *International Conference on Learning Representations*. 2017.

[10] Wenyu Lv et al. “DETRs Beat YOLOs on Real-time Object Detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024.

[11] Asit Mishra and Deborah Marr. “Accelerating Sparse Deep Neural Networks”. In: *arXiv preprint arXiv:2104.08378* (2021).

[12] Pavlo Molchanov et al. “Pruning Convolutional Neural Networks for Resource Efficient Inference”. In: *International Conference on Learning Representations*. 2017.

[13] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems* (2019).

[14] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Advances in Neural Information Processing Systems*. 2015.

[15] Ultralytics. *RT-DETR-L model configuration (rtdetr-l.yaml)*. 2023. URL: https://github.com/ultralytics/ultralytics/blob/main/ultralytics/cfg/models/rt-detr/rtdetr-l.yaml (visited on 06/07/2026).

[16] Ultralytics. *Ultralytics block modules (block.py)*. 2025. URL: https://github.com/ultralytics/ultralytics/blob/main/ultralytics/nn/modules/block.py (visited on 04/21/2026).

[17] Ultralytics. *Ultralytics transformer modules (transformer.py)*. 2025. URL: https://github.com/ultralytics/ultralytics/blob/main/ultralytics/nn/modules/transformer.py (visited on 06/07/2026).

[18] Ultralytics. *YOLOv8 model configuration (yolov8.yaml)*. 2023. URL: https://github.com/ultralytics/ultralytics/blob/main/ultralytics/cfg/models/v8/yolov8.yaml (visited on 04/21/2026).

[19] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems*. 2017.

[20] Mingjian Zhu, Shuai Tang, and Xiangxiang Chen. “Vision Transformer Pruning”. In: *arXiv preprint arXiv:2104.08500* (2021).

[21] Pengfei Zhu et al. “Detection and Tracking Meet Drones Challenge”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021). DOI: 10.1109/TPAMI.2021.3119563.