Compressing YOLOv7

Benjamin van Zwienen

Master of Science Thesis

November 24, 2023

Student Number: 4471180
Supervisor: Prof. Dr. Ir. M. Wisse

Abstract

In the literature, neural network compression can significantly reduce the number of floating-point operations (FLOPs) of a neural network with limited accuracy loss. At the same time, it is common to manually design smaller networks instead of using modern compression techniques. This thesis will compare the two approaches for the object detection network YOLOv7. YOLOv7 can run in real time on a desktop GPU. For edge GPUs a smaller version, called YOLOv7-tiny, was manually designed by the authors of YOLOv7. This thesis answers the question: *Can a state-of-the-art compression of YOLOv7 achieve higher accuracy than YOLOv7-tiny at the same number of floating-point operations?*

First, two state-of-the-art compression methods are selected and compared on YOLOv7-tiny. Then the best performing method, GBIP, is used to compress YOLOv7 till it has the same number of FLOPs as YOLOv7-tiny. From the experiments it is determined that GBIP is not able to achieve higher accuracy than YOLOv7-tiny at the same number of FLOPs.

Contents

1 Introduction

1.1 Motivation

In the last decade, the performance of neural networks has increased dramatically. They have become state of the art in a wide range of areas, such as image classification, object detection and localization, and speech recognition [1, 2]. A lot of the improvements have come through creating larger and more complex models. Besides a larger memory footprint, complex neural networks take longer to train and run inference and have higher energy consumption. Therefore, neural networks are usually run on servers with high-end GPUs, allowing for the increased computational demand needed for increased accuracy [3].

There are a lot of edge devices, like smartphones, home automation devices and robots, which could also benefit from large neural networks. There are two paradigms for providing machine learning to these devices: cloud computing and edge computing [4]. In the first case, the edge device sends its data to a server with powerful hardware, which runs neural network and sends back the result [5]. In the latter case, all the computation is handled on the edge device itself.

There are several drawbacks to cloud computing since all data must be transmitted to a server [6, 7]. Especially for edge devices with a lot of sensors, the network bandwidth will become a problem. For example, an autonomous car is estimated to produce 3 Gbit of data per second [8]. For real time applications, the latency of the connection might be too high to function at the desired speed. Also, in safety-critical situations, the internet connection must be extremely reliable. And lastly, certain edge devices, especially health monitoring devices, collect sensitive, personal data which people might not want uploaded to a server.

Although cloud computing has its drawbacks, edge computing is not always the obvious choice. Edge devices are typically constrained in terms of memory, computation and energy usage. This makes running large neural networks on these devices infeasible. Assuming large neural networks are required, upgrading the hardware or using cloud computing are reasonable options. However, there has also been a lot of research into creating smaller neural networks that can deal with the constraints of edge devices [5, 9, 10]. The creation of smaller networks can be split into two categories: designing new, compact architectures and compressing existing architectures.

Compression of existing neural networks is based on the observation that large networks only need a part of their parameters for accurate predictions [11]. The amount of compression depends on the network and dataset, with a reduction in floating-point operations (FLOPs) of more than 99% for LeNet-5 on MNIST [12] and up to 89% for ResNet-56 on CIFAR-10 [13], all while the accuracy loss stays below 1%.

Compact architectures are difficult to compare to other compressed networks since they typically do not have a clear baseline. For example, MobileNet [14] loses less than 1% accuracy with only 4% of FLOPs compared to VGG-16. However, MobileNet is not a smaller version of VGG-16, so it might just as well be compared with any other neural network. On the other hand, there are some architectures that could be compared, like the different ResNets [15]. Maybe compressing ResNet-110 will lead to a model with higher accuracy and lower FLOPs than ResNet-50. No such comparison, between compact and compressed networks, has been found in the literature.

1.2 Research Questions

Given the promising results of neural network compression, and the lack of comparison between compressed and compact networks, this thesis will focus on comparing these two for the case of YOLOv7 [16]. YOLOv7 is one of the best performing object detection networks. But given its size, it is meant to run on a powerful GPU. The creators of YOLOv7 have hand-designed YOLOv7-tiny especially for the use on edge devices. The goal of this thesis is to see if a state-of-the-art compression method will be able to compress YOLOv7 more efficiently than the hand-designed YOLOv7-tiny.

Main Research Question

The main research question of this thesis is:

> *Can a state-of-the-art compression of YOLOv7 achieve higher accuracy than YOLOv7-tiny at the same number of floating-point operations?*

Based on the compression literature, it seems plausible that a compression method should be able to outperform YOLOv7-tiny.

Sub Questions

To answer the main research question, several sub questions will be answered throughout this thesis. These sub questions are listed and motivated below:

1. *What is the state of the art in neural network compression?*

It is important to understand what compression techniques exist and how they work, as well as the state-of-the-art performance.

2. *What are the best networks for comparing compression methods?*

To make a fair comparison between compression methods, they should be evaluated on the same networks. It is, therefore, important to determine which networks are most suited for this comparison.

3. *Which compression method is best suited for compressing YOLOv7?*

Based on the latest research, two of the best methods will be assessed on YOLOv7-tiny to see if they can be adapted to the YOLOv7 architecture and select the best performing method. An important constraint on the compression methods for this thesis is the amount of finetuning required. There is no hard limit, but the compression method should be able to work on a standard GPU and complete within hours or days rather than weeks. This is also why YOLOv7-tiny is initially used to compare methods. YOLOv7-tiny runs and trains much faster, while the architecture is very similar to YOLOv7. Typically, both the reduction in FLOPs and parameters is reported for the compression of neural networks. This is also the case in this thesis. However, object detection is usually run in real-time, which makes speed, and thus FLOPs, the key factor.

4. *What are the optimal hyperparameters to compress YOLOv7?*

The best performing method on YOLOv7-tiny will be used to compress YOLOv7 till it reaches the same speed as YOLOv7-tiny, as measured by the number of FLOPs. To achieve the best result, the hyperparameters of this method have to be adapted to YOLOv7.

5. *Does one large pruning step work better than several smaller pruning steps?*
The selected compression method runs multiple pruning steps. Therefore, this thesis will also test if several small pruning steps or one large step achieves better results.

6. *Why is the selected compression method not able to outperform YOLOv7-tiny?*
Surprisingly, it turns out the compressed YOLOv7 model is not able to outperform YOLOv7-tiny. The results have to be examined, to see why this is the case.

1.3 Outline

Chapter 2 will provide the background knowledge for this thesis: a short introduction to neural networks and YOLOv7, is followed by an extensive overview of neural network compression. Chapter 3 will explain the search method and the selected compression methods. These methods will then be adapted for YOLOv7-tiny, which is the topic of Chapter 4, as well as the proposed experiments. In Chapter 5 the results of these experiments are given, followed by the selection of the best method for YOLOv7.

The adaptation of the selected method to YOLOv7, including the optimization of the hyperparameters, and the experiments are discussed in Chapter 6. The results are given in Chapter 7. Based on these results, the research questions will be answered in Chapter 8, and recommendations for future research are given.

2 Background

This chapter will provide the background for the rest of the thesis. It starts by introducing the basics of neural networks in Section 2.1. Section 2.2 will explain YOLOv7, the neural network used in this thesis. A deep dive into the compression of neural networks is presented in Section 2.3.

2.1 Neural Networks

The notation used in the rest of the thesis will be introduced in this section. Followed by a brief overview of object detection and the relevant metrics.

2.1.1 Basics

This thesis assumes a basic understanding of neural networks. Here, the activation functions used in YOLOv7 are introduced, as well as the notation for a convolutional layer. Different papers use slightly different notations, thus it is important to explicitly specify the notation used in this thesis.

Activation functions

Activation functions introduce nonlinearity to a neural network. They are usually added after each fully-connected or convolution operation. For a long time, the most common activation function was the Rectified Linear Unit (ReLU) [17]. In the past years more complex functions have been proposed [18]. One such activation function, which is used in YOLOv7, is the Sigmoid Linear Unit (SiLU) [19].

Another common activation function is the sigmoid function. In case of YOLOv7, it is used in the final layer. Given that the output of the sigmoid function is between 0 and 1, it can be interpreted as a probability. The equations for these three activation functions are given in Equations 2.1-2.3 and are plotted in Figure 2.1.

$$f_{\text{ReLU}}(x) = \max(0, x) \tag{2.1}$$

$$f_{\text{sigmoid}}(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

$$f_{\text{SiLU}}(x) = x\sigma(x) \tag{2.3}$$

Convolutional Layer

The operation of a convolutional layer can be expressed as:

$$Y_n = f\left(\sum_{c=1}^{C} W_{n,c} * X_c + b_n\right) \tag{2.4}$$

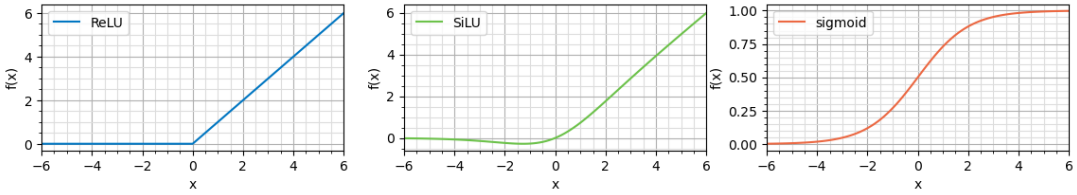


Figure 2.1: Activation functions.

(a) Example 1 (b) Example 2

Figure 2.2: Output of an object detection network for two images from the MS COCO dataset. Detected objects are classified and located with a bounding box.

With output $Y \in \mathbb{R}^{N \times H_{out} \times W_{out}}$, weights $W \in \mathbb{R}^{N \times C \times K_h \times K_w}$, input $X \in \mathbb{R}^{C \times H_{in} \times W_{in}}$ and bias $b \in \mathbb{R}^N$. Where N is the number of output channels, C the number of input channels, W and H the width and height of the channels, K_H and K_W the height and width of the convolution kernels, and f the activation function.

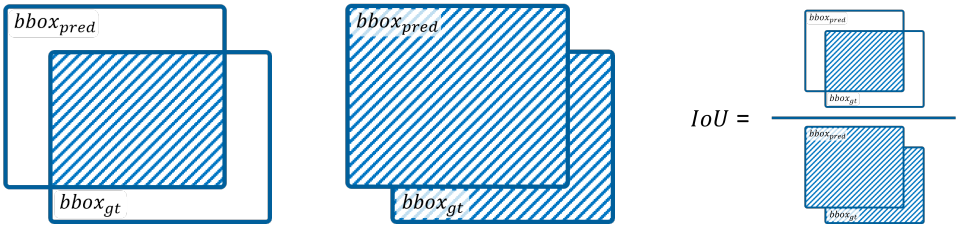
The input and output channels can also be referred to as input and output feature maps. Though in some cases the output feature maps only refer to the direct result of the convolution before the activation function is applied. In this thesis, the output feature maps will refer to the post-activation feature maps, unless explicitly mentioned otherwise.

The weights W of a convolutional layer can be split into N 3D filters. Each of these filters convolved with the input X corresponds to one of the output feature maps Y_n.

2.1.2 Object Detection

Object detection is the task of locating and classifying objects in an image. For each object, the network should output a bounding box that fits closely around the object and produce a classification for that object. Figure 2.2 shows the output of an object detection network for two different images.

This subsection provides background on the metrics, datasets and models used for object detection. The metrics and datasets are used for YOLOv7, while a short overview of the existing models provides context.



(a) Intersection (b) Union (c) IoU

Figure 2.3: Intersection over Union (IoU) of a predicted bounding box, $bbox_{pred}$, and the ground truth, $bbox_{gt}$.

Metrics

To determine how well an object detection network is performing, first the proposed bounding boxes must be evaluated. This is done using the Intersection over Union (IoU) metric:

$$IoU = \frac{bbox_{gt} \cap bbox_{pred}}{bbox_{gt} \cup bbox_{pred}} \tag{2.5}$$

This measures how close the predicted bounding box, $bbox_{pred}$, matches with the ground truth, $bbox_{gt}$. Figure 2.3 visualizes the IoU equation. The IoU is zero when the bounding boxes have no overlap, and one when the bounding boxes are identical.

Precision and recall are defined in terms of True Positives (TP), False Positives (FP) and False Negatives (FN):

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.7}$$

It is unrealistic to expect all predicted bounding boxes to match perfectly with the ground truth. However, they must match to some extent. This extent is measured with the IoU metric. To compute the number of true and false positives for the precision and recall, only the predicted bounding boxes are used that have an IoU above a certain threshold. This threshold is typically set to a value of 0.5 or higher.

The precision is largest when there are no false positives, while the recall is largest when there are no false negatives. To avoid false positives fewer bounding boxes could be proposed, but this creates more false negatives. Thus, there is a tradeoff between precision and recall.

This tradeoff can be captured in a precision-recall curve $p(r)$. This curve is created by ordering all predictions from highest to lowest confidence and computing intermediate precision and recall from top to bottom. At high confidence, the precision will be high since it is unlikely to be very certain about an incorrect prediction. At the same time, the number of false negatives will be higher if lower confidence predictions are ignored. An example of a precision-recall curve is shown in Figure 2.4.

Finally, based on an interpolated version of the precision-recall curve (Figure 2.4), the most common metric for evaluating object detection networks, namely Average Precision (AP), can be computed. The interpolated precision-recall curve $p_{interp}(r)$ is given as:

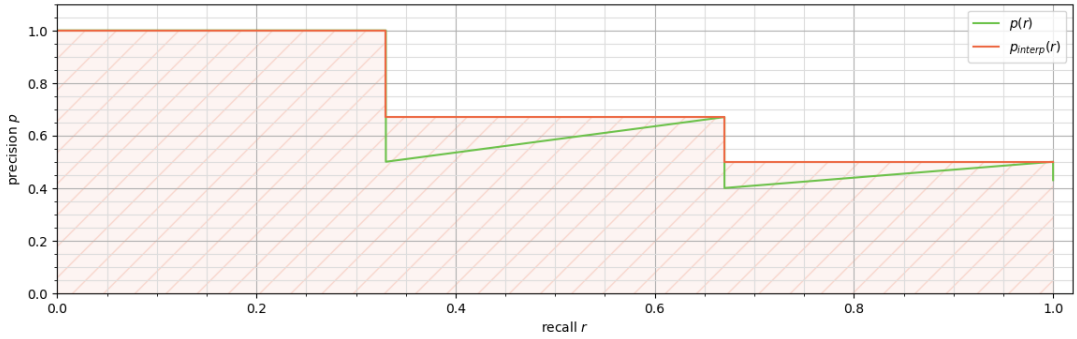


Figure 2.4: Precision-recall curve $p(r)$ and interpolation $p_{interp}(r)$. The colored area under $p_{interp}(r)$ is the Average Precision (AP), which is the metric used in this thesis. Adapted from [20].

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{2.8}$$

The AP is simply defined as the area under the interpolated precision-recall curve. The area is computed by sampling $p_{interp}(r)$ at 101 points:

$$AP = \frac{1}{101} \sum_{i=0}^{100} p_{interp}(0.01i) \tag{2.9}$$

Since the definition of AP includes precision and recall, it requires an IoU threshold. For example, AP_{50} refers to the AP at an IoU threshold of 0.5. The AP can also be an average at different IoU thresholds. $AP_{50:95}$ refers to the average of AP_{50}, AP_{55}, ..., AP_{95}. Each object class has its own AP, for example AP^{car} or AP^{person}, but typically only the average over all classes is reported. This is called the mean Average Precision, mAP. Following common notation, in this thesis, unless specifically stated otherwise, the mean Average Precision at IoUs from 0.5 to 0.95, $mAP_{50:95}$, will be referred to as simply AP.

Datasets

The two most common datasets for training and benchmarking object detection networks are PASCAL VOC [21] and MS COCO [22]. PASCAL VOC is an older dataset with official competitions from 2005 to 2012. The 2012 version of the dataset has 20 classes and consists of 6k training, 6k validation and 11k test images. The best reported result on PASCAL VOC achieves an AP of 97.2% [23], which makes it difficult to show any further improvement. Most newer methods do no longer report results for PASCAL VOC.

Instead, MS COCO has become the default benchmarking dataset for object detection, with competitions from 2015 to 2020. The dataset was last updated in 2017 and now contains 118k training, 5k validation and 41k test images. Objects have to be classified into 80 categories, which include the same 20 categories as PASCAL VOC. Compared to PASCAL VOC, MS COCO contains objects at all different scales, including a large number of small objects [24]. This makes MS COCO a more challenging dataset than PASCAL VOC. Currently the best reported result on MS COCO is an AP of 66.0% [25].

Models

There are two types of deep learning approaches for object detection: two-stage and one-stage object detectors [26]. Two-stage networks split the detection of objects into two parts: (i) find regions in an image where objects may be present, (ii) classify these region proposals. One-stage object detectors do not use region proposals, but instead predict all bounding boxes and classifications in one go.

A common example of a two-stage detector is Faster R-CNN [27] which builds on Fast R-CNN [28]. With Faster R-CNN the region proposal network and the classifier network are not separated. Firstly, a set of feature maps is generated from an input image using convolutional layers. Then, based on these feature maps the region proposals are generated. Finally, the classifier takes the feature maps and looks at the regions that were proposed. Two-stage detectors are usually slower since they must first find region proposals and then run a classifier for all these regions. Usually, the increase in inference time will be compensated by an accuracy increase [29]. However, for real-time applications this might not be a beneficial tradeoff.

For this reason, one-stage detectors are more likely to be used in real-time applications. Commonly referenced one-stage detectors are SSD [30] and different YOLO versions. Both SSD and Faster R-CNN were introduced in 2015, and are no longer state-of-the-art, achieving 28.8% and 34.9% AP on MS COCO, respectively. The original YOLO [31] network was also introduced in 2015, but newer versions, like YOLOv7 [16], are still in use. YOLOv7 was specifically designed for real-time applications and runs 10 times faster than most other non-YOLO methods. It achieves 51.4% AP on MS COCO. As mentioned above, currently the best reported result on MS COCO is 66.0% AP. Thus, YOLOv7 should not be used if very high accuracy is needed, and slower inference time is acceptable.

2.2 YOLOv7

This thesis deals with the compression of YOLOv7. An explanation of this object detection neural network is given in this section. Subsection 2.2.1 deals with the structure of YOLOv7 and its different versions. Next, the computation of the output and its interpretation is discussed in Subsection 2.2.2. And lastly, the training of YOLOv7 is described in Subsection 2.2.3.

2.2.1 Structure

Versions

There are three different YOLOv7 versions for three different types of GPU: YOLOv7-tiny for edge GPUs, YOLOv7 for normal desktop GPUs and YOLOv7-W6 for cloud GPUs. They all share the same type of architecture but are specifically designed to run in real-time on the respective hardware. Table 2.1 shows the difference in number of parameters and FLOPs, the speed at which each version runs on a NVIDIA V100 GPU, and the achieved AP on MS COCO. The tradeoff between speed and accuracy is clear, with YOLOv7-tiny optimizing for speed at the cost of accuracy. YOLOv7-tiny and YOLOv7 are evaluated with input images resized to 640x640, while YOLOv7-W6 is designed for input images of 1280x1280. For the rest of this thesis all reported results are obtained using 640x640 images.

Table 2.1: Comparison of the different YOLOv7 versions [16]. Reported FPS is achieved on a NVIDIA V100 GPU.

Version	#Params	#FLOPs	FPS	AP
YOLOv7-tiny	6.2M	13.8G	286	38.7
YOLOv7	36.9M	104.7G	161	51.4
YOLOv7-W6	70.4M	360.0G	84	54.9

Information Flow

A simplified overview of the YOLOv7 architecture is shown in Figure 2.5. The numbers on the left indicate the scale at that level. For example, at the top (32x) the input image has been downsampled from a high resolution of 640x640 to a low resolution of 20x20. The different scales make it easier to focus on different sized objects. The arrows show how the information flows through the network to the three output layers, it has a bottom-to-top pathway as well as a top-to-bottom and another bottom-to-top pathway. These are called feature pyramids and were introduced in [32]. The upper layers contain more semantic information, but due to the lower resolution lack the precise localization. By allowing information to flow between the different scales, the semantic information can be combined with the more precise localization from the higher resolution, resulting in better object detection.

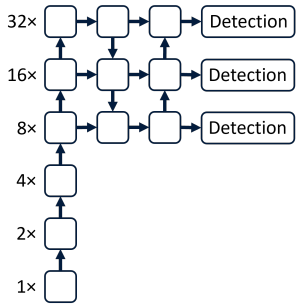


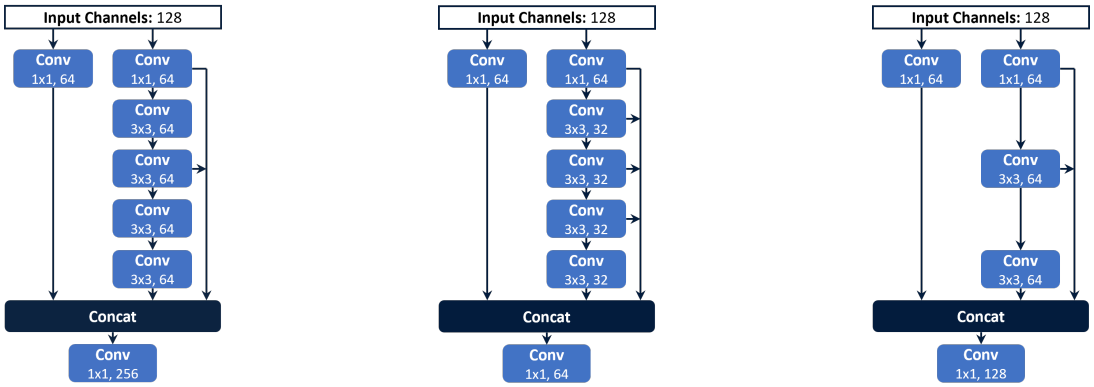
Figure 2.5: Simplified overview of the YOLOv7 architecture showing the flow of information. Numbers on the left indicate the scale: an input image of 640x640 will be reduced to 20x20 at the top (640/32 = 20).

ELAN

Most of the convolutional layers are built up from ELAN (Efficient Layer Aggregation Networks) blocks [33]. Stacking more and more layers in a neural network leads to reduced accuracy increase, and after a point, it will actually reduce the overall accuracy. ELAN is designed to solve this problem by reducing the length of the shortest gradient path. Figure 2.6 shows different configurations of an ELAN block. ELAN 2-4 and ELAN 1-4 are used in YOLOv7, while the smaller ELAN 1-2 is used in YOLOv7-tiny.

Detection Head

YOLOv7(-tiny) contains a detection head at three different scales (8x, 16x, 32x). These heads consist of two convolution layers, where the last layer has 255 output channels. Thus, in the case of a 640x640 input image, the output sizes are: 80x80x255 (8x scale), 40x40x255 (16x scale) and 20x20x255 (32x scale). The next subsection will explain how the output is interpreted to get to bounding boxes and classifications of objects in the image.



(a) ELAN 2-4 (b) ELAN 1-4 (c) ELAN 1-2

Figure 2.6: Different ELAN block configurations. Numbers under 'Conv' indicate kernel size and number of output channels.

2.2.2 Output

As mentioned in the previous subsection, the output of YOLOv7(-tiny) consists of three tensors with sizes: 80x80x255, 40x40x255 and 20x20x255. For now, only 20x20x255 will be considered. The first two dimensions can be interpreted as a grid on the original input image. For each cell in this 20x20 grid three bounding boxes are predicted. Each of these bounding boxes is characterized by 85 values (three bounding boxes: $3\text{x}85 = 255$). The first four of these values determine the location of the bounding box, the fifth is the object confidence score, and the last 80 values are the class probabilities. Thus, if a different dataset is used, with a different number of classes, the number of output channels must change to match.

At each scale three anchors are defined, that are boxes of a specific width and height. Instead of predicting the width, height and position of the bounding boxes, YOLOv7 predicts how the anchors should be scaled and translated to achieve the optimal bounding box. Figure 2.8 shows an example of three anchors centered inside a cell. These anchors are identical for all cells at the same scale. The predicted values (t_x, t_y, t_w, t_h) can then be transformed into the actual bounding box with the following equations:

$$b_x = 2 \cdot \sigma(t_x) - 0.5 + c_x \tag{2.10}$$

$$b_y = 2 \cdot \sigma(t_y) - 0.5 + c_y \tag{2.11}$$

$$b_w = \big(2 \cdot \sigma(t_w)\big)^2 \cdot p_w \tag{2.12}$$

$$b_h = \big(2 \cdot \sigma(t_h)\big)^2 \cdot p_h \tag{2.13}$$

Here b_x and b_y give the center position of the bounding box. c_x and c_y are the offsets of this cell from the top left of the grid. For the third cell of the left at the top row, $c_x = 2$ and $c_y = 0$. The width and height of the bounding box, b_w and b_h, are based on the width and height of the anchor, p_w and p_h. Finally, all these values must be multiplied by the scale, also called stride, to go from the grid to input image position. The scale is simply the number of pixels of the input image that are represented in one cell.

The 80 class probabilities for each bounding box can be interpreted in exactly the same way as with a standard classification network. The bounding box will be assigned to the class with the highest probability.

Now with three predicted bounding boxes at each cell, the total number of predictions is 25200 ($(20 \times 20 + 40 \times 40 + 80 \times 80) \times 3$). Clearly, most of these bounding boxes will not actually contain an object. That is why for each bounding box an object confidence score is predicted. As the name implies, the confidence score indicates how confident the network is that this bounding box contains an object. Multiplying the object confidence score with the probability of the assigned class, gives the overall confidence score or the estimated probability that an object with this specific class is present in the bounding box.

Non-maximum suppression (NMS) is used to remove all low-confidence predictions and predictions with high overlap. First, all predictions with an overall confidence score below a certain threshold (typically 0.25) are removed. Next, for all bounding boxes that have an IoU with another box above a threshold (0.65 for YOLOv7), the lower confidence prediction is removed. This last step is repeated till there are no bounding boxes with an overlap above the threshold. Thus, NMS takes in 25200 predictions and will typically only return a handful (e.g., two in the case of Figure 2.2a).

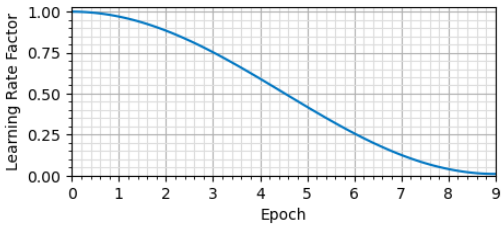


Figure 2.7: Cosine annealing.

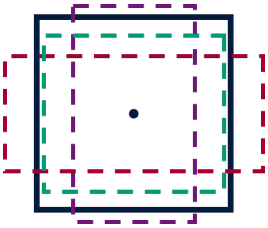


Figure 2.8: Anchor boxes (dashed) centered in cell (solid black).

2.2.3 Training

Data Augmentation

YOLOv7 is trained and evaluated on the MS COCO dataset. Different types of data augmentation are used to avoid overfitting. Firstly, Mosaic augmentation (introduced by YOLOv4 [34]) places 4 or 9 images side by side into one combined input image. Then MixUp [35] and Cutout [36] are applied with a probability of 15%. MixUp combines two images (with Mosaic already applied) into one using a weighted average of both images. Cutout randomly masks out square sections of the image. Next, 50% of the images is flipped left to right. Finally, some noise is applied to the hsv color values.

Loss

The loss of YOLOv7 consists of three parts: box, objectness and classification loss. The box loss is the CIoU loss [37] between the target bounding box and the prediction. The CIoU loss is similar to IoU but also penalizes the distance between the center points of both boxes and the deviation of the aspect ratios. The objectness score should be zero if there is no object in the bounding box, one if it exactly fits the target, and have the same value as the IoU in case of partial overlap. The objectness loss is then calculated as the cross-entropy between this target value and the actual predicted score. The classification loss is also implemented as a cross-entropy loss between the predicted distribution and the actual class.

Evaluation

The main evaluation metric for MS COCO is the AP (abbreviated from $\text{mAP}_{50:95}$). A detailed explanation of how the AP is calculated is given in Subsection 2.1.2. For YOLOv7 the main evaluation metric, which is used to determine if the network accuracy is increasing, is called fitness:

$$fitness = 0.1 \cdot AP_{50} + 0.9 \cdot AP \tag{2.14}$$

This metric is also used in the rest of this thesis to evaluate the compressed models.

Learning Rate Schedule

The learning rate is slowly decreased during training using cosine annealing. Figure 2.7 shows the learning rate factor going from 1.0 to 0.01 in 10 epochs. The actual learning rate for each epoch is the start learning rate multiplied by the learning rate factor.

2.3 Compressing Neural Networks

Compression techniques can be categorized into four different types. Section 2.3.1 will introduce these categories, followed by an overview of each category. The evaluation metrics to compare different compression methods are explained in Section 2.3.2. Section 2.3.3 will give a brief overview of the different datasets and networks commonly used for benchmarking. These datasets and networks are needed for comparing and selecting compression methods.

2.3.1 Compression Techniques

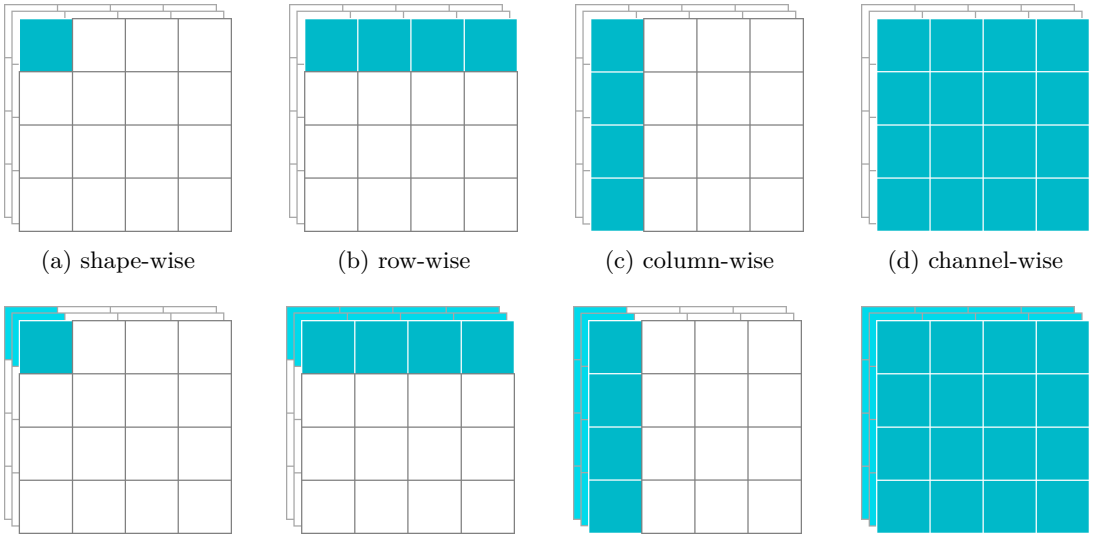
Different papers use different ways to categorize compression techniques, or use different subcategories [5, 9, 10]. In this thesis they are categorized into: (i) pruning, (ii) quantization, (iii) tensor decomposition and (iv) compact architectures. Tensor decomposition includes the decomposition of both fully-connected and convolutional layers.

Pruning

Network pruning is based on the Minimal Description Length (MDL) principle, which states that the best model for describing a dataset is the one that leads to the highest compression [5]. Together with the observation that large networks only need a fraction of their parameters for accurate prediction [11], pruning methods try to find unimportant parameters and remove them from the network or set them to zero. Besides compression, pruning also has a regularization effect on the network [38].

A common criterion for pruning a parameter is magnitude-based, pruning parameters whose weights are below a certain threshold. This is generally used in combination with L_2 or L_1 regularization, which will force parameters that have a small impact on the network's performance close to zero [5]. The L_2 norm mainly pushes the value of larger weights down, while the L_1 norm can achieve some sparsity. The L_0 norm would induce even more sparsity, but since it is not differentiable, cannot be used for neural network training with gradient descent [39].

Pruning can be categorized into structured and unstructured methods. Unstructured methods will prune individual weights, while structured methods will prune complete neurons or entire



(a) shape-wise (b) row-wise (c) column-wise (d) channel-wise

(e) stack shape-wise (f) stack row-wise (g) stack column-wise (h) filter-wise

Figure 2.9: Types of structured sparsity in 3D filters. Blue squares indicate weights to be pruned. Sparsity types (a)-(d) can also be used for fully-connected layers. Note: each convolutional layer includes many filters, but only one is shown for simplicity.

filters. Unstructured pruning requires specific software to take full advantage of the sparsity, while structured pruning completely removes certain parts of the network, resulting in a more compact architecture, with fewer FLOPs, which can be run directly without the need for specialized software [40]. Figure 2.9 shows the different kind of structures that can be used for sparsity in convolutional layers. Figure 2.9a shows no structure at all and 2.9h is fully structured. Figure 2.9b-2.9g could be classified as semi-structured, which does induce some structure but still requires additional optimization. Sparsity types 2.9a-2.9d can also be applied to the 2D weights of fully-connected layers. In that case, 2.9b-2.9d would be considered fully structured, which directly reduces the model size.

Pruning methods using structured sparsity can lead to a more compact architecture, but not necessarily to a smaller file size, since they usually do not prune the individual weights in the remaining layers. Therefore, methods using structured sparsity may achieve less compression but end up with a model that uses fewer FLOPs and thus runs faster.

Quantization

Quantization reduces the number of bits used to represent the value of each parameter. [41] shows that neural networks are resistant to certain amounts of low precision. There are a lot of different quantization methods. The standard way is to reduce the precision used for each parameter. For example, when an 8-bit format is used, instead of the standard 32-bit format, the model size is reduced by a factor of four. In the extreme case the network can be binarized, using only one bit for each parameter [42].

The previous approach uses linear quantization, but the weights of neural networks are not uniformly distributed [43]. One way to use nonlinear quantization is k-means clustering, which can be done during [11] or after [44] training. When one of the clusters is located at zero, this effectively prunes all parameters assigned to this cluster.

An important insight of [45] is the fact that pruning and quantization can compress neural networks without interfering with each other. Combining the two techniques can therefore lead to higher compression with little to no extra accuracy loss.

Tensor Decomposition

Tensor decomposition is a technique to approximate the full weight matrix of a fully-connected layer or the filter kernels of a convolutional layer with a low-rank approximation. Tensor decomposition methods include Truncated Singular Value Decomposition [46], Tucker Decomposition [47], Canonical Polyadic Decomposition [48] and Tensor Train Decomposition [49]. There are several disadvantages to this approach [40]. Firstly, it is not always obvious which rank should be used. Secondly, the decomposition operation is computationally expensive. And finally, the factorized neural network converges slower, meaning that extensive retraining is required. However, most compression methods require fine-tuning and a lot of extra computation. Which is usually not a problem since this can be done using powerful computers.

Compact Architectures

The above-mentioned techniques change the original network to reduce its size. Another common technique is the use of compact architectures. These networks are not changed during or after training but are specifically designed to be compact. The most well-known networks are MobileNet [14] and SqueezeNet [50]. Another option is using knowledge distillation. With this approach a large network, or an ensemble of large networks, which acts as teacher is fully trained without any compression. Then, a much smaller student network is trained using both the actual dataset the teacher was trained on, as well as the dark knowledge [51] (softmax output) of the teacher. The extra knowledge from the teacher allows the student to minimize the accuracy loss [52].

Typically, the softmax output of both the teacher and student are softened using temperature T before computing the loss [51]. The output is then calculated as:

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{2.15}$$

Where inputs z_i are converted to q_i. Note that if $T = 1$, this results in the standard softmax function.

2.3.2 Evaluation Metrics

There are a lot of possible evaluation metrics, but most metrics are related to accuracy, number of parameters and number of floating-point operations (FLOPs) [5, 9, 10]. Since the comparison of compression methods is done using similar networks and datasets, it makes sense to use the relative metrics accuracy loss and compression ratio.

The accuracy loss can be used as a constraint when searching for the right compression method. In certain applications an accuracy loss of up to 10% might be acceptable, while in other applications a loss of just 1% may cause significant problems [53].

It is important to note that the number of parameters is not linearly correlated with the number of FLOPs [40]. Convolutional filters do not need many parameters, but depending on the image size, will require a lot of FLOPs. And the opposite is true for fully-connected layers. These layers take up a lot of the parameters but contribute little to the number of FLOPs.

Depending on the way the parameters are saved, the model size saved on disk might not be linearly correlated with the number of parameters. This is obvious when the parameters are saved with different precision but could also happen when a different data structure is used to manage sparse tensors. In the latter case, the differences are usually negligible, but when available both the number of parameters and the model size should be used for comparison between different methods.

In the literature it is common to calculate the compression ratio for the number of FLOPs (or parameters) as the original number of FLOPs divided by the number of FLOPs in the compressed network. The higher the compression ratio, the fewer parameters an increase in compression ratio will prune. For example, the difference between a compression ratio of 10 and 20 is 5% of the parameters, but the difference between a compression ratio of 100 and 110 is below 0.1%. Given this nonlinearity, a better way to represent compression ratios seems to be the percentage of parameters or FLOPs remaining after compression. In this thesis this will be referred to as the FLOPs Pruning Ratio (FPR) and Parameter Pruning Ratio (PPR). The way it is defined, a FPR of 100% means no compression, so a lower FPR and PPR is better.

2.3.3 Benchmark Datasets and Networks

To make fair comparisons between different compression methods, as many external factors as possible should remain constant. The most important factors are the dataset and the network that are used to evaluate the compression method. For example, using a large network will generally allow for more compression compared to a small network for the same task. Similarly, a simple binary classification task will need a smaller network than a complicated classification task with a thousand classes, and thus allow for more compression.

Fortunately, there are several common datasets and networks that are often used for benchmarking. In this section we discuss the ImageNet [54] and CIFAR-10 [55] datasets. Besides these datasets, we will briefly introduce the following networks: ResNet-50 and ResNet-56 [15]. Typically, ResNet-50 is used for ImageNet and ResNet-56 for CIFAR-10.

ImageNet + ResNet-50

ImageNet is a large dataset of over 14 million images with almost 22 thousand classes. A popular subset comes from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). It is sometimes called ImageNet-1k or ILSVRC2017, but given its popularity is usually just referred to as ImageNet.

This subset includes a thousand classes and is already split up in training, validation and test sets. It contains roughly 1.3 million training, 50 thousand validation and 100 thousand test images. The size of the images varies, but it is common to resize all images to 224 by 224 pixels. Currently, the best accuracy achieved on ImageNet-1k (from now on referred to as ImageNet) is 91.1% [56].

The ResNet architecture was introduced in 2015 as a solution to the degradation problem. Namely, the problem that increasing the depth of a neural network only increases the accuracy up to a point, after which it degrades rapidly [15]. ResNets consist of residual building blocks. The input of each block is directly added to the output of one or more convolutional layers (see Figure 2.10), this way each block can learn a residual function

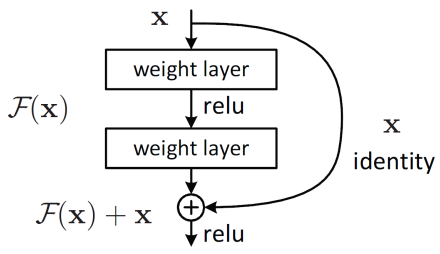


Figure 2.10: ResNet building block [15].

$\mathcal{F}(x)$. It is shown that this solves the degradation problem, since by pushing the weights of certain layers to zero, it easily mimics a smaller network if necessary. While at the same time more building blocks allow for more residual refinement of the network output, resulting in better accuracy.

The number in the name indicates the number of layers, both fully-connected and convolutional layers, in the neural network. ResNet-50 consists of a convolutional layer, 16 building blocks with 3 convolutional layers each, and a fully-connected layer outputting 1000 probabilities for the ImageNet classes. ResNet-50 achieves an accuracy of 75.3% on ImageNet.

CIFAR-10 + ResNet-56

CIFAR-10 is a dataset of 50 thousand training and 10 thousand test images. As the name implies, the dataset contains 10 classes. For each class there are 5 thousand training images and a thousand test images. All images have the same size of 32 by 32 pixels. Note that the number of classes and image size are much smaller compared to ImageNet. A similar neural network will require significantly less FLOPs for CIFAR-10 than ImageNet. Currently, the best accuracy achieved on CIFAR-10 is 99.9% [57].

ResNet-56 consists of a convolutional layer, 27 building blocks with 2 convolutional layers each, and a fully-connected layer outputting 10 probabilities for the CIFAR-10 classes. ResNet-56 achieves an accuracy of 88.8% on CIFAR-10.

For both ImageNet and CIFAR-10, ResNets do not achieve state-of-the-art accuracy. This is to be expected, given that ResNets were introduced in 2015. By using a benchmark model that older compression methods already reported results for, it is easier to compare methods. Using the latest models, requires rerunning older compression methods for comparison. This only works up to a certain point. For example, MNIST and LeNet-5 [58] have been used for a long time as benchmark, and are still sometimes used, but LeNet-5 has been compressed to less 0.5% of its weights without any loss of accuracy [12], making any further improvements difficult to quantify.

3 Methods

In this chapter two compression methods are selected for testing on YOLOv7-tiny. Section 3.1 will answer the question: *What are the best networks for comparing compression methods?* This is followed by a compilation of best performing compression methods: *What is the state-of-the-art in neural network compression?* From this list, two methods are selected, which will narrow down *which compression method is best suited for compressing YOLOv7?* The selected methods are explained in Sections 3.2-3.3.

3.1 Compression Search

For compressing YOLOv7, a list of the state-of-the-art compression methods must be compiled. Fortunately, there are several surveys categorizing and rating compression methods. The largest comparison of compression methods is given in [59]. This comparison is used as a starting point for finding the best methods. The authors categorize 150 methods and count how often each network and dataset is used to report the compression results. The most used networks and datasets are: ResNet-56 on CIFAR-10, AlexNet and ResNet-50 on ImageNet, and LeNet-5 on MNIST. As mentioned in Chapter 2, LeNet-5 is no longer useful as benchmark. Both LeNet-5 and AlexNet are relatively old and small networks that are less representative of bigger networks like YOLOv7.

So, *what are the best networks for comparing compression methods?* Based on how often these networks are used as benchmark, ResNet-50 (on ImageNet) and ResNet-56 (on CIFAR-10) are the best networks to compare and rank compression methods. Note that these networks and datasets are introduced in Section 2.3.3.

The latest methods reported in [59] are from 2020. It is likely that since then new methods have been proposed with better performance. Therefore, a search has been performed for papers from after 2020. By only including the papers that report on ResNet-50 on ImageNet and ResNet-56 on CIFAR-10, the search is narrowed down considerably.

Table 3.1 shows the result of this search. The FLOPs pruning ratio (FPR) and parameter pruning ratio (PPR) are the remaining percentage of FLOPs and parameters, respectively. To limit this table to the best performing compression methods, only those with a FPR below 30% for ResNet-56 or a FPR below 40% for ResNet-50 are included. If a method achieves a FPR below this threshold on one, but not the other, network, the results are still given for both networks. For most methods PPR and FPR are close together. Since the goal is to speed up a neural network, the number of FLOPs is more important than the number of parameters.

From this table it can be concluded that FPFS outperforms all other methods on ResNet-56. PCA-Pruner, CONPLSF, HRel-1, NNCS and GBIP are quite similar in compression, with NNCS outperforming them on accuracy loss. PKSMIO and KSE are clearly at the bottom based on their compression ratio but do limit the accuracy loss or even increase accuracy. For ResNet-50 on ImageNet, NNCS has slightly better compression and KSE has slightly better accuracy. PKSMIO, GBIP and FPFS also have similar compression performance, with GBIP having a slight edge on accuracy. HRel-1 is significantly outperformed by the other methods on ResNet-50.

Although the compression ratio and accuracy loss of a method are important, the ease

Table 3.1: Reported results from the best performing compression methods for ResNet-56 on CIFAR-10 and ResNet-50 on ImageNet. Accuracy Loss is the percentage point reduction in accuracy of the pruned network. Negative accuracy loss indicates an increased accuracy after pruning. FPR and PPR refers to FLOPs Pruning Ratio and Parameter Pruning Ratio, respectively. These pruning ratios give the percentage of FLOPs or parameters remaining after pruning. A lower pruning ratio is better. Only compression methods with a FPR below 30% for ResNet-56 or a FPR below 40% for ResNet-50 are included in this table. For completeness, the results of methods with a FPR below this threshold for only one network are still reported for both networks. This table helps answer the question: *What is the state-of-the-art in neural network compression?*

Model	**Method**	**Accuracy Loss**	**FPR**	**PPR**
ResNet-56 (CIFAR-10)	FPFS [13]	0.7	11.2	12.9
	PCA-Pruner [60]	1.05	11.2	18.8
	CONPLSF [61]	1.35	21.0	21.9
	HRel-1 [62]	1.1	23.1	22.2
	NNCS [63]	0.21	23.8	13.7
	GBIP [64]	0.38	26.6	29.6
	PKSMIO [65]	-0.32	35.8	36.4
	KSE [66]	0.15	40.0	41.7
ResNet-50 (ImageNet)	NNCS	1.24	21.1	24.4
	KSE	0.84	21.3	34.5
	PKSMIO	1.48	33.9	27.8
	GBIP	0.47	36.7	44.6
	FPFS	0.96	39.7	-
	HRel-1	0.68	51.3	51.8

Table 3.2: Number of training epochs that each method uses to go from a pretrained to a pruned, finetuned network. '50 per layer' means that the network layers are pruned sequentially with 50 epochs of finetuning after each layer. * *These numbers are not explicitly mentioned in papers but are estimated.*

Method	**Training Epochs**	
	CIFAR-10	*ImageNet*
FPFS	50 per layer	100 per layer
PCA-Pruner	50 per layer	-
CONPLSF	260	-
HRel-1	100	33
NNCS	450*	-
GBIP	30	20
KSE	200	21
PKSMIO	150*	-

of implementation is also of relevance. Specifically, the time it takes to go from a pretrained model to a pruned and finetuned model should be within certain bounds. Table 3.2 shows the number of training epochs each methods requires to achieve a pruned and finetuned model. Regarding size, the dataset used for training YOLOv7, MS COCO, is more similar to ImageNet than CIFAR-10. Unfortunately, not all papers report training epochs on ImageNet. Since it can take up to two hours on a simple GPU to train YOLOv7 for one epoch, most methods are not feasible. From this table, the three methods that seem feasible are HRel-1, GBIP and KSE.

GBIP outperforms KSE for ResNet-56 on CIFAR-10, and HRel-1 for ResNet-50 on ImageNet, which makes it an easy choice. The second chosen method is KSE. It significantly outperforms HRel-1 on FPR with similar accuracy loss on ResNet-50. It is outperformed by HRel-1 on ResNet-56, but in this case the difference in accuracy loss is clearly in favor of KSE.

Therefore, the answer to the question, *Which compression method is best suited for compressing YOLOv7?* is narrowed down to two methods: GBIP and KSE. The next subsections will describe the selected methods. The specific implementations of these methods for YOLOv7-tiny will be discussed in Chapter 4.

3.2 GBIP

Global balanced iterative pruning for efficient convolutional neural networks [64] introduces a compression method (GBIP) based on a simple pruning strategy but with a more advanced approach to recovering accuracy. It prunes entire filters (see Figure 2.9h) from each convolutional layer. Removing filters reduces the number of output channels of this layer as well as the input channels of the next layer.

As criterion for pruning, the L_1 norm of the output feature maps is used. The use of output feature maps, means that GBIP requires a dataset to compress a network. By normalizing the L_1 norm for each layer, layers with large L_1 norms do not dominate the compression result. Rather the network is compressed more evenly, resulting in a *balanced* pruning of the network. This balanced pruning means that the difference in pruning ratios of FLOPs and parameters is small. The importance score m_n^l is calculated for all N_l output feature maps Y_n^l for all layers l:

$$m_n^l = \|Y_n^l\|_1 / \max\{\|Y_1^l\|_1, \|Y_2^l\|_1, ..., \|Y_{N_l}^l\|_1\} \tag{3.1}$$

It is not explicitly mentioned whether the output feature maps are pre- or post-activation. However, other compression methods typically use post-activation [67, 68, 69]. This makes sense, given that the post-activation feature maps contain the information that is passed on through the rest of the network. Therefore, the post-activation feature maps are used for computing the importance score.

The pruning threshold is determined by the mean of the importance scores for each layer and a global pruning factor k:

$$m_p^l = k \frac{1}{N} \sum_{n=1}^{N} m_n^l, \quad \text{with } k \in (0, 1) \tag{3.2}$$

For all feature maps where m_n^l is below m_p^l the corresponding filters are removed. The next layer now has a reduced number of input channels and thus its weights can also be partially pruned.

As the name indicates, an *iterative* pruning schedule is used, where the network is trained for ten epochs after each pruning step to partially restore its accuracy. This increases the

control over the pruning process. If the accuracy stays high, a next pruning step can be performed. But if the accuracy starts to deteriorate too quickly, the network might be compressed to its limit. The authors give no rationale for the number of pruning steps they use in their experiments, but depending on the dataset there are a total of 2 or 3 pruning steps.

Using the L_1 norm for pruning filter is common, but the authors of GBIP also focus on the finetuning stage. They add three ways to improve the accuracy recovery: knowledge transfer from the (i) output and (ii) intermediate features of the original network, as well as an (iii) adversarial network to discriminate between the two networks.

3.2.1 Output Transfer

The output of the original model (teacher) is used to guide the training of the compressed model (student). As mentioned in Subsection 2.3.1, the knowledge of the teacher can be transferred to the student by adding the KL divergence between the output of the teacher, $f_T(x)$, and the student, $f_S(x)$, to the student loss function. In this case, a temperature T is used to soften the outputs. To keep the magnitude of the KL loss independent of the temperature, the loss is multiplied by T^2. Finally, the output transfer loss ($\mathcal{L}_{OT}$) is a weighted sum of the cross-entropy between $f_S(x)$ and hard targets, and the KL loss ($\mathcal{L}_{KL}$):

$$p(x) = F_{softmax}(f_S(x)/T) \tag{3.3}$$

$$q(x) = F_{softmax}(f_T(x)/T) \tag{3.4}$$

$$D_{KL}(p \parallel q) = \sum_{i=1}^{n} [p(x)\log(p(x)) - p(x)\log(q(x))] \tag{3.5}$$

$$\mathcal{L}_{KL}(W_S) = T^2 D_{KL}(p \parallel q) \tag{3.6}$$

$$\mathcal{L}_{OT}(W_S) = \alpha \mathcal{L}_{KL}(W_S) + (1-\alpha)\mathcal{L}_{CE}(W_S) \tag{3.7}$$

3.2.2 Attention Transfer

Not only the output, but also the intermediate features of the teacher can be used to guide the training of the compressed student. The implementation in GBIP is taken from [70]. The activations M of three different layers in the network are used to calculate spatial attention maps A^l. These maps are a measure of how much attention the network is paying to each pixel of the feature map. This way the student can be taught to focus on the same pixel locations as the teacher. The spatial attention maps are normalized by their L_2 norm. The attention transfer loss $\mathcal{L}_{AT}$ is given by the L_2 norm of the difference between the normalized attention maps:

$$A^l(M^{ab}) = \frac{1}{N}\sum_{n=1}^{N}(M_n^{ab})^2 \tag{3.8}$$

$$\mathcal{L}_{AT}(W_S) = \sum_{l=1}^{3}\left\| \frac{A_S^l}{\|A_S^l\|_2} - \frac{A_T^l}{\|A_T^l\|_2} \right\|_2 \tag{3.9}$$

3.2.3 Adversarial Game

To further converge the output of the student to that of the teacher, a discriminator network is created to differentiate between the two networks. This network has three fully-connected

layers with 128-256-128 neurons. The task of this network is to get better at distinguishing between teacher and student output, while the student network tries to fool the discriminator. The idea is taken from [71], where a simple fully-connected network is also used to distinguish between an original and a compressed network. The goal of the discriminator is to output a value close to one if the input came from the teacher and close to zero if it came from the student. It does this by minimizing the following loss function:

$$\begin{aligned}\mathcal{L}_G(W_G) = E_{f_T(x)\sim p_T(x)}[\log(1 - G(f_T(x, W_T), W_G))] \\ + E_{f_S(x)\sim p_S(x)}[\log(G(f_S(x, W_S), W_G))]\end{aligned} \tag{3.10}$$

while the discriminator is being optimized to predict a zero for student input, the student is simultaneously updated to fool the discriminator and get it to output a value closer to one, with the adversarial game loss $\mathcal{L}_{AG}$:

$$\mathcal{L}_{AG}(W_S) = E_{f_S(x)\sim p_S(x)}[\log(1 - G(f_S(x, W_S), W_G))] \tag{3.11}$$

Combining these losses, gives the following total loss function for the compressed network:

$$\mathcal{L}_S(W_S) = \mathcal{L}_{AG}(W_S) + \mathcal{L}_{AT}(W_S) + \mathcal{L}_{OT}(W_S) \tag{3.12}$$

3.2.4 Results

The authors test GBIP on several networks and datasets. From this, it is clear that $k = 0.5$ is the maximum for smaller networks like VGG-16 and VGG-19, as well as for larger networks on more complex datasets (CIFAR-100, ImageNet). For a large network on a smaller dataset, like ResNet-110 on CIFAR-10, k can go up to 0.7 without accuracy loss.

They also assess the effect of the three finetuning techniques. All three show improvements with output transfer (OT) having the largest effect, followed by the adversarial game (AG) and attention transfer (AT). For ResNet-18 on ImageNet, the authors report the best accuracy improvement of 1.24% using all three techniques. While the accuracy improvement for VGG-16 on CIFAR-10 and ResNet-56 on CIFAR-100 is only 0.20% and 0.24%, respectively. The difference between these accuracy improvements might be due to the initial accuracy loss of the baseline without any finetuning techniques. For these last two networks, the baseline accuracy loss is very small (0.42%) for ResNet-56 and negative (-0.34%) for VGG-16, leaving less room for improvement than ResNet-18 with 1.90% baseline accuracy loss.

3.3 KSE

Exploiting Kernel Sparsity and Entropy for Interpretable CNN Compression [66] introduces a compression method (KSE) focusing on the channels of the input feature maps and its corresponding 2D kernels. Instead of pruning, this method uses clustering of the 2D kernels into a few clusters to compress the network. It introduces an indicator that uses both the sparsity and entropy of a kernel. Based on this indicator the number of clusters required for each input feature map is calculated.

3.3.1 Kernel Sparsity

The convolution operation can be formulated as:

$$Y_n = \sum_{c=1}^{C} W_{n,c} * X_c \tag{3.13}$$

with C channels in the input feature map and N channels in the output. For each channel in the input, X_c, the matching 2D kernels are given by $\{W_{n,c}\}_{n=1}^{N}$. Instead of using the sparsity of X_c as pruning indicator, the sparsity of its corresponding 2D kernels is used. In the paper, the authors verify the relationship between the sparsity of the input and that of the weights. This makes KSE a data-free compression method since it only uses the weights of the network without having to run a dataset through it.

The sparsity for channel c of the input feature map is defined as:

$$s_c = \sum_{n=1}^{N} \|W_{n,c}\|_1 \tag{3.14}$$

Clearly, s_c is low for sparse kernels, where the weights are small.

3.3.2 Kernel Entropy

If an input feature map contains a lot of information, its 2D kernels are less suitable for pruning. The amount of information increases if the 2D kernels are very diverse. The authors propose kernel entropy as a measure of this information.

First, a density metric dm is calculated for all 2D kernels based on the distance between each kernel and its k-nearest neighbors:

$$dm(W_{i,c}) = \sum_{j=1}^{N} A_{C_{i,j}} \tag{3.15}$$

$$A_{C_{i,j}} = \begin{cases} \|W_{i,c} - W_{j,c}\|_2 & \text{if } W_{j,c} \text{ is among k nearest neighbors of } W_{i,c} \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

Larger values for $dm(W_{i,c})$ mean its closest neighbors are far away. Large differences between kernels also result in large differences in the convolution of these kernels with the input feature map. Thus, if many kernels have a high dm, the input feature map contains a lot of information.

Based on this density metric, the proposed kernel entropy is defined as:

$$e_c = -\sum_{i=1}^{N} \frac{dm(W_{i,c})}{d_c} log_2 \frac{dm(W_{i,c})}{d_c}, \quad \text{with} \quad d_c = \sum_{i=1}^{N} dm(W_{i,c}) \tag{3.17}$$

If the kernels are very diverse, the kernel entropy will be lower.

3.3.3 Kernel Clustering

The kernel sparsity and entropy (KSE) indicator combines the two metrics introduced above:

$$v_c = \sqrt{\frac{s_c}{1 + \alpha e_c}} \tag{3.18}$$

With balancing factor α set to 1 for all experiments. As mentioned above, a lower kernel sparsity refers to a higher sparseness, while a lower entropy refers to a more diverse distribution of kernels. Therefore, a lower v_c should indicate higher compression. Based on v_c the number of kernels for the c-th input feature map is calculated:

$$q_c = \begin{cases} 0 & \text{if} \quad \lfloor v_c G \rfloor = 0 \\ N & \text{if} \quad \lceil v_c G \rceil = G \\ \left\lceil \frac{N}{2^{G - \lceil v_c G \rceil + T}} \right\rceil & \text{otherwise} \end{cases} \tag{3.19}$$

Where G and T are the hyperparameters controlling granularity and compression ratio, respectively. If $G = 2$, q_c is limited to 0 or N. Larger values for G allow for finer grained compression. T reduces the number of clusters by a factor 2^T for the third case in the equation above.

In the case of $q_c = 0$, the entire input feature map can be ignored. And if $q_c = N$, no compression is applied. For the values in between 0 and N, all the original N kernels are assigned to one of the q_c clusters and its weights replaced by the centroid of this cluster. The centroids are denoted as $\{B_{i,c}\}_{i=1}^{q_c}$, while the index set $\{I_{n,c} \in \{1, 2, ..., q_c\}\}_{n=1}^{N}$ links each kernel to one of the q_c kernels. In the finetuning stage, only the cluster centroids are updated.

To make full use of the induced sparsity, the 3D convolutions are split into 2D convolutions. Using the notation from above, q_c 2D activation maps are generated: $Z_{i,c} = B_{i,c} * X_c$. These are then combined into the n-th output feature map:

$$Y_n = \sum_{c=1}^{C} Z_{I_{n,c},c} \tag{3.20}$$

3.3.4 Results

From the paper, one would assume the authors use Equation 3.20 to speed up the network, but from the published code it is clear that this is not the case. Instead, a full weight matrix is reconstructed from the clusters. Only in the case of $q_c = 0$, when there are no clusters, the size of the weight matrix is reduced. This means that the actual reduction in FLOPs is minimal. The fact that an unoptimized implementation is used, is not mentioned in the paper itself. They do, however, refer to the reduction in FLOPs as 'theoretical' exactly once.

They test KSE with ResNet-56, DenseNet-40 and DenseNet-100 on CIFAR-10, as well as ResNet-50 on ImageNet. For CIFAR-10, T is set to 0, while for ImageNet it is set to 1. In all cases, the best compression is achieved with $G = 5$ or 6.

For ResNet-50 on ImageNet the reduction in FLOPs is about 1.5 times the reduction in parameters, while for CIFAR-10 these are almost identical. And the accuracy loss is slightly higher for ImageNet, though still below 1%. This might be due to the different setting for T but is not mentioned in the paper.

They also evaluate the effect of kernel sparsity and entropy. For ResNet-56 on ImageNet, they show that using both kernel sparsity and kernel entropy outperform using either one of these alone.

This chapter explained the search for compression methods, showing the state-of-the-art in Table 3.1, which methods have been selected and how these selected methods work. Specifically, it answered the question: *What are the best networks for comparing compression methods?* Namely, ResNet-50 (on ImageNet) and ResNet-56 (on CIFAR-10). As well as answering: *What is the state-of-the-art in neural network compression?* and narrowing down *Which compression method is best suited for compressing YOLOv7?* to two methods: GBIP and KSE.

4 YOLOv7-tiny Experiments

The previous chapter introduced the two selected compression methods. This chapter will go over the specific implementation details of adapting these methods for compressing YOLOv7-tiny on MS COCO. It will also introduce the experiments that are run to determine the best performing method for YOLOv7.

First, Section 4.1 mentions the evaluation metrics and implementation details that are similar for both methods. Then, Section 4.2 and 4.3 will go into the specifics for GBIP and KSE, respectively.

4.1 General

The available implementation of both YOLOv7-tiny and KSE use PyTorch [72] as machine learning framework. To avoid unnecessary complexities, all the experiments also use PyTorch.

As much as possible, the hyperparameters chosen by the creators of YOLOv7 are kept the same for the compression experiments. For example, the batch size, learning rate scheduler and data augmentation, are not changed. This means a batch size of 64 and the use of cosine annealing (See Subsection 2.2.3). The starting learning rate is determined by training the network for different learning rates and check how fast the network converges with each.

Also from the YOLOv7 implementation, while training and evaluating the network the *fitness* metric is used. For the following experiments the fitness, AP and AP_{50} are reported. See Subsection 2.2.3 how these metrics are computed. To determine the amount of compression that is achieved, the parameter pruning ratio (PPR) and FLOPs pruning ratio (FPR) are used. These are explained in Section 2.3.2.

4.2 GBIP

4.2.1 Experiments

GBIP has only one hyperparameter influencing the amount of compression, pruning threshold factor k. The authors find the best results when using $k = 0.3 - 0.6$. Since YOLOv7-tiny is already relatively small, it makes sense to also test $k = 0.2$, so the following values will be tested: $k = \{0.2, 0.3, 0.4, 0.5, 0.6\}$. These experiments will run three pruning cycles, where each pruning cycle consists of a pruning step followed by 10 epochs of finetuning.

To increase the accuracy of the pruned network, GBIP introduces three finetuning techniques: output transfer (OT), attention transfer (AT) and adversarial game (AG). The authors do an ablation study to determine the efficacy of each of these techniques. The ablation study is done on three different networks, but only for image classification. Since the object detection task of YOLOv7(-tiny) is significantly different, and the fact that the effect measured in the ablation study differs considerably per network and dataset, before running the full experiments mentioned above an ablation study is done for YOLOv7-tiny. For this ablation study, only one pruning cycle will be tested with $k = 0.4$. That is, the network will be pruned once, followed by 10 epochs of finetuning. If any of the finetuning techniques does not improve the final accuracy, it can be omitted for the other experiments.

4.2.2 Implementation

The authors do not provide their own implementation online. Therefore, GBIP has been implemented from scratch in PyTorch. The explanation in the paper is entirely focused on an application for image classification. Some adjustments have been made to be able to use this method for object detection with YOLOv7-tiny on MS COCO.

The pruning step is identical, but the finetuning techniques have been (slightly) changed to fit the YOLOv7-tiny architecture. For attention transfer, only the location of the attention maps must be chosen. But given the different output structure of YOLOv7-tiny, compared to a typical classification network like ResNet-56, the output transfer and adversarial game requires more changes. The rest of this subsection will describe these changes.

Attention Transfer

The original implementation uses the activations of three layers in the network to calculate an attention transfer loss. No explanation is given for the exact positioning of these layers, but they are spaced out evenly through the network. Several combinations of layers have been tested by pruning with $k = 0.5$ and finetuning for one epoch. The results are shown in Table 4.1. Note that due to the limited finetuning, the accuracy increase is only an indication of the relative performance. So, which layers should be used for attention transfer? From the limited testing done, it seems that (i) it is better to include more layers than three, and (ii) that later layers ($[57, 65, 73]$) work better than earlier layers ($[0, 1]$). The best result (shown in bold) is obtained when using the output of all ELAN blocks to create the attention maps.

Table 4.1: Layer indices of attention maps tested for Attention Transfer. See Appendix A.1 for the YOLOv7-tiny architecture with indices. YOLOv7-tiny has been pruned with $k = 0.5$ and finetuned for one epoch.

Layer Indices	**Fitness Increase**
37, 40, 50	0.49
14, 21, 28	0.67
7, 14, 21, 28	0.79
57, 65, 73	1.39
0, 1	-0.12
7, 14, 21, 28, 47, 57, 65, 73	**1.77**
0, 1, 7, 14, 21, 28, 47, 57, 65, 73	1.71

Output Transfer

Given that the output of an image classification network is different from YOLOv7, the way the output transfer loss is calculated requires some changes. As explained in Subsection 3.2.1, to align the pruned network with the teacher, the KL divergence between the (softened) output of both networks is added to the loss function. This makes sense for an output of class probabilities, but the output of YOLOv7 not only contains class probabilities but also a bounding box and objectness score (see Subsection 2.2.2). To make full use of the output, the implemented transfer learning loss, $\mathcal{L}_{TL}$, consists of three parts: (i) a KL divergence loss between the class probabilities, (ii) a binary cross-entropy loss between the objectness scores, and (iii) an IoU loss between the bounding boxes of the student and the teacher.

Note that these three additional losses match with the three loss components of YOLOv7 (see Subsection 2.2.3). The KL divergence loss, $\mathcal{L}_{KL}(W_{S,class\_probs}$, is identical to Eq 3.3-3.6, while the IoU loss, $\mathcal{L}_{IoU}(W_{S,bbox})$, is similar to the IoU loss of YOLOv7. These six losses are combined, in the same way as Equation 3.7, to form the output transfer loss:

$$\mathcal{L}_{TL}(W_S) = \mathcal{L}_{KL}(W_{S,\text{class\_probs}}) + \mathcal{L}_{BCE}(W_{S,\text{objectness}}) + \mathcal{L}_{IoU}(W_{S,\text{bbox}}) \tag{4.1}$$

$$\mathcal{L}_{OT}(W_S) = \alpha\mathcal{L}_{TL}(W_S) + (1-\alpha)\mathcal{L}_{\text{YOLO}}(W_S) \tag{4.2}$$

Adversarial Game

As mentioned in Subsection 3.2.3, the discriminator for the image classification tasks is a 128-256-128 fully-connected network. This network takes as input a 1D tensor with class probabilities. Given the difference in output of YOLOv7, some changes to the discriminator network must be made. The output of YOLOv7 consists of three 3D tensors (see Subsection 2.2.3). For an input image of $[640 \times 640]$, the output is $([255 \times 80 \times 80], [255 \times 40 \times 40], [255 \times 20 \times 20])$. It is not directly obvious how the adversarial network should deal with this output. Therefore, three different types of architecture have been tested.

The first architecture uses max pooling to reduce the size of all 3D tensors to $[255 \times 1 \times 1]$. Then, the three tensors are concatenated and flattened, resulting in a 1D tensor with a length of 765. Finally, this 1D tensor is run through a 128-256-128 fully-connected network.

For the second network, max pooling is used on the second and third 3D tensors to create three tensors of size $[255 \times 20 \times 20]$. These tensors are concatenated and run through two convolutional layers, followed by one final fully-connected layer.

The last architecture most closely resembles the original adversarial network from the GBIP paper. First, each 3D tensor is reshaped from $[255 \times 80 \times 80]$ to $[19200 \times 85]$ and removing the bounding box information and objectness score gives $[19200 \times 80]$. This tensor contains 19200 probability distributions for each 80 classes in MS COCO. The original 128-256-128 fully-connected network can now be used on all 19200 probability distributions. The final output of the network is simply the average of all these 19200 runs.

Several tests have been done, slightly tweaking certain parts like kernel size for convolution and max pooling, number of neurons in fully-connected layers and total number of layers. Given these tests, and the fact that it is most similar to the original network, the fully-connected 128-256-128 network has been chosen.

In the final implementation, based on [71], alternately the student network is trained with adversarial loss for a period while the adversarial network stays fixed, and then the adversarial network is updated while the student network is trained without adversarial loss.

4.3 KSE

4.3.1 Experiments

KSE has two hyperparameters regulating compression: G and T. A higher value for G results in higher compression granularity, while higher values for T lead to higher compression ratios. To find the best values for these hyperparameters, the same values as used in the KSE paper ($G = \{3, 4, 5, 6\}, T = \{0, 1\}$) are tested on YOLOv7-tiny.

In contrast to GBIP, KSE compresses the network once, followed by several epochs of finetuning. In the case of YOLOv7-tiny, the accuracy reaches its maximum after about 15 epochs of finetuning. To be sure, all compressed networks will be finetuned for 20 epochs.

4.3.2 Implementation

Fortunately, the authors of KSE provide an implementation of their code online [LINK]. In contrast to GBIP, which requires significant changes for YOLOv7 due to the used finetuning

techniques, KSE does not use any special finetuning. This makes it relatively easy to adapt to YOLOv7-tiny.

Unfortunately, the published code contains an unoptimized implementation of their method. The output of each layer is not computed using Equation 3.20, but rather a weight matrix is created by matching the clusters $B_{i,c}$ and indices $I_{n,c}$ with $W_{n,c} = B_{I_{n,c},c}$ (See Subsection 3.3.3 for notation). This weight matrix has a size of $[N \times C' \times k \times k]$, where C' is the number of input channels with $q_c > 0$. This is almost identical to the size of the original weight matrix, since only a fraction of the input channels can be removed entirely.

For now, the optimization will be ignored, and revisited if the KSE outperforms GBIP. The same formula used by the authors of KSE to report the achieved acceleration will be used to compare with GBIP. This formula for one layer, rewritten to match FLOPs pruning ratio (FPR) definition, is given by:

$$FPR = \frac{\sum_c q_c}{NC} \tag{4.3}$$

Note that with the current unoptimized implementation, the actual FPR for each layer is simply:

$$FPR = \frac{C'}{C} \tag{4.4}$$

The theoretical PPR is the same as for the unoptimized implementation. Although a full weight matrix is constructed, the actual parameters that are saved and tracked are the cluster centroid.

Similar, to the original implementation, for YOLOv7-tiny the first and last layers are not compressed. The reason for this is that the first layer contains all information in only three input channels (RGB), so compressing this layer might remove too much information. Compression in the intermediate layers can be partly recovered in later layers, which is not the case for the last layer.

This chapter listed the experiments that will be run on YOLOv7-tiny, as well as the specific implementation details of GBIP and KSE. In the next chapter, the results of these experiments are given, based on which one method is selected to run on YOLOv7.

5 Selecting the Best Method

This chapter will begin by reporting the results of running GBIP and KSE on YOLOv7-tiny in Section 5.1. Based on these results, Section 5.2 will answer the question: *Which compression method is best suited for compressing YOLOv7?*

5.1 Compression Results

5.1.1 GBIP

An ablation study is performed to determine the efficacy of the three added finetuning techniques: attention transfer (AT), output transfer (OT), adversarial game (AG). The results are shown in Table 5.1. AT performs significantly better than the baseline. OT only manages a marginal increase in fitness, while AG shows no improvement at all. Combining AT and OT, which both increase fitness individually, does not result in an improvement over just using AT. Similarly, combining AT with AG does not increase the fitness. Given these results, the next experiments have been performed using only AT.

Table 5.1: Results of the ablation study on YOLOv7-tiny. All experiments are performed with $k = 0.6$. AT: Attention Transfer, OT: Output Transfer, AG: Adversarial Game. Fitness and AP are percentages. Best results are marked in bold.

AT	OT	AG	Fitness	AP	AP_{50}
-	-	-	31.4	29.7	47.1
✓	-	-	**33.9**	**32.2**	**49.9**
-	✓	-	32.4	30.7	47.9
-	-	✓	31.4	29.7	47.1
✓	✓	-	33.8	32.0	49.6
✓	-	✓	**33.9**	**32.2**	**49.9**

The results of running GBIP on YOLOv7-tiny for $k = \{0.2, 0.3, 0.4, 0.5, 0.6\}$ are shown in Table 7.1. There are two observations that can be made from this data. Firstly, the number of FLOPs decreases faster than the number of parameters. The authors of GBIP show that for VGG-16 and GoogLeNet on CIFAR-10 the FPR and PPR stay within about 5%pt of each other. For $k = 0.6$ there is a difference of 23.1%pt between FPR and PPR. The reason for this, is that the first layers, which contain the most FLOPs, are pruned slightly more than the rest of the network. For the networks tested by the authors, the FLOPs are more evenly distributed throughout the networks than is the case for YOLOv7-tiny.

Secondly, the FPR does not decrease linearly with increasing k. This is to be expected, given that the network is pruned multiple times. A smaller k will remove a small portion of channels in the first pruning cycle, and then another small portion of the remaining network in the next cycle. While larger values for k will remove a larger portion of channels the first time, and another large portion of an already smaller network the second time, thus compounding the compression effect.

Table 5.2: Results of GBIP experiments on YOLOv7-tiny. Fitness, AP, FLOPs Pruning Ratio (FPR) and Parameter Pruning Ratio (PPR) are given as percentages. Note $k = 0$ refers to the unpruned baseline.

k	Fitness	AP	AP_{50}	PPR	FPR
0	39.2	37.4	55.2	100	100
0.2	39.0	37.2	55.4	99.8	99.6
0.3	38.4	36.6	54.7	97.9	93.6
0.4	37.6	35.8	54.0	95.2	86.8
0.5	34.8	33.0	50.9	85.8	68.6
0.6	28.0	26.3	42.9	65.4	42.2

5.1.2 KSE

The results of running KSE on YOLOv7-tiny are shown in Table 5.3. For the FPR both the actual and theoretical values are given, while the actual PPR is identical to the theoretical values. The actual values refer to the used unoptimized implementation, while the theoretical values are computed using Equation 4.3 (see Subsection 4.3.2).

Regarding the actual FPR, if granularity G gets higher, the less reduction in FLOPs is achieved. This is expected, since with higher granularity, the number of clusters assigned to an input channel can be very small. As mentioned in Subsection 4.3.2, the actual FPR is based on the number of input channels that are completely ignored, that is $q_c = 0$. With low granularity, either no clusters or a large number of clusters is assigned, resulting in the removal of relatively unimportant input channels, which in the case of high granularity might have been assigned just a few clusters.

In general, increasing G and T increases the compression. It seems that the specific combination of G and T does not matter much for the resulting accuracy. For both $G = 5, T = 0$ and $G = 3, T = 1$ the FPR is about 70%, and their fitness is identical. The same can be seen for $G = 6, T = 0$ and $G = 4, T = 1$, where the FPR is around 60% and the difference in fitness is 0.6%pt.

Table 5.3: Results of KSE experiments on YOLOv7-tiny. Fitness, AP, PPR and FPR are percentages. Note that for the FPR both the actual and theoretical values are given, theoretical PPR is the same as the actual values (see Subsection 4.3.2).

G	T	Fitness	AP	AP_{50}	PPR	FPR	FPR
					theoretical		*actual*
-	-	39.2	37.4	55.2	100	100	100
3	0	36.9	35.0	53.4	88.1	67.4	92.9
4	0	36.8	35.0	53.2	80.7	55.7	96.7
5	0	36.0	34.2	52.4	70.7	46.3	97.9
6	0	34.8	33.0	51.2	62.3	39.6	98.2
3	1	36.0	34.2	52.4	69.9	54.7	92.9
4	1	35.4	33.6	51.8	59.4	40.4	96.7
5	1	34.1	32.3	50.3	51.2	32.2	97.9
6	1	31.8	30.0	47.8	44.8	27.4	98.2

5.2 Best Method

5.2.1 Comparing GBIP and KSE

From the results of compressing YOLOv7-tiny, the most relevant information is the reduction in fitness with respect to the FLOPs Pruning Ratio (FPR). This information is shown in

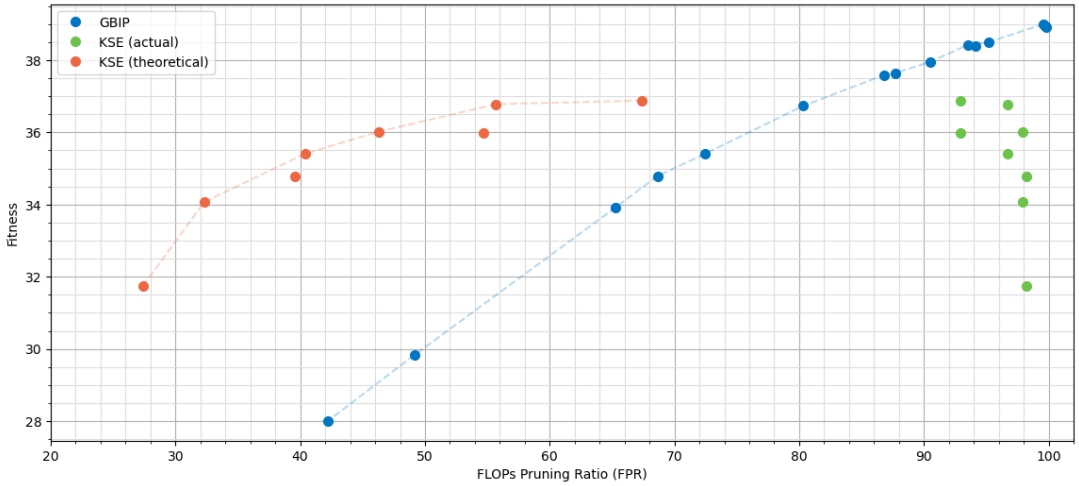


Figure 5.1: Fitness vs FPR for GBIP, KSE (theoretical) and KSE (actual). According to the KSE paper, the theoretical FLOPs should be possible with some optimization. KSE (actual) gives the current FPR of the authors' own implementation of KSE, without optimization. For GBIP the end result of each pruning cycle is shown, so 5 different values for k with 3 pruning cycles each gives 15 results.

Fig. 5.1 and is taken directly from Tables 7.1-5.3. For KSE it shows both the actual and theoretical FPR. Because the current implementation of KSE can only ignore input channels that are fully pruned ($q_c = 0$), which is only a fraction of the total channels, the actual FPR is very close to 100. However, comparing GBIP with the theoretical values of KSE shows a significant advantage for KSE. For all FPR the fitness is higher for KSE than GBIP. Assuming KSE can indeed be optimized to achieve these theoretical values, KSE is the preferred method for compressing YOLOv7. If not, clearly the unoptimized version of KSE is significantly worse and GBIP is the better method.

KSE uses a type of unstructured pruning, which means that some additional optimization is required to achieve acceleration. This is in contrast with structured pruning methods, like GBIP, that directly prune entire channels or filters. The authors of KSE detail how the computation of a compressed convolutional layer could be optimized. Unfortunately, the optimization of KSE is not as straightforward as the paper itself might suggest. The following subsection documents how the optimization is implemented in PyTorch.

5.2.2 PyTorch optimization

As discussed in Subsection 4.3.2, the implementation by the authors create a full weight matrix, by copying the calculated clusters several times, which only results in a small acceleration if an entire input channel was pruned. To accelerate the pruned network, the authors suggest taking the convolution of each input channel c with all clusters $B_{i,c}$ for that channel. Using the indices that matches output channels n with the clusters, $I_{n,c}$, the output can be obtained:

$$Z_{i,c} = B_{i,c} * X_c \tag{5.1}$$

$$Y_n = \sum_{c=1}^{C} Z_{I_{n,c},c} \tag{5.2}$$

Where $B_{i,c}$ is the i-th cluster for input channel c and $I_{n,c}$ is the index of the cluster that convolved with input channel c is part of output channel n.

This is relatively easy to implement in PyTorch but turns out to make the network much slower. One large convolution is split into a lot of small ones. Although the network is compressed, the number of small convolutions is still two orders of magnitude higher than the original number of convolutions. This has two main drawbacks.

Firstly, the optimization requires a lot of manual indexing, which is relatively slow. All those small convolutions still need to be matched and summed together (Equation 5.2). This also means that the intermediate results must be stored till the final output is computed. With a large convolution the intermediate results are only needed in the CUDA kernel, where they can immediately be added to the final output without storing the result.

This is part of the second problem: running a CUDA kernel has some overhead. This overhead comes from launching and initializing the kernel, and moving the data from the slower, global memory of the GPU to the fast L_1 memory. So, running a lot of smaller convolutions ends up costing more time than a few larger convolutions.

To see the difference between the two implementations, the output of the PyTorch Profiler is given in Table 5.4. This shows the total and average GPU time as well as the number of calls for the most used operators. Especially the number of calls is interesting. It shows 7625 convolutions for the optimized implementation compared to 58 in the original. And although these are much smaller convolutions, the average GPU time is almost identical. The other operators are almost all related to storing the intermediate results and indexing these to compute the final output. Most of these operations still happen in the original implementation but are run on an optimized convolution CUDA kernel.

From the profiler output, it is clear that using the 'optimization' in PyTorch is not an option. It is possible that a custom-made, optimized CUDA kernel implementation will accelerate KSE. However, this falls outside the scope of this thesis.

5.2.3 Selecting Best Method

Although KSE could theoretically outperform GBIP, with the current implementation, provided online by the authors of KSE, this is not the case. An attempt has been made to optimize this implementation in PyTorch, but this turns out to make things worse.

GBIP, on the other hand, uses structured pruning which makes it easy to obtain a working, faster model without the need for optimization. GBIP also allows for more control in the size of the resulting network, by varying the number of pruning cycles. So, *which compression method is best suited for compressing YOLOv7?* Looking at Figure 5.1, comparing GBIP with the actual values for KSE, it is clear that GBIP is the better method. Thus, GBIP is selected for compressing YOLOv7.

Based on the selection of GBIP for compressing YOLOv7, the next chapter will detail the experiments that are run on YOLOv7.

Table 5.4: PyTorch profiler output for the original (*orig.*) and optimized (*opt.*) implementation of KSE. The values are an average of 10 runs, with $G = 3, T = 0$. This table contains all operators that take up at least 1% of GPU time in the optimized implementation.

Operator Name	**Total GPU [ms]**		**Num. Calls**		**Avg. GPU [us]**	
	orig.	*opt.*	*orig.*	*opt.*	*orig.*	*opt.*
`aten::conv2d`	20.60	2732.40	58	7625	355	358
`aten::convolution`	19.99	2584.40	58	7625	345	339
`aten::_convolution`	19.13	2458.60	58	7625	330	322
`aten::cudnn_convolution`	17.83	1271.70	58	7625	307	167
`aten::index`	5.56	922.90	54	7621	103	121
`aten::contiguous`	6.37	852.70	57	7570	112	113
`aten::clone`	5.56	746.30	57	7570	97	99
`aten::select`	0.27	705.90	9	22710	30	31
`aten::add`	0.08	451.40	3	7624	27	59
`aten::slice`	2.07	445.90	63	15197	33	29
`aten::as_strided`	1.42	418.60	189	53050	8	8
`aten::item`	-	413.90	-	7567	-	55
`aten::_local_scalar_dense`	-	285.40	-	7567	-	38
`aten::empty_like`	3.16	264.90	112	7625	28	35
`aten::unsqueeze`	-	238.00	-	7567	-	31
`aten::reshape`	1.80	231.60	57	7624	32	30
`aten::copy_`	1.62	231.30	60	7573	27	31
`aten::empty`	3.76	186.10	408	15497	9	12
total	**80**	**7187**				

6 YOLOv7 Experiments

Based on the experiments on YOLOv7-tiny, which are discussed in the previous chapters, GBIP has been chosen for compressing YOLOv7. This chapter will explain which experiments are performed in Section 6.1. This is followed by the implementation details in Section 6.2, which will answer the question: *What are the optimal hyperparameters to compress YOLOv7?*

6.1 Experiments

The goal of the experiments is to compress YOLOv7 to the same number of FLOPs as YOLOv7-tiny. As shown in Table 2.1, YOLOv7 has about 7.7 times more FLOPs than YOLOv7-tiny, which corresponds to a FLOPs pruning ratio (FPR) of 13.2%. Given that for YOLOv7-tiny a FPR of 42% was achieved with $k = 0.6$, it is expected that a large value of k is needed to compress YOLOv7 to the required FPR of 13.2%. There is no way to determine exactly what k will result in enough compression, but fortunately, by increasing the number of pruning cycles, it is possible to keep compressing the network further. This way it is always possible to achieve the required FPR, but the increased training time might make this infeasible.

Since it is difficult to know which k will produce the required compression, the experiments start with $k = 0.9$ running till it reaches this compression. Based on the results of this experiment, the value of k will be adjusted. If it takes longer than three pruning cycles to reach the required compression, higher values of k will be tested and vice versa.

6.2 Implementation Details

Most of the implementation details of GBIP for YOLOv7 are identical to YOLOv7-tiny, which can be found in Subsection 4.3.2.

6.2.1 Attention Transfer

Given that the output of ELAN blocks worked well for attention transfer on YOLOv7-tiny, these are used as starting point for YOLOv7. Table 6.1 shows which layers have been tested. ELAN refers to all ELAN blocks, DOWN to the downsample layers at the first bottom-to-top pathway. Layers 51, 54 and 66 link the bottom-to-top pathway with the top-to-bottom pathway. See Appendix A.2 for a detailed overview of the YOLOv7 architecture with indices. The best result is highlighted in bold and will be used in the experiments.

6.2.2 Pruning Batch Size

Since the importance score for each output channel is computed based on the activation values, instead of the weight values, data is required. The amount of data used to calculate the importance scores, called the pruning batch size, can significantly increase the pruning time. The authors of GBIP do not mention how much data is used, or if they use the entire dataset.

Table 6.1: Layer indices of attention maps tested for Attention Transfer. See Appendix A.2 for the YOLOv7 architecture with indices. YOLOv7 has been pruned with $k = 0.8$ and finetuned for one epoch. DOWN: downsample layers 16, 29, 42

Layer Indices	**Fitness Increase**
ELAN, 51	1.05
ELAN, DOWN	1.12
ELAN, 51, 54, 66	1.51
ELAN, 51, DOWN	0.85
ELAN, 51, DOWN, 54, 66	**1.61**

To find a reasonable value for the pruning batch size, the pruning step is run for increasing pruning batch sizes and recording which output channels are pruned. The results for four different layers of the network are shown in Figure 6.1. At small batch sizes the number and location of pruned channels varies, this is especially clear for layer 24 (bottom left). At larger batch size there only a few channels, with importance scores close to the pruning threshold, that still show change. From these figures the pruning batch size has been set to 2^{10} training samples.

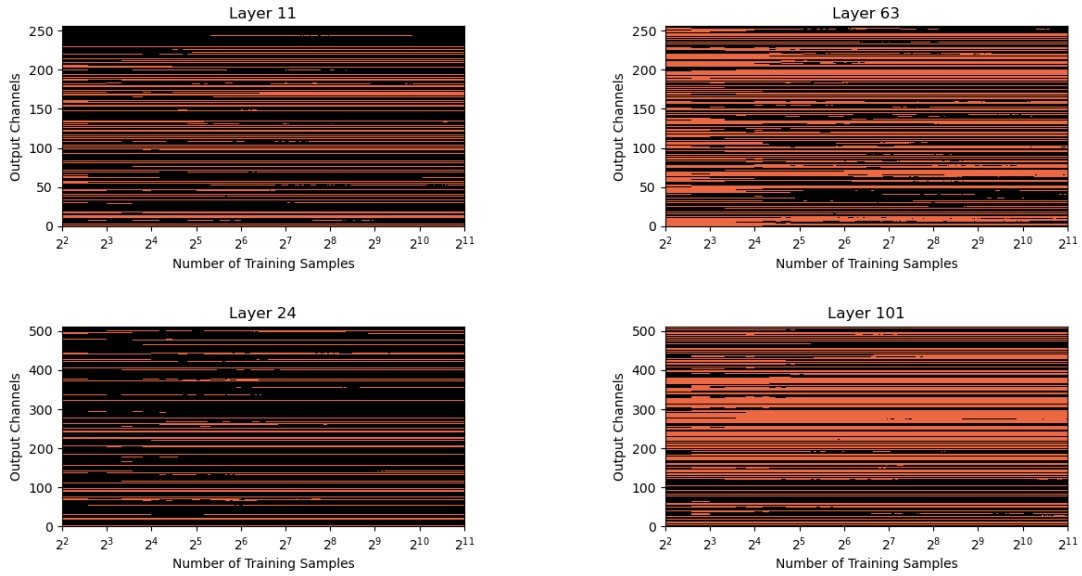


Figure 6.1: Pruning YOLOv7 with $k = 0.9$ for a different number of training samples, or pruning batch size. Orange indicates the output channel at this index is pruned. Solid horizontal lines mean that whether this channel is pruned does not change with increasing batch size.

6.2.3 Ablation Study

Given the change of network, an ablation study is again performed to the test the importance of GBIP's finetuning techniques: attention transfer, output transfer and adversarial game. Since higher values of k are expected, the ablation study will run with $k = 0.9$. Based on the results of the ablation study on YOLOv7-tiny, the number of epochs for each test is reduced to

three. At three epochs the relative difference between techniques was obvious in YOLOv7-tiny, with only a slight increase in fitness afterwards.

Table 6.2 shows the results of the ablation study. Like YOLOv7-tiny, attention transfer again has the largest effect on accuracy recovery. But this time combining attention transfer with output transfer has a slight edge over just attention transfer. For this reason, attention transfer and output transfer are both used in the experiments.

Table 6.2: Results of the ablation study on YOLOv7. All experiments are performed with $k = 0.9$. AT: Attention Transfer, OT: Output Transfer, AG: Adversarial Game. Fitness and AP are percentages. Best results are marked in bold.

AT	OT	AG	Fitness	AP	AP_{50}
-	-	-	41.2	39.5	57.3
✓	-	-	44.2	42.3	**60.7**
-	✓	-	43.4	41.6	59.3
-	-	✓	42.0	40.2	58.0
✓	✓	-	**44.3**	**42.5**	60.5

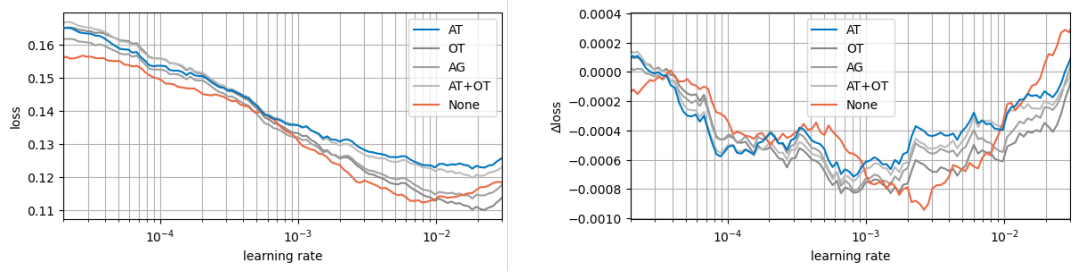
6.2.4 Learning Rate

To decide the initial learning rate a learning rate range test [73] is performed. While training the learning rate is slowly increased from a very low (10^{-6}) to a very high (1) value while logging the loss value. The optimal learning rate, where the loss decreases fastest, is obtained by taking the derivative of the loss with respect to the learning rate. A small amount of smoothing, using a moving average, is applied to filter out noise for both the loss and its derivative.

Figure 6.2 shows the loss and its derivative for the ablation study experiments. The loss decreases up to a learning rate of around 10^{-2}. From the derivative plot, the optimal learning rate for 'None' (no finetuning techniques) is around 3×10^{-3}. The other variants are all very similar to each other (which is why only 'AT' is shown in color) and have an optimal learning rate around 6×10^{-4}.

For all experiments, including the ablation study, this approach has been taken to determine the optimal learning rate. Not only at the start of the experiment, but also after each successive pruning step. This learning rate is still used in combination with cosine annealing as described in Subsection 2.2.3.

This chapter explained how the experiments will be run. It answered the question: *What are the optimal hyperparameters to compress YOLOv7?* The layers used for attention transfer are indicated in Table 6.1. The pruning batch size is set to 2^{10} based on Figure 6.1. An ablation study (Subsection 6.2.3) is performed to determine that attention transfer and ouput transfer will be used as finetuning techniques. And the procedure for determining the learning rate is explained in Subsection 6.2.4.



(a) Loss.

(b) Derivative of the loss.

Figure 6.2: Determining learning rates for $k = 0.9$ using different finetuning techniques. Learning rate is slowly increased while monitoring loss. Maximum learning rate is at the point where the loss starts increasing. The minimum of the derivative of the loss indicates the optimal learning rate where the loss decreases the fastest. For 'None' this optimal learning rate lies around 2×10^{-3}, while for the other variants it is around 6×10^{-4}.

7 Results

This chapter reports the results of compressing YOLOv7 with GBIP. In Section 7.1, these results are used to answer the main research question: *Can a state-of-the-art compression of YOLOv7 achieve higher accuracy than YOLOv7-tiny at the same number of floating-point operations?* In Section 7.2, the different compressed models are evaluated and compared, answering the questions: *Does one large pruning step work better than several smaller pruning steps?* and: *Why is the selected compression method not able to outperform YOLOv7-tiny?*

7.1 Final Results

YOLOv7 has been compressed with different values of k, stopping when the FLOPs Pruning Ratio (FPR) is close to that of YOLOv7-tiny (13.2). Table 7.1 gives the results of these experiments. The best compressed network is achieved with $k = 0.95$ after 2 pruning steps. $k = 0.92$ has a very small fitness increase, but a worse FPR.

Can a state-of-the-art compression of YOLOv7 achieve higher accuracy than YOLOv7-tiny at the same number of floating-point operations? Surprisingly, none of the compressed networks achieves a fitness close to that of YOLOv7-tiny. This means that it is better to use the handcrafted YOLOv7-tiny than compressing YOLOv7 with a state-of-the-art compression method.

Table 7.1: Results of GBIP experiments on YOLOv7. Fitness, AP, FLOPs Pruning Ratio (FPR) and Parameter Pruning Ratio (PPR) are given as percentages. Note $k = 0$ refers to the unpruned baseline.

k	**Pruning Steps**	**Fitness**	**AP**	**AP$_{50}$**	**PPR**	**FPR**
YOLOv7-tiny		39.2	37.4	55.2	16.8	13.2
0.9	5	22.9	21.6	34.3	20.9	13.0
0.91	3	27.0	25.6	39.5	19.6	12.6
0.92	3	28.5	27.0	41.3	19.3	12.6
0.95	2	28.4	27.0	41.3	15.3	11.3
1.015	1	21.5	20.3	32.5	12.8	13.1

The fitness and FPR after each pruning step for all experiments are shown in Figure 7.1. The bottom left of the left plot, where the compressed models reach the required FPR, is magnified on the right. The fitness degrades quicker the further the model is compressed. This is expected since at first less relevant information can be removed with little accuracy loss, but at lower FPR all unimportant filters are already removed and compressing further reduces the accuracy faster.

It also seems that with many pruning steps, like $k = 0.9$ with 5 steps, the accuracy decreases faster than with only two or three pruning steps. Therefore, $k = 1.015$ has been tested to see if this extends to only one pruning step. However, after pruning once with $k = 1.015$ the required FPR is reached, but the resulting fitness is below even that of $k = 0.9$.

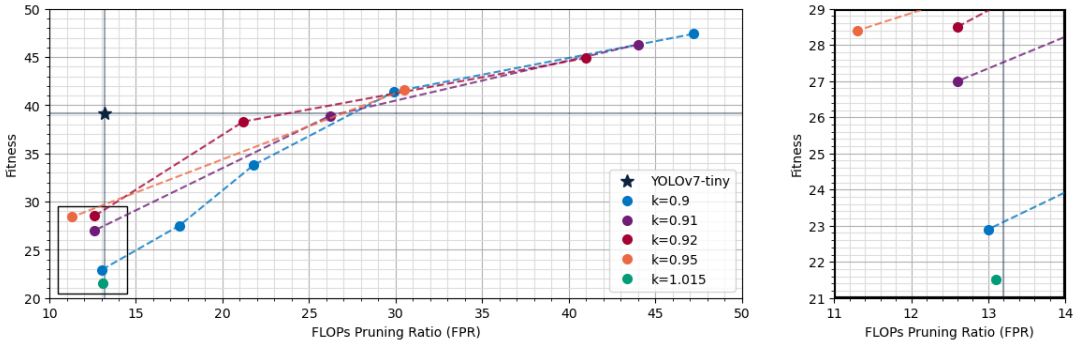


Figure 7.1: Fitness vs FPR for GBIP experiments after each pruning step for different k. Final models are at the left bottom (within the black box) and are enlarged on the plot on the right. Solid lines indicate FPR (13.2%) and fitness (39.2%) of YOLOv7-tiny.

7.2 Comparison

This section answers the questions: *Does one large pruning step work better than several smaller pruning steps?* and: *Why is the selected compression method not able to outperform YOLOv7-tiny?*

Figure 7.2 shows the pruning ratio of each layer in YOLOv7 after the first and last pruning step for different values of k. Looking at Figure 7.2a, obviously a higher k compresses the network further in one step. For $k = 0.9$ to 0.92 and to 0.95 the pruning ratio decreases quite gradually with highs and lows in the same locations. However, for $k = 1.015$ the pruning ratio curve is almost flat, with only minor variation between layers. This suggests that at such high k the compression is no longer able to distinguish between layers with important filters and those with less important filters.

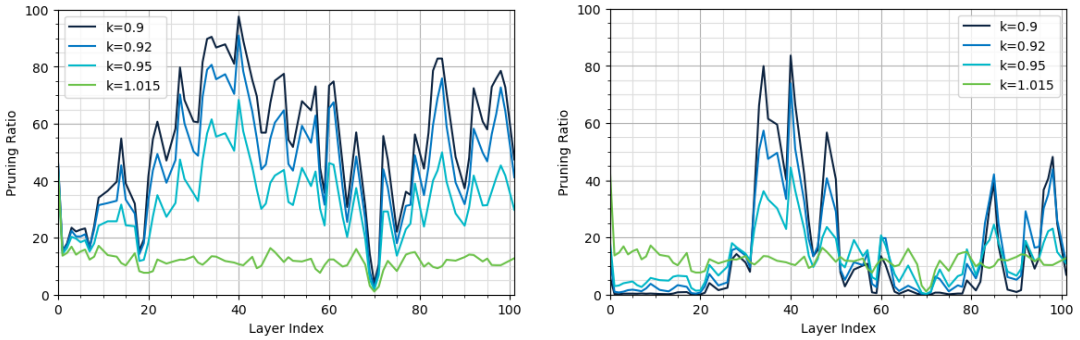
In Figure 7.2b all models have very similar FPR. $k = 1.015$ is again an outlier, but the other curves all follow a similar trajectory. However, the lower k, the more accentuated the peaks and valleys are. This means that certain layers are compressed extremely far, while others are compressed significantly less compared to higher k.

It seems that this is mostly caused by the number of pruning steps. Figure 7.3 shows the pruning ratio for all pruning steps at $k = 0.9$. Each successive pruning step further accentuates the peaks and valleys, resulting in an extremely low pruning ratio for certain layers, which might create a bottleneck for the information flow in the network and thus cause increased accuracy degradation.

So far it is found that large k, like 1.015, results in a very even pruning ratio, which ignores the relative importance of the layers. At the same time, low values of k require more pruning steps and thus over compress less important layers. These two facts explain why the experiments with 2 or 3 pruning steps perform better than the others.

It is also interesting to look at the locations of the layers with high and low pruning ratio. From Figure 7.2b it can be seen that layers 32 to 50, 84 to 86 and 96 to 99 have a relative high pruning ratio, suggesting that these layers contain more relevant information. Figure 7.4b highlights the location of these layers in the YOLOv7 architecture.

Similarly, layers 1 to 21 and 63 to 78 have an extremely low pruning ratio. The location of these layers is shown in Figure 7.4a. Comparing the locations of high and low pruning ratios,



(a) After first pruning step.

(b) After last pruning step.

Figure 7.2: Pruning Ratio of each convolutional layer in YOLOv7 for different values of k after (a) the first and (b) last pruning step.

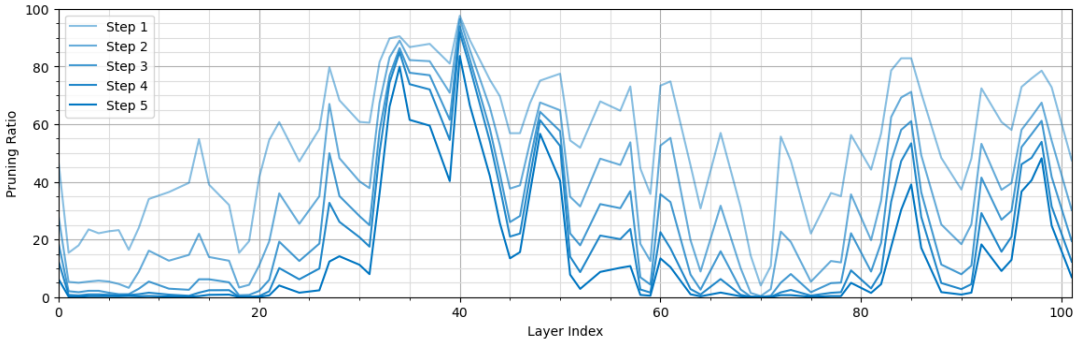
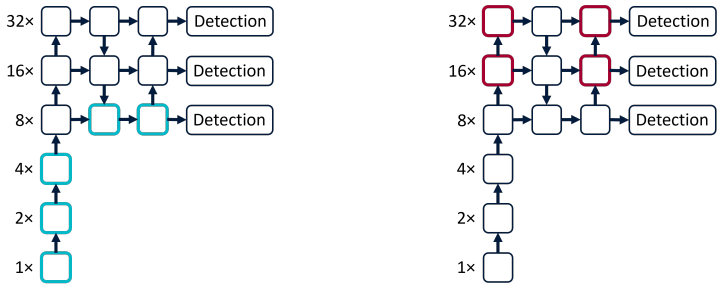


Figure 7.3: Pruning Ratio of each convolutional layer in YOLOv7 after each pruning step at $k = 0.9$.

the detections at higher scales are clearly prioritized over that at the lower scale. At the higher scales, the information can flow through several paths, but at the lowest scales there is only one path. It is precisely this path that is compressed very heavily. And there is no way to recover the information later in the network. This explains why $k = 0.9$ with 5 pruning steps, which has the most extreme compression in these layers, performs worse than those with only two or three pruning steps.

So, *does one large pruning step work better than several smaller pruning steps?* No, from the comparison above it can be concluded that one large pruning step does not work better than several smaller pruning steps. At the same time, too many steps also hurt the performance. In this case, the optimal number of pruning steps is found to be two or three.

The results in this chapter shows that YOLOv7-tiny outperforms YOLOv7 compressed with GBIP. But *why is the selected compression method not able to outperform YOLOv7-tiny?* One reason for this seems to be the fact that GBIP can only remove filters and not entire layers. Especially in the lower layers, any information that is removed cannot be recovered. At the same time some layers are left with only a few percent of their weights, making it impossible to pass all input information through to the next layer. This means that a layer with only a few weights will decrease the accuracy, and it would be better to remove the layer in its entirety.



(a) Blue: Low FPR.

(b) Red: High FPR.

Figure 7.4: Location of layers with high and low FPR.

Another way to avoid the loss of information in the earlier layers is to change the pruning threshold to allow less compression in the earlier layers and more in the later layers. Both changes would require a lot of extra testing and are left as recommendations for further research.

8 Conclusion

In this final chapter the research questions are answered in Section 8.1 and some recommendations for further research are given in Section 8.2.

8.1 Research Questions

8.1.1 Main Research Question

The goal of this thesis is to see if compression of a large neural network can achieve better results than a hand-designed smaller network. Specifically:

> *Can a state-of-the-art compression of YOLOv7 achieve higher accuracy than YOLOv7-tiny at the same number of floating-point operations?*

From the experiments done on YOLOv7 with GBIP, it turns out that YOLOv7-tiny outperforms the compressed YOLOv7. The best performing model resulted from $k = 0.95$, which achieves a 11.3% FLOPs Pruning Ratio (FPR) with a fitness of 28.4%. YOLOv7-tiny (with a FPR of 13.2%) achieves a significantly higher fitness of 39.2%.

Clearly, YOLOv7-tiny is preferable over YOLOv7 compressed with GBIP.

8.1.2 Sub Questions

1. What is the state-of-the-art in neural network compression?

Section 2.3 gives some background information about different types of compression techniques. In general, these techniques can be divided into four categories: pruning, quantization, tensor decomposition and compact architectures. In Chapter 3, based on a search of the literature, a list of best performing compression methods is compiled. The current state-of-the-art achieves a FPR of 11.2% with ResNet-56 on CIFAR-10 [13] and 21.3 % FPR with ResNet-50 on ImageNet [66], both with an accuracy loss of less than 1%.

2. What are the best networks for comparing compression methods?

The main reason a network is suitable for comparing compression methods, is the fact that most recent papers report results using this network. From the literature, Section 3.1 found that the best networks are ResNet-50 trained on ImageNet and ResNet-56 trained on CIFAR-10.

3. Which compression method is best suited for compressing YOLOv7?

From the list of best performing compression methods (Table 3.1), two methods were selected: GBIP and KSE. These methods were selected after first removing methods with very high finetuning requirements.

Both GBIP and KSE have been tested on YOLOv7-tiny. Chapter 4 goes into the implementation details of these methods for YOLOv7-tiny. Based on the results (Chapter 5), GBIP is chosen as best method for compressing YOLOv7. KSE could theoretically outperform GBIP, but as mentioned in Section 5.2, these theoretical values were unattainable.

4. What are the optimal hyperparameters to compress YOLOv7?

In Chapter 6 the hyperparameters are discussed. The learning rate is determined by a learning rate range test (Section 6.2.4). The learning rate is slowly increased from a very low to very high values while logging the loss. Using the derivative of this loss curve, the learning rate where the loss decreases fastest can be selected.

Another hyperparameter that requires tuning is the number of training samples, or pruning batch size, used for calculating the importance score. Section 6.2.4 shows that after 2^{10} training samples there is almost no more change in which filters gets pruned.

In the GBIP paper, three finetuning techniques are introduced to recover the accuracy of a pruned network: attention transfer, output transfer and adversarial game. An ablation study (Section 6.2.3) shows that for YOLOv7 a combination of attention and output transfer works best.

5. Does one large pruning step work better than several smaller pruning steps?

From the experiments on YOLOv7 (Chapter 7), it shows that using only one pruning step does not achieve good results. In that case, the layers of the network are very evenly compressed, ignoring any difference in relevance between layers. On the other hand, with five pruning steps the compression is very uneven over the layers, to the point that less than 1% of the parameters are left in several layers. For the experiments in this thesis, two or three pruning steps results in the best accuracy. Incidentally, in the GBIP paper the number of pruning steps is also set to two or three, although no explanation is given for this choice.

6. Why is the selected compression method not able to outperform YOLOv7-tiny?

It is difficult to give a definitive answer to this question. One very probable answer lies in the way GBIP compresses a network, namely by removing filters. YOLOv7-tiny has fewer layers than YOLOv7, but GBIP cannot remove entire layers, so all the information must still flow through each layer. This means that even if a layer is not necessary it still needs a way to pass the information on to the next layer and the weights cannot be set to zero.

8.2 Recommendations

Based on the experiments in this thesis there are several recommendations for future research. One obvious recommendation is further pursuing the optimization of KSE. In theory KSE should be able to outperform GBIP, but the optimization was unsuccessful. It is very well possible that with a custom CUDA implementation of KSE the results will be closer to the theory.

Another recommendation is investigating a varying pruning threshold. Currently, the pruning threshold is the same in each layer of the network. However, if the compression in the early layers is too large, it is impossible to recover the lost information. Maybe by allowing lower compression in the early layers and more in the later layers, it is possible to achieve better accuracy at the same FPR.

The last recommendation is testing the removal of entire layers. Based on the performance of YOLOv7-tiny it is clear that a network with fewer layers can outperform a compressed network with more layers. However, allowing the removal of layers while compressing might result in a model that can outperform YOLOv7-tiny.

A Architectures

A.1 YOLOv7-tiny

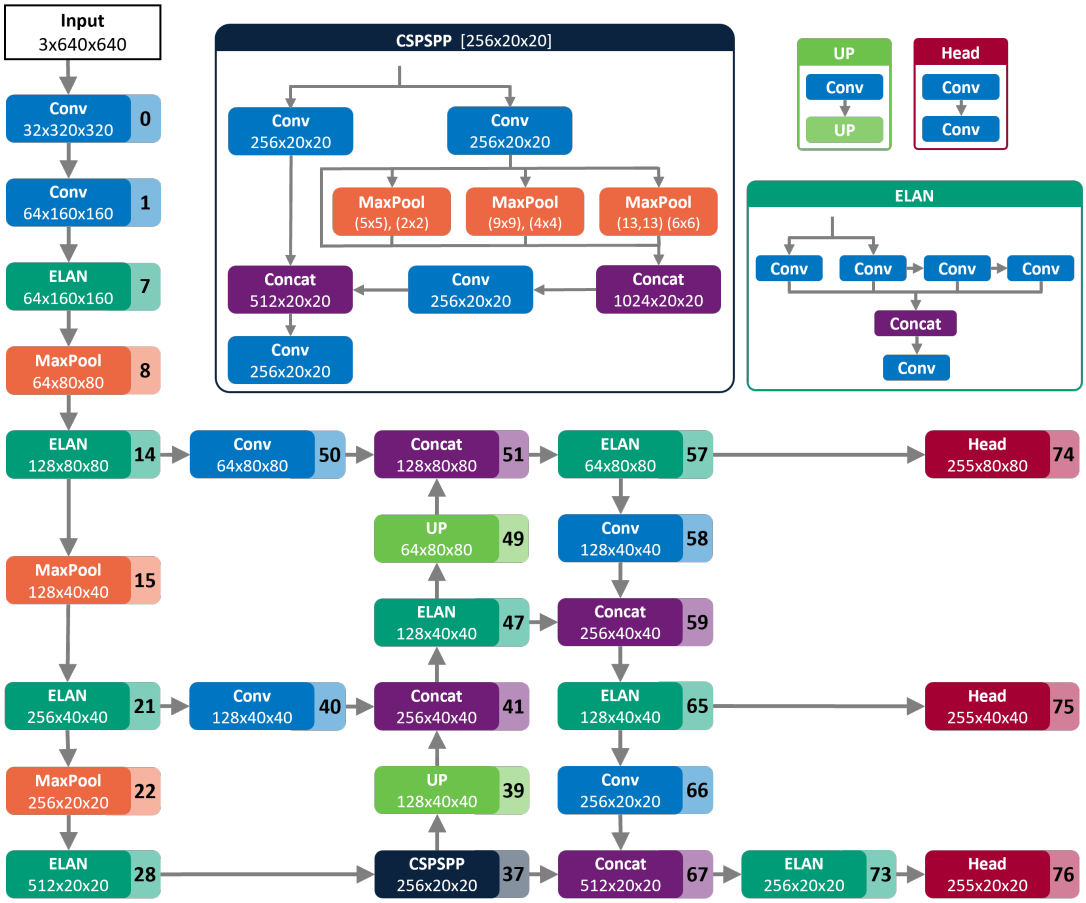


Figure A.1: YOLOv7-tiny architecture. Name of each block is given on top of the box; output dimensions are given below; numbers on the right indicate index of last layer in the block. In the CSPSPP block, the numbers under MaxPool indicate kernel and padding size.

A.2 YOLOv7

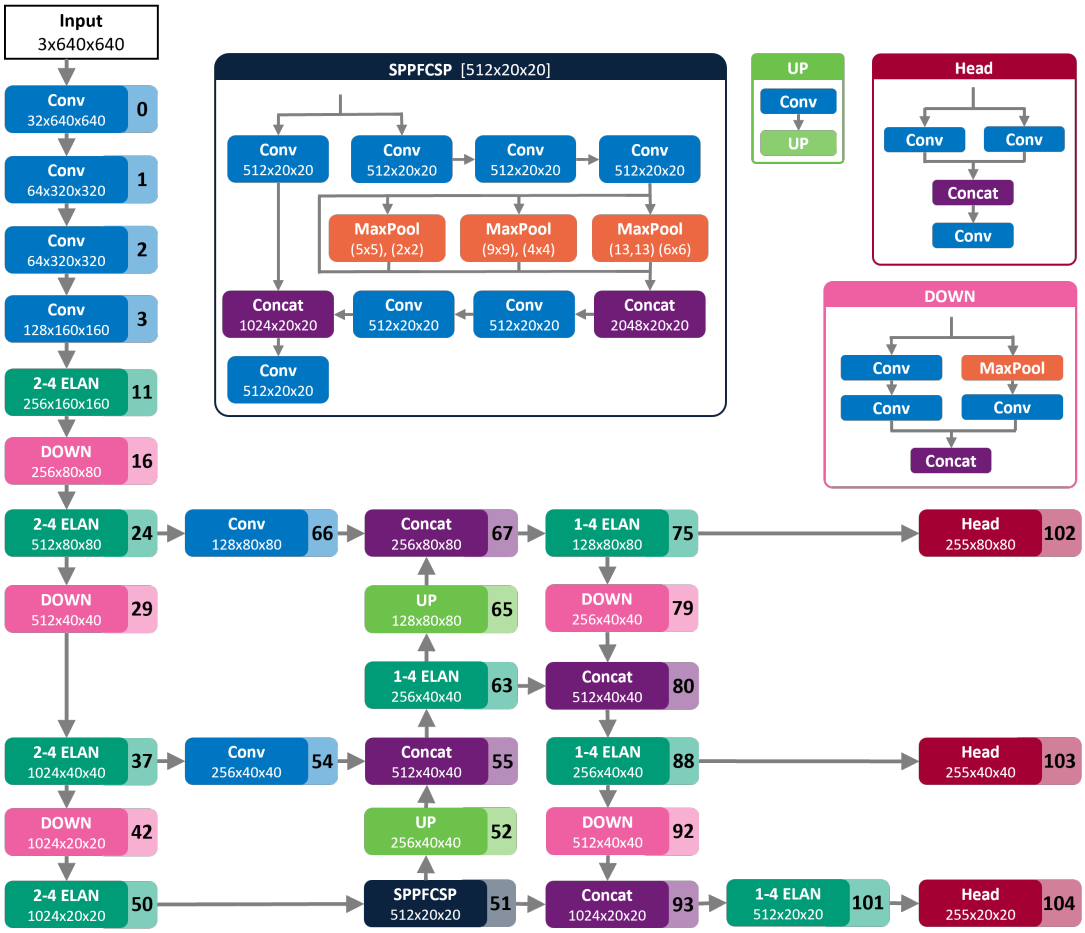


Figure A.2: YOLOv7 architecture. Name of each block is given on top of the box; output dimensions are given below; numbers on the right indicate index of last layer in the block. In the SPPFCSP block, the numbers under MaxPool indicate kernel and padding size. 2-4 and 1-4 ELAN blocks are show in Figure 2.6.

Bibliography

[1] J. Steinbrener and P. Desai, "Comparison of deep learning architectures on embedded devices and generalized fixed-point conversion algorithm," pp. 343–347, 06 2019.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[3] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, 03 2017.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] R. Pilipovic, P. Bulić, and V. Risojević, "Compression of convolutional neural networks: A short survey," pp. 1–6, 03 2018.

[6] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020.

[7] L. Dutta and S. Bharali, "Tinyml meets iot: A comprehensive survey," *Internet of Things*, vol. 16, p. 100461, 10 2021.

[8] F. Götz, "The data deluge: What do we do with the data generated by avs?." https://blogs.sw.siemens.com/polarion/the-data-deluge-what-do-we-do-with-the-data-generated-by-avs/, Jan 2021. Accessed: 2023-11-06.

[9] K. Nan, S. Liu, J. Du, and H. Liu, "Deep model compression for mobile platforms: A survey," *Tsinghua Science and Technology*, vol. 24, pp. 677–693, 12 2019.

[10] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 10 2017.

[11] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," 02 2017.

[12] J. Bi and S. Gunn, "Sparse deep neural networks for embedded intelligence," pp. 30–38, 11 2018.

[13] W. Zhang and Z. Wang, "Fpfs: Filter-level pruning via distance weight measuring filter similarity," *Neurocomputing*, vol. 512, pp. 40–51, 2022.

[14] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 04 2017.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[16] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022.

[17] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," vol. 27, pp. 807–814, 06 2010.

[18] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "A comprehensive survey and performance analysis of activation functions in deep learning," *CoRR*, vol. abs/2109.14545, 2021.

[19] D. Hendrycks and K. Gimpel, "Bridging nonlinearities and stochastic regularizers with gaussian error linear units," *CoRR*, vol. abs/1606.08415, 2016.

[20] R. J. Tan, "Breaking down mean average precision (map)." https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52, Mar 2019. Accessed: 2023-11-06.

[21] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, June 2010.

[22] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[23] "Object detection on pascal voc 2012." https://paperswithcode.com/sota/object-detection-on-pascal-voc-2012. Accessed: 2023-11-06.

[24] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *CoRR*, vol. abs/1807.05511, 2018.

[25] "Object detection on coco test-dev." https://paperswithcode.com/sota/object-detection-on-coco. Accessed: 2023-11-06.

[26] R. Kaur and S. Singh, "A comprehensive review of object detection with deep learning," *Digital Signal Processing*, vol. 132, p. 103812, 2023.

[27] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[28] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.

[29] A. Benali Amjoud and M. Amrouch, "Object detection using deep learning, cnns and vision transformers: A review," *IEEE Access*, vol. PP, pp. 1–1, 01 2023.

[30] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[31] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[32] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016.

[33] C.-Y. Wang, H.-Y. M. Liao, and I.-H. Yeh, "Designing network design strategies through gradient path analysis," 2022.

[34] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020.

[35] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *CoRR*, vol. abs/1710.09412, 2017.

[36] T. Devries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *CoRR*, vol. abs/1708.04552, 2017.

[37] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-iou loss: Faster and better learning for bounding box regression," *CoRR*, vol. abs/1911.08287, 2019.

[38] H. Salehinejad and S. Valaee, "Ising-dropout: A regularization method for training and compression of deep neural networks," 05 2019.

[39] J. Chang and S. Jin, "Prune deep neural networks with the modified l1/2 penalty," *IEEE Access*, vol. PP, pp. 1–1, 12 2018.

[40] M. Alnemari and N. Bagherzadeh, "Efficient deep neural networks for edge computing," pp. 1–7, 07 2019.

[41] P. Merolla, R. Appuswamy, J. Arthur, S. Esser, and D. Modha, "Deep neural networks are robust to weight binarization and other non-linear distortions," 06 2016.

[42] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 02 2016.

[43] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[44] Q. Wu, W. Li, X. Lu, H. Zhang, H. Luo, and C. Lei, "A deep learning model compression algorithm based on optimal clustering," p. 171, 05 2019.

[45] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 10 2016.

[46] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pp. 2365–2369, 01 2013.

[47] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 05 2016.

[48] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," 12 2014.

[49] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, "Tensorizing neural networks," 09 2015.

[50] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," 02 2016.

[51] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 03 2015.

[52] M. V. de Carvalho and M. Pratama, "Improving shallow neural network by compressing deep neural network," pp. 1382–1387, 11 2018.

[53] C.-J. Chen, K.-C. Chen, and M.-c. Martin-Kuo, "Acceleration of neural network model execution on embedded systems," pp. 1–3, 04 2018.

[54] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[55] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[56] "Image classification on imagenet." https://paperswithcode.com/sota/image-classification-on-imagenet. Accessed: 2023-11-06.

[57] "Image classification on cifar-10." https://paperswithcode.com/sota/image-classification-on-cifar-10. Accessed: 2023-11-06.

[58] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[59] S. Vadera and S. Ameen, "Methods for pruning deep neural networks," *CoRR*, vol. abs/2011.00241, 2020.

[60] W. Zhang and Z. Wang, "Pca-pruner: Filter pruning by principal component analysis," *J. Intell. Fuzzy Syst.*, vol. 43, pp. 4803–4813, jan 2022.

[61] M. U. Haider and M. Taj, "Comprehensive online network pruning via learnable scaling factors," *CoRR*, vol. abs/2010.02623, 2020.

[62] C. Sarvani, M. Ghorai, S. R. Dubey, and S. S. Basha, "HRel: Filter pruning based on high relevance between activation maps and class labels," *Neural Networks*, vol. 147, pp. 186–197, mar 2022.

[63] W. Gao, Y. Guo, S. Ma, G. Li, and S. Kwong, "Efficient neural network compression inspired by compressive sensing," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022.

[64] J. Chang, Y. Lu, P. Xue, Y. Xu, and Z. Wei, "Global balanced iterative pruning for efficient convolutional neural networks," *Neural Computing and Applications*, Jul 2022.

[65] J. Zhu and J. Pei, "Progressive kernel pruning cnn compression method with an adjustable input channel," *Applied Intelligence*, vol. 52, pp. 1–22, 07 2022.

[66] Y. Li, S. Lin, B. Zhang, J. Liu, D. S. Doermann, Y. Wu, F. Huang, and R. Ji, "Exploiting kernel sparsity and entropy for interpretable CNN compression," *CoRR*, vol. abs/1812.04368, 2018.

[67] K. Zhao, A. Jain, and M. Zhao, "Iterative activation-based structured pruning," *CoRR*, vol. abs/2201.09881, 2022.

[68] M. R. Ganesh, J. J. Corso, and S. Y. Sekeh, "MINT: deep network compression via mutual information-based neuron trimming," *CoRR*, vol. abs/2003.08472, 2020.

[69] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 07 2016.

[70] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," *CoRR*, vol. abs/1612.03928, 2016.

[71] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. S. Doermann, "Towards optimal structured CNN pruning via generative adversarial learning," *CoRR*, vol. abs/1903.09291, 2019.

[72] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[73] L. N. Smith, "No more pesky learning rate guessing games," *CoRR*, vol. abs/1506.01186, 2015.