Deep Neural Network Pruning and Sensor Fusion in Practical 2D Detection

by

Morteza Mousa Pasandi

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

Convolutional Neural Networks (CNNs) have been extensively studied and applied to various computer vision problems, including object detection, semantic segmentation, and autonomous driving. Convolutional Neural Network (CNN)s extract complex features from input images or data to represent objects or patterns. Their highly complex architecture, however, and the size of their learned weights make their time and resource intensive. Measures like pruning and fusion, which aim to simplify the structure and lessen the load on the network's resources, should be considered to resolve this problem.

In this thesis, we intend to explore the effect of pruning on segmentation and object detection as well as the benefits of using sensor fusion operators in the 2d space to boost the existing networks' performance. Specifically, we focus on structured pruning, quantization, and simple and learnable fusion operators. We also study the scalability of different algorithms in terms of the number of parameters and floating points used.

First, we provide a general overview of CNNs and the history of pruning and fusion operations. Second, we explain the advantages of pruning and discuss the contrast between the unstructured and structured types. Third, we discuss the differences between simple fusion and learnable fusion.

In order to evaluate our algorithms, we use several classification and object detection datasets such as Cifar-10, KITTI and Microsoft COCO. By applying our proposed methods to the studied datasets, we can assess the efficiency of the algorithms. Furthermore, this allows us to observe the improvements in task-specific losses.

In conclusion, our work is focused on analyzing the effect of pruning and fusion to simplify existing networks and improve their performance in terms of scalability, task-specific losses, and resource consumption. We also discuss various algorithms, as well as datasets which serve as a basis for the evaluation of our proposed approaches.

Acknowledgements

I feel incredibly fortunate to have had the support of so many people throughout the writing of my master's thesis. In particular, I owe immense thanks to my professor, Robert Laganiere. He has been an inspiring professor who has provided invaluable guidance and mentorship during my academic journey. He helped to ensure I was pursing my thesis in the right direction and pushed me to think through difficult problems and put forth creative solutions. I'm truly thankful for all his support.

I must also thank my family, especially my father, mother, and brothers, Mohammad Ebrahim and Mohammad Eshmael, and their families. Over the last year and a half, they have been there to encourage me, provide me with words of wisdom, and offer their unwavering love and support. I'm thankful for my sister-in-law, Helen, and their amazing children Naveed and Alvand, for always bringing smiles to my face, and for always believing in me.

Finally, I would like to thank my amazing friends, Tianran Liu, Zeping, Wen Wen, Yahya Masoud and Rahman Atiquar. They provided much-needed encouragement throughout my thesis writing, and in moments of doubt, inspired me to keep going and to give it my best. I thank them sincerely for all the moments we have shared together and for helping me stay on track. Their advice, support, and friendship has been priceless.

Table of Contents

List of Tables

List of Figures

Abbreviations

ANNs Artificial Neural networks 1

BGF Bilateral Guided Fusion ix, xii, 62, 63, 67, 70, 75, 76

CNN Convolutional Neural Network iii, 2, 7, 10, 11, 19, 27, 30, 32, 36–40, 43, 49, 88, 89

CV Computer Vision 14

DFS Depth-First Search 42

DHI Depth Height Intensity 57, 72

DNNs Deep neural networks 1–3, 6, 7, 24, 36

Fisher Group Fisher Pruning 41, 42, 50, 51, 54, 55

FLOPS floating-point operations per second ix, 43, 47, 51, 70

FPN Feature Pyramid Network vi, ix–xii, 19–21, 45, 46, 49–52, 58, 59, 67–71

FPS Frames Per Second 50, 51

GREG Neural Pruning via Growing Regularization vi, 27, 40–43, 48, 55, 75

IOU Intersection Over Union 44, 45

LTH Lottery Ticket Hypothesis 50, 51, 53–55

MAC Multiply and Accumulate 37

MFB multi-modal factorized bilinear pooling ix, x, xii, 62, 63, 67, 70, 71, 74–76

MRCNN Mask R-CNN xi, xii, 23, 24, 38, 45, 46, 48–52, 55

NAS Neural Architecture Search 6

NLP Natural language processing 14

PFEC Pruning Filters for Efficient ConvNets vi, 27, 32, 38, 47

PUFA Pruning Filter with Attenuation vi, 32, 39, 47, 48, 55

ROI Region of Interest 21, 23, 57, 58

RPN Region Proposal Network 21, 57

SGD Stochastic Gradient Descent 12, 44–46, 48, 66

SN Simple Neck xii, 59, 70

SOTA state-of-the-art 2, 21

ViT Vision Transformers 13, 14

Chapter 1

Introduction

Artificial Neural networks (ANNs) draw some inspiration from biological neural networks, but they differ significantly from them in their practical implementation. They accept input from other neurons and integrate them in different ways to produce an output response. Deep neural networks (DNNs) are networks that have many layers and perform a large number of calculations. DNNs are used to solve real-world problems, such as autonomous driving, finding tumours, and helping robots. To tackle these problems, DNNs need to be trained in such a way that they can learn how to solve problems that are presented to them.

Supervised learning is when a computer can learn to do things based on data that it has been given and a set of instructions to look at. A labelled dataset is a set of data that has labels attached to it, like a name or a number. So when you give the computer a labelled dataset, it knows what the data it is looking at means. For example, if you give it a dataset of pictures of cars and ships and it has been labelled, it can learn to tell the difference between ships and cars. Classification and Regression are two examples of supervised learning, where the computer can take in data and use it to make predictions about new data. Unsupervised learning is when a computer looks at data and figures out how it fits together, without any labels or instructions. Examples of this are clustering, which looks at groups of data points and tries to find patterns and relationships between them. Feature reduction looks at reducing the complexity of data to make it easier to process.

As illustrated in Figure 1.1, Classification usually predicts the given data points or input to pre-specified target categories or labels, which are called classes. This process could be visualized as a mapping function that directly puts each input variable like images, texts

or videos to individual output sections that are target classes. Object localization involves pinpointing one or more objects in an image by drawing bounding boxes around their shape. Object detection takes this concept one step further; in addition to localizing the object, it can also classify it accurately. A more elaborate version of this, called semantic segmentation, goes beyond simply outlining the object and gives a categorization based on each individual pixel. Finally, instance segmentation is the most detailed as it can differentiate between each instance of the object as well as indicate the class to which it belongs.

Figure 1.1: Instance Segmentation vs Object Detection vs Classification

1.1 Motivation

CNNs are a type of DNNs that is used to analyze visual patterns in images and videos. They use convolutional filters (ConVNets) and hierarchical patterns of data to detect visual features, objects and other patterns in the input data.

As illustrated in Figure1.2, our motivation for this thesis stems from three distinct events that are related to each other. The first event is the increased use of power-constrained specialized devices. This rise in usage means that there is a need for systems or methods that can perform specific tasks yet consume as little power as possible.

The second event is the impressive accuracy of state-of-the-art (SOTA) CNN models. This shows that machine learning models can capture complex patterns and accurately recognize them. Therefore, we can use this potential to build models that may be more performant and better suit the industry's needs.

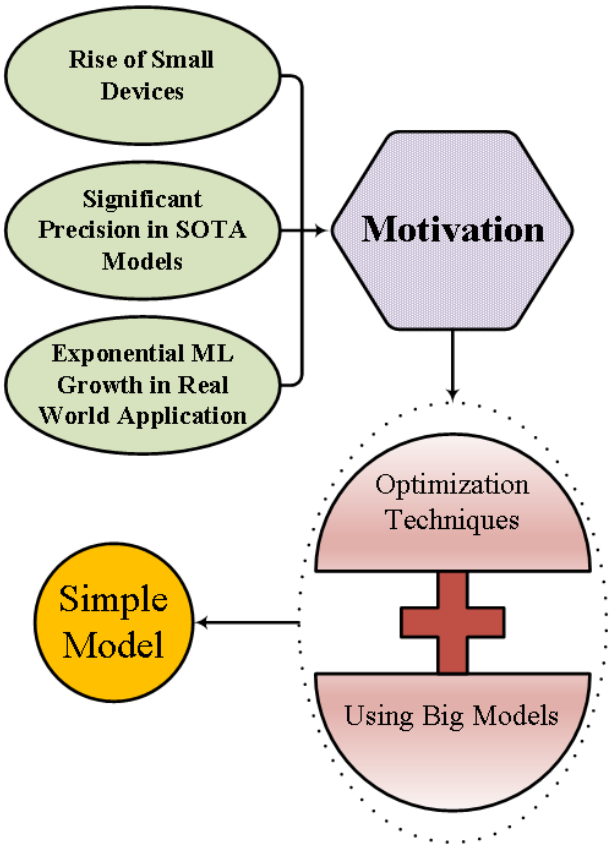

Figure 1.2: Motivation on small devices

The third event is the exponential growth in Machine-Learning usage in different world applications. This event indicates that ML models can be deployed and used in real-life applications. Therefore, by establishing and implementing optimization techniques we may be able to build simpler models that meet better the industry's requirements.

To guarantee the deployment of DNNs in the real world, like autonomous driving, two stringent requirements must be fulfilled: one is the accuracy and dependability of the perception results and the other is the ability to support a low-power computing environment. This combination of requirements ensures the robust output of the model while at the same time keeping the power consumption to a minimum.

The main focus of this paper is to discuss techniques for combining large models to produce simplified and efficient variations that align with the industry's requirements. For this purpose, we explore different optimization techniques and investigate the impact of

compounding big models in order to develop such models. Throughout the research, we will measure and analyze the performance of the resulting simpler models, in terms of accuracy and time consumed, among other things.

1.2 Summary of Problems

Deep networks are powerful complex architectures that can be used to detect patterns in large datasets. These networks tend to have a lot of layers both on the convolutional (focusing on pixel patterns) and linear (focusing on high-level correlations and data representations) sections. As shown in Figure 1.3, building, training, and deploying machine learning models involves a great deal of complexity. This is especially true due to the three broad categories of problems that tend to arise during this process: model, dataset, and hardware issues.

The most common dataset problems arise from misalignment or missing labels in the data, as well as noise from the dataset or model which can decrease accuracy. To make sure accuracy is maintained, proprietary datasets must be managed properly and tested under the right conditions.

Hardware problems can be a major issue as well, such as insufficient video graphic memory (VRAM), random access memory (DRAM) or local storage. Additionally, the device and model architecture can influence the precision of the floating-point operations.

Finally, model issues tend to involve complex simulations, equations, large datasets, and lengthy inference times. Deep learning requires even more layers and functions, increasing the risk of errors.

As such, it is important for machine learning developers to be aware of all the potential pitfalls that come with building, training, and deploying models. Knowing which category of problem is at fault can be a huge help in finding a resolution to the issue. Additionally, research must be done in order to determine which hardware and datasets are most suitable to achieve the best results.

When making changes to something as complicated as a neural network, the best approach is often to take small steps. This fact is especially relevant when it comes to the architecture of the network, which includes how its various parts are laid out and how they interact with each other. In making too dramatic changes to this framework all at once, it can be difficult to anticipate and prevent issues that may arise, compromising the network's learning and performance abilities. Fortunately, there are a variety of techniques

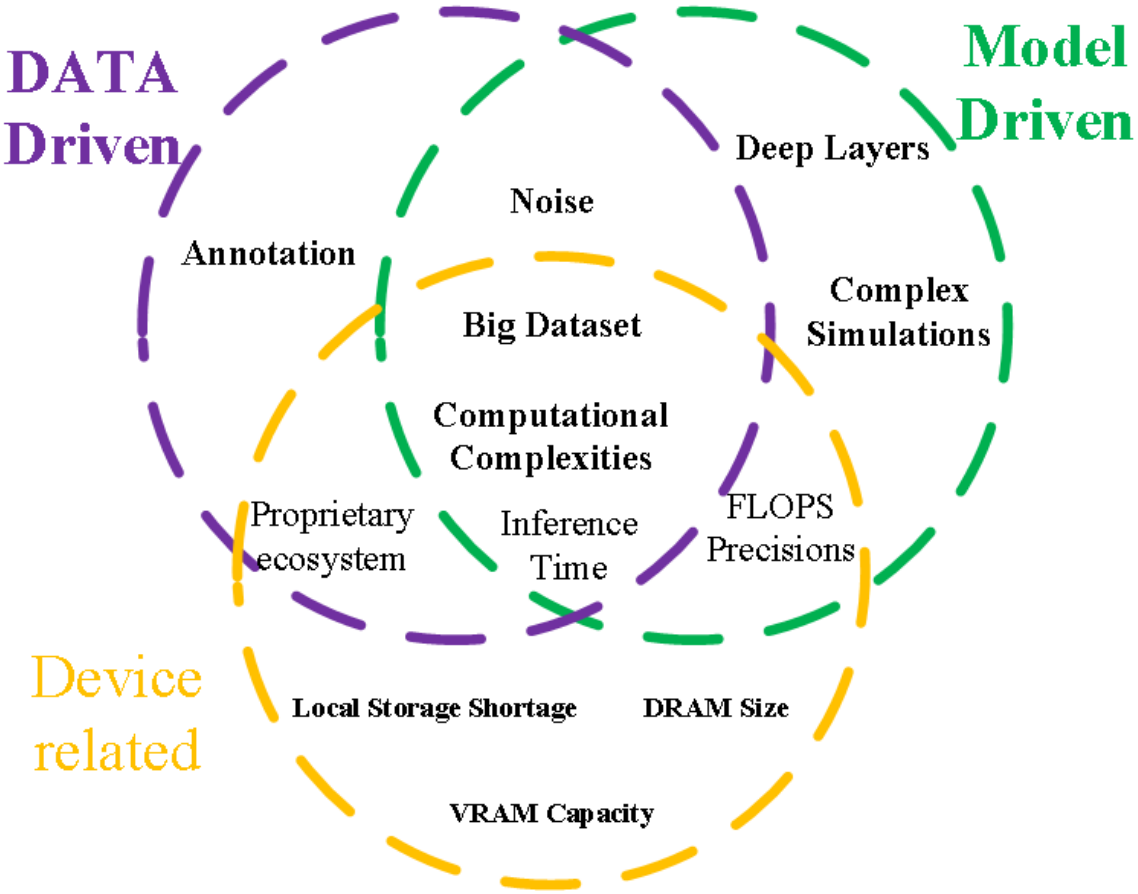

Figure 1.3: Problems Statement

out there (illustrated in Figure 1.4) that can be used to reduce the potentially damaging effects of such changes. These techniques (like pruning, quantization, weight sharing, NAS (Neural Architecture Search), knowledge distillation and low-rank factorization) can help to ensure that any modifications occurring to the architecture of the computer network have a minimal impact on its functionality. it should also be noted that when adapting the model for use in mobile devices, which likely have more limited hardware capabilities, additional challenges must be faced. However, sensor fusion, which combines multiple sources of data in order to provide more comprehensive input to the model, could be used as a way to improve model performance on mobile devices.

In order to design neural networks in a way that is both efficient and robust, sensor fusion and compression are two prominent abilities. Sensor fusion is the process of com-

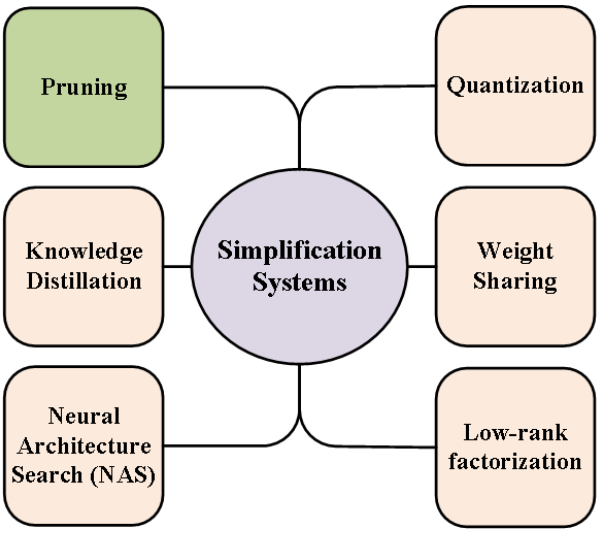

Figure 1.4: Resolution Statement

bining multiple measurements (from sensors) to produce a single more precise and reliable estimate. Compression reduces the amount of data that a neural network needs to process.

1.2.1 Compression

Compression methods can speed up real-time DNNs training and reduce DNNs running costs. One way of reducing the amount of memory used by neural networks is pruning. Pruning is like trimming branches on a plant in order to keep it healthy and strong. Pruning is done by removing and masking weights, channels, filters, layers, or blocks from the architecture. The idea of pruning is to cut away any parameters that are not contributing to the network's results. Doing this can free up memory and space, as well as speed up the processing time.

Quantization is another compression method that can reduce DNNs memory and storage. Quantization reduces weights and activations to 16-bit or 8-bit integers from 32-bit floating points. Smaller mobile devices can then operate more efficiently. Parameter sharing helps DNNs to work efficiently and accelerate training.

If we provide a dataset and goal, a Neural Architecture Search (NAS) system will search for the architecture (classification, regression, etc.). NAS chooses a feasible architecture using a performance-optimized search. Another type of model compression, knowledge

distillation, uses a larger, more accurate model (teacher) to teach a smaller model about the dataset (student). The student can replicate the teacher's probability distribution. Lastly, Low-rank factorization reduces DNNs weight and activation parameters by approximating them in a lower-dimensional space. The objective is to reduce space complexity, speed computation, and improve generalization performance.

1.2.2 Sensor Fusion

Sensor fusion is an AI (Artificial Intelligence) tool that combines data from a variety of sources to make more accurate decisions. These sources could be radar, lidar, and RGB images taken by a camera. The idea is to feed the model with all the available information to provide the most accurate reading possible. Improved precision on the detection devices leads to better decision-making and safer intelligence presence in the vehicles.

When gathering data from different sources, each one has its pros and cons. For example, radar can offer reasonable distance, speed, and depth information. However, because of the noise produced it can be challenging to identify the object accurately. Cameras have the advantage of outperforming others in using colour information for precise object detection. But, the images can be easily disturbed by elements such as light noise, water, or snow. Lidar can be expensive, but it can produce accurate 3d information.

Early sensor fusion combines different inputs before the feature extraction stage is conducted. Mid-level sensor fusion in CNN consists of combining multiple sensing modalities at a higher level than early fusion, closer to the end of the process. This entails combining different layers of a multilayer CNN to obtain a more sophisticated representation of the data that is more expressive and captures more information about the environment. By combining multiple layers, the CNN can gain access to a greater level of detail and become more accurate in recognizing features and learning the data.

1.3 Proposal Points

In this thesis, we tackle the aforementioned complexities using pruning and sensor fusion. In pruning, our focus is on implementing filter and neuron pruning on the instance segmentation task and observing their footprint. We review and summarize the potential problems that could arise in the use of filter pruning on the model and how fine-tuning could enhance the diminishing accuracy in each task. We also explore sensor fusion in the case of mid-level and early fusion. When we use sensor fusion on different modalities, we

investigate both on early fusion model applied to the raw output of the sensors. In order to validate and evaluate mid-level fusion models, we apply fusion to the output of two identical feature extraction models. In pruning, we aim to answer the following questions:

1. In the context of the instance segmentation task, what are the primary distinctions between connection pruning, neuron pruning, filter pruning, and channel pruning?

2. What is the nature of the relationships that exist between different filter pruning algorithms and element selections?

3. How does Fine-tuning improve the precision of a Pruned model?

As well as the following questions in fusion:

1. What are the key distinctions between early and mid-level fusion in this multi-model sensor fusion implementation?

2. In the context of early sensor fusion, how do fusion operators impact the precision of various architectures?

3. Regarding early sensor fusion, What are the computational complexities associated with the implementation of learnable or simple fusion operators?

4. What is the impact of utilizing simple fusion operators versus learnable fusion operators on the process of training?

1.4 Contribution

This thesis has two main contributions: Primarily, we provide new approaches for pruning and fusion and compare different frameworks in the optimization of machine learning simplifications under classification, object detection and instance segmentation. We compared different techniques of pruning and fusion in different datasets.

We have not only developed frameworks that compare their strengths and weaknesses but also investigated different structures and algorithms in the given frameworks. In chapters 3 and 4, our main contributions are the following:

- Develop and implement experimental protocols to assess the efficacy of five pre-existing pruning algorithms on instance segmentation and three filter pruning algorithms on classification models.

- ✣ Conduct a case study to evaluate the efficacy of various algorithms on pruned models and provide valuable insights into the effectiveness of pruning with respect to selected datasets.
- ✣ Perform a case analysis to assess and compare five distinct fusion operators in the context of early and mid-level fusion and enhance their implementation on the 2D object detection models.

1.5 Thesis Outline

In Chapter 2, we review recent work conducted on the evolution of feature extraction, instance segmentation, object detection models, pruning and fusion. The challenges mentioned in the models with respect to the dataset were brought into the equation, and we made it to the literature review on Pruning and Fusion. We provide an in-depth framework for each process that we could elaborate on each part when we dive into the specific task.

In Chapter 3, we provide a comprehensive description of the pruning algorithm and the experimental results associated with the different models we use. There is a discussion on the significance of individual contributions in each sector, as well as a comparison between the various pruning techniques and their respective effectiveness levels.

Chapter 4 concludes with an examination and discussion of experimental findings across models and tasks, focusing on the use of various fusion operators. We examine early and mid-level fusion and demonstrate their effectiveness in object detection. Chapter 5 is a conclusion.

Chapter 2

Literature Review

When neural networks are developed, they are constructed so that they can produce the necessary output by carrying out complex operations on the data sent to them, also known as input. We are able to solve real-world issues like classification, object identification, and segmentation with the assistance of this approach [40]. It is important to note that complex processes are constructed using many specific functions that imitate the surrounding environment to support the complexity of those processes. Therefore, to resolve issues with supervised learning, we must first examine the difficulties experienced and then determine the possible obstacles that may arise along the route.

In the following section 2.1, we will review some recent articles that discuss the development of feature extraction models utilized in CNN for classification. Section 2.2 discusses the differences between the object detection models and the challenges they present. Section 2.3 contrasts the instance segmentation models and their characteristics with the semantic segmentation models. In the following section (2.4), we conclude our in-depth examination of the pruning framework, focusing specifically on its advantages and disadvantages. Finally, in the final section, section 2.5, we detail the fusion mechanism used in recent publications.

2.1 How to extract features using Backbone

CNNs use the convolutional and linear layers in sequence, parallel, and cascade forms to extract features from the image, sounds or data point cloud in the data. These crucial features, which are extracted patterns, could aid the CNNs in classifying them into the same classes or segmenting the related pixel.

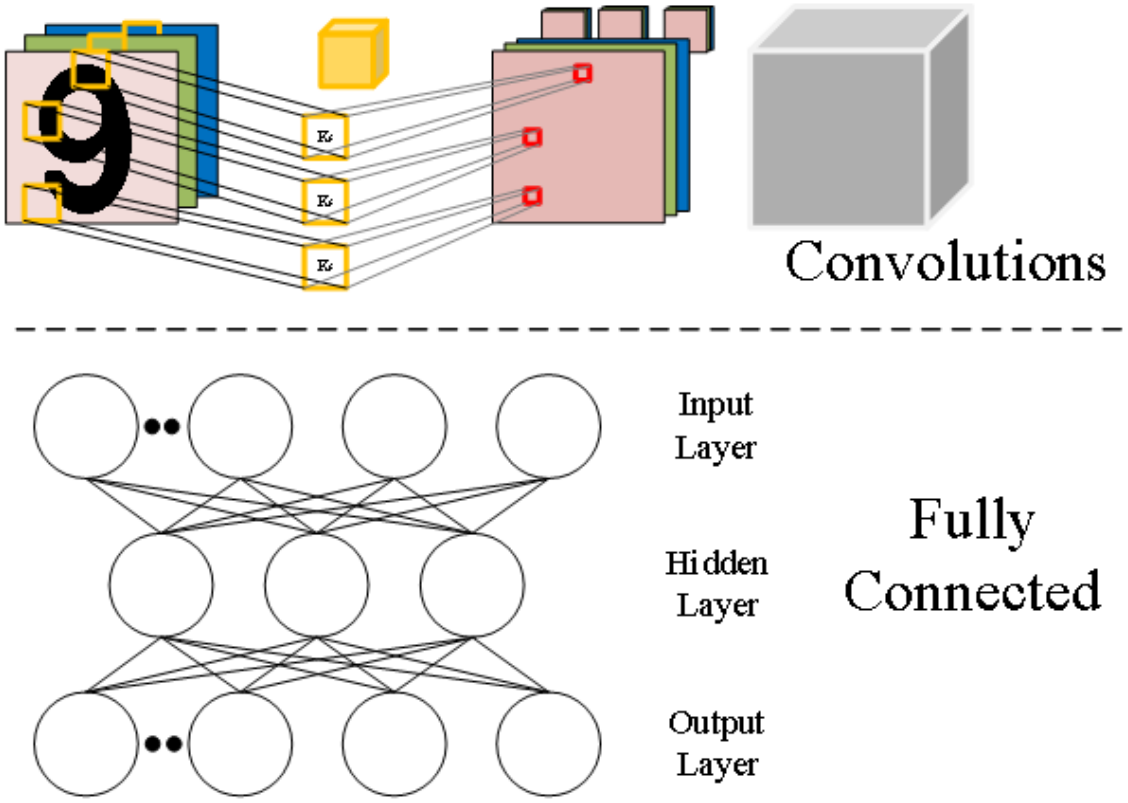

Figure 2.1: Convolutions and fully connected layers operations' structures

Fully connected parts and convolutions are the two primary fundamental components that make up CNNs. In the parts of the estate that are fully connected, which are those in which the connection of each perceptron takes place to every other perceptron, the estate passes on all of the necessary information from one layer to the next. The most significant drawbacks associated with it include the acceptance of only flattened input vectors, the ignorance of spatial information, the elevated levels of inefficient redundancy at large dimensions, and the massive growth of parameters in proportion to the number of layers.

Convolutions, on the other hand, bring spatial focus to the picture by applying the filter as a stamp to all of the features. They reflect both local understanding and pattern recognition based on the shape and stride of the filter. They establish hierarchical and global information about features as they move from one layer to the next. The primary

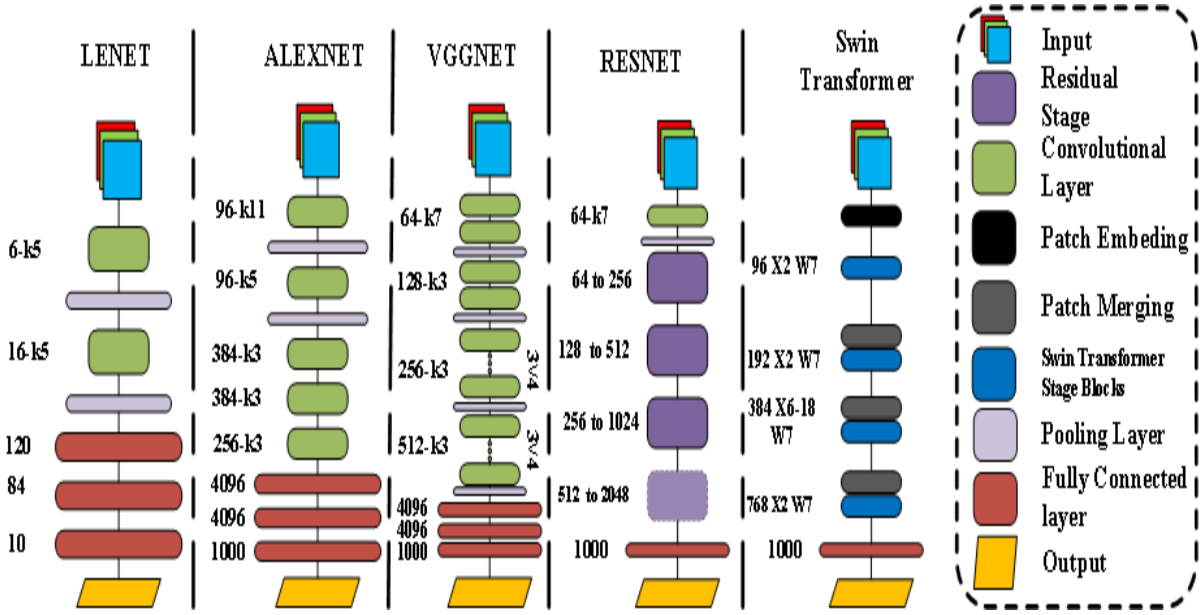

Figure 2.2: The evaluation of each model structure

advantages of these layers are their scalability, their potential for sharing, and the ease with which they can be trained. On the other hand, it has a plain vanishing gradient in deep layers, does not have full connectivity, and has a high parameter count in large filter sizes. The convolutions and fully connected layers operations are depicted in Figure 2.1.

2.1.1 Exploring the Evolution of Backbone Architectures

The details of our comparison of various extractors are presented in Figure 2.9. LeCun et al. in [41] 1998 used Lenet-5 with two convolutional, two subsamples, and two fully connected layers to create a hand-written number classifier. The goal of the classifier was to assign a handwritten number with an image size of 32 by 32 pixels to one of ten categories. Alex Krizhevsky and his colleagues in [36] created AlexNET by using deeper and denser convolutional layers. AlexNET had three different kernel sizes (11, 7, and 3) convolutions, RElu activation, and had been trained with Stochastic Gradient Descent (SGD) using momentum. In VGG, Simonyan and Zisserman [73]increased the number of convolutional layers to 13 and 16 (at VGG-16 and VGG-19) while maintaining a uniform structure and utilizing only a kernel size of 3. The authors Kaming He and colleagues introduce residual blocks that solve vanishing gradients in the deep networks with the skip

connection and by adding recurrent units to the model in the Resnet [26] paper.

The pooling layer is an operation that can take place in between or within the convolutional layers. Its purpose is to lessen the pixel density and the amount of computation required in subsequent layers. The procedures of averaging and maximum pooling reveal the maximum and average of the elements that can be found in the portion of the feature map that is covered by the filter. [46] Networks known as transformers may transform data

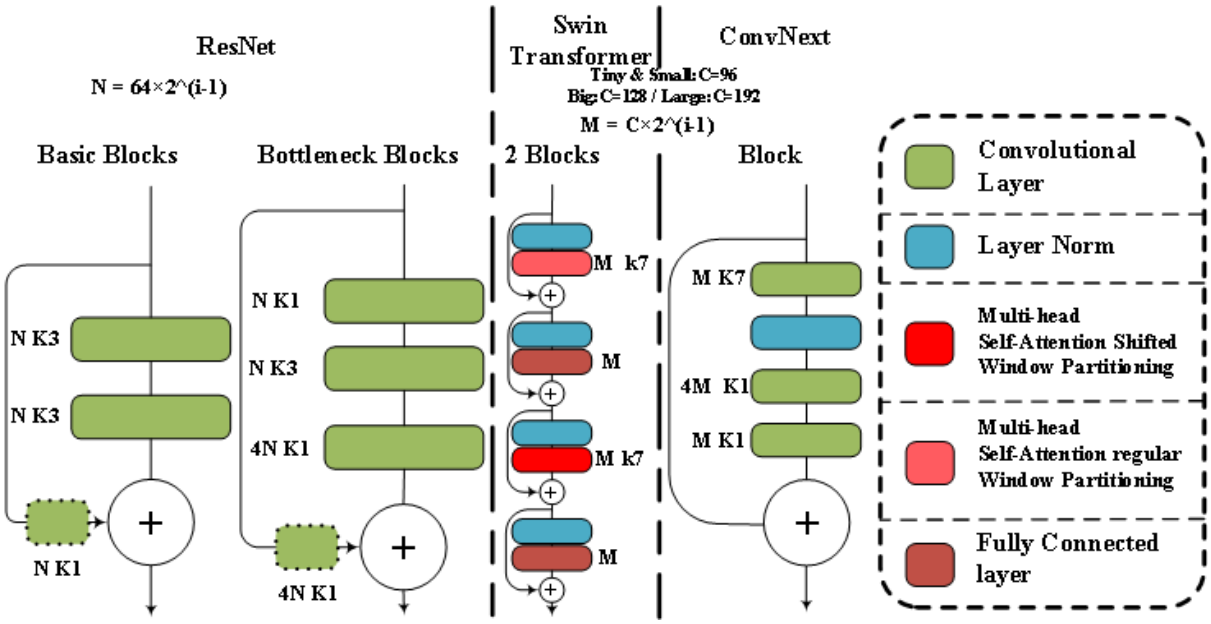

Figure 2.3: The evaluation of each model structure

sequences such as a string of words. These word sequences are tokenized before being supplied into the transformers. The attention that transformers provide is calculated by taking the pairwise inner product of each pair of tokenized words, also known as a quadratic process. The greater the word count, the more steps are required to process it. In the absence of preprocessing, the transformer would be unable to comprehend the spatial relationships among pixels in a given image, thereby leading to incorrect or nonsensical outcomes. The number of pixels in an image might range from a few thousand to several million. In a transformer, each pixel interacts with every other pixel in the picture in a paired fashion. This is an enormous job, even with many GPUs. Image analysis so often employs some kind of local attention (a cluster of pixels) as opposed to global attention.

The issue is addressed by the Vision Transformers (ViT) authors by the use of global attention, albeit not on the full image, but rather on several image portions. To begin,

a large image is sliced into a series of smaller ones. The transformers are first unable to determine which hole should be patched. Therefore, positional embeddings assist the transformer in knowing where each patch belongs. The authors of the paper just used numbers to indicate where the patches should go. They are more than simply numbers, though; they are also teachable vectors. In other words, the value 1 is not employed directly; rather, a lookup table is provided, with vectors for each number reflecting the position of the patch.

The issue is addressed by the ViT authors in [14] by the use of global attention, albeit not on the full image, but rather on several image portions. To begin, a large image is sliced into a series of smaller ones. The transformers are first unable to determine which hole should be patched. Therefore, positional embeddings assist the transformer in knowing where each patch belongs. The authors of the paper just used numbers to indicate where the patches should go. They are more than simply numbers; they are also teachable vectors. In other words, the value 1 is not employed directly; rather, a lookup table is provided, with vectors for each number reflecting the position of the patch.

A little picture (16x16 pixels) is used for the image patch. This must be inputted in a fashion that is comprehensible to the transformer. The picture grid was then flattened to generate a 1D sequence of 256 patches, such that the 2D position of each patch could be mapped to a unique 1D position. However, the linear projection was employed in the actual writing of the article. An embedding matrix, then, is a representation of a single matrix. First, we pick a single patch and unroll it into a straight line. Next, the embedding matrix E is multiplied by this vector. The result, together with the positional embedding, is then sent to the transformer. A transformer encoder receives both the patches and their positional embeddings (after they have been linearly projected). Attention is all that's required for this transformer's design. When it comes down to it, an image's ultimate classification relies on the embedded data's output. Historically, using batch norm for Computer Vision (CV) tasks and layer norm for Natural language processing (NLP) tasks has been the accepted practice. The Attention Is All You Need paper [78] tested only natural language processing tasks, and as a result, layer norm was used. Even though the use of transformers is becoming more common in CV applications, it appears that layer norm is still the most commonly used standard.

Remember that in batch norm, the mean and variance statistics used for normalizing are generated individually for each feature across all elements of all instances in a batch. For an NLP task, "element" and "instance" denote "word" and "sentence," respectively, and "pixel" and "picture" for a CV task. Layernorm, on the other hand, calculates statistics throughout the feature dimension for each element and instance separately (source). In transformers, it is computed individually for each instance across all features and com-

ponents. Figure 3 from this recent post illustrates the difference between batch norm and layer norm.

Patched images, linear layers, and convolutional layers were employed in conjunction with one another by Liu et al. in the construction of the feature-extracting blocks for Swin Transformer [51]. They used a technique that was more hierarchical and embedding size, in addition to an approach that was more local to each patch inside each block. In the paper ConvNeXt [53], Liu et al. used the Swin Transformer and ResNet frame of thinking by maintaining the block method but entirely altering the block structure. They employed a procedure that was more efficient and had fewer linear layers in each block, but they maintained the same broad window size 7 in the convolution layer and the layer norm.

In the ConvNeXt and SwinTransformer, we see using the Layernorm [2] and Gelu [30] activation layer because of more effectiveness in the blocks and handling gradient more smoother.

The initial intention behind handling the data was to put it in the black box and to receive results through this process. Following the introduction of AlexNet and VGG Very Deep Convolutional Networks (VGGNet), the concept has been changed to using deeper layers. However, the vanishing gradient became a problem that arose when introducing a deeper layer. By introducing the residual format of the present, this problem has been resolved. However, by introducing the attention in [78] the idea of self-attention modules arises in works like vision transformer and others to increase employ that feature efficiently. However, they have some problems in the local and general side of the image in that they don't efficiently make for images rather than words. The differences between different blocks could be seen in 2.3 in Swin Transformer the consideration of using shifted and regular window partitioning using the self-attention multi-head brings the transformer idea in classification. In the ConvNeXt, the focus had been changing and using the lower amount of the linear layer and more focus on the convolutional side to modernize the ResNet idea.

The ConvNeXt architecture was based around improving the ResNet architecture to be more similar to the Swin Transformer. This was done by changing various architectural enhancements: the blocks computation ratio in each stage, the stem convolutional layer to a patchify(split images into small patches) non-overlapping patch of four, inverting the bottleneck so that the internal block is the largest and making various convolutional layer changes. Additionally, the ReLU was replaced with GELU in the activation layers and batch normalization was replaced with layer normalization and downsampling layers were added between stages with a layer normalization, a two-by-two convolutional layer with a stride of two and another layer normalization.

As illustrated in Figure 2.4,Layer normalization [2] is a method used to normalize inputs at a given layer across the entire dataset while batch normalization normalizes the inputs at a given layer of a batch. Batch normalization [33] was used in ResNet because it is better at reducing internal covariance shifts that can make training difficult and slow. Layer normalization was used in transformer models because it can reduce the time required for training and allows for larger learning rates and batch sizes. The changes to the architecture resulted in an overall accuracy increase of 3.2 percent and a GFLOPS increase of 0.9 percent. Overall, the various architectural changes provided an attractive solution as ConvNeXt performed better than ResNet while still maintaining a relatively low GFLOPS throughput.

The ConvNeXt paper training process has been extended by 233.33 percent in epochs numbers for ResNets and has implemented a variety of data augmentation and regularization techniques such as Mixup [88], Cutmix [87], RandAugment [10], Random Erasing [90], Stochastic Depth [32] and Label Smoothing [59]. CutMix and MixUp examples help the model improve its generalized ability and also reduce over-reliance on individual features [88] [87]. Random Erasing randomly erases parts of images [90], and Stochastic Depth shortens the network during training [32]. Label Smoothing is a regularization technique that adds noise to the labels [59]. All of these techniques improve the model's accuracy and reliability.

Without a specified assignment, the RELU activation function demonstrates the highest performance. It maintains linearity in positive activations and inhibits negative activations. Leaky-RELU can prevent neurons from "dying off" by preventing activated units in the negative regime from having a gradient of zero. However, powerfully negative activations may still have unintended effects. GELU, on the other hand, is a smoother replacement for Relu because it is differentiable in all ranges and allows for gradients (albeit minor) in the negative ranges. This eliminates 'dying RELU' issues, enabling for enhanced mixing of features between layers. Overall, GELU provides a more uniform gradient in the negative regime and is a more refined alternative to RELU.

2.1.2 Challenges of Backbone Architectures

The presence of noise in the image, insufficient illumination, and the presence of classes with shapes that are too similar to one another are all difficulties that can arise during the classification process. In some instances, like when referring to humans, the relationships between different classes and images that share the same colour spectrum would be quite complicated. Another obstacle to overcome would be the uniqueness of the data that

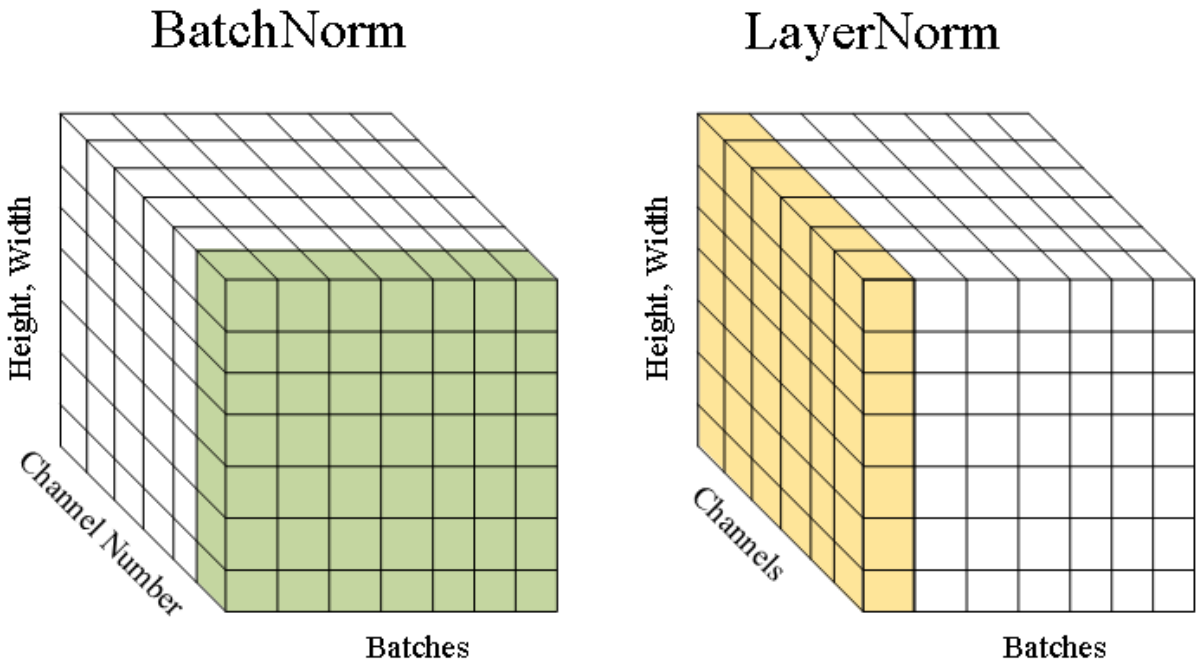

Figure 2.4: The evaluation of each normalization method

results from a limited number of observed images. Even while using the ImageNet dataset, it is difficult to believe that they would be able to see every object from every conceivable angle and location.

It is difficult to justify the reason that all of the parameters in a highly parameterized network with over 20 million parameters are necessary in that case or that all of the parameters play a key role in predicting the probability of an event using a Gaussian distribution when we are confronted with a highly parameterized network that contains such a large number of parameters.

Classification provides various solutions to various inquiries, though it is difficult to find one solution that would address all classifications' challenges. To determine appropriate responses, the problems have to be broken down into manageable parts, for which we need to figure out strategies to increase the ability to recognize objects and sort them into categories, extract primary parameters without damaging the structure, and incorporate supplementary data from data clouds and unmarked terrain.

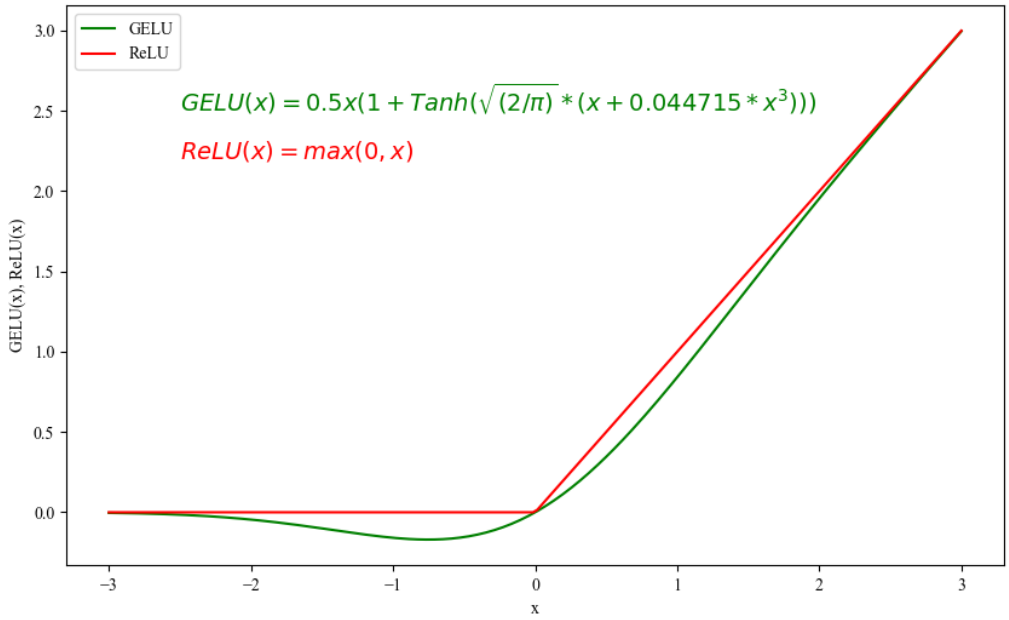

Figure 2.5: GELU vs RELU

2.2 Object Detection

Object detection refers to the process of assigning the bounding box that corresponds to a certain object and then classifying the objects that are defined in the dataset according to their relativity, similarity, and objectness in relation to a particular class or group [89]. This challenge would include localizing objects by identifying the precise centre, breadth, and length; in other words, it would possess the complexity of the problem to be heightened compared to classification as it necessitates the simultaneous resolution of a classification and regression problem.

The accurate identification and differentiation of various objects in an image, despite their similar appearance, partial occlusion by other objects, small objects and image noise, are some primary challenges in object detection and classification tasks. The task can be particularly demanding in intricate scenarios where there exists a multitude of objects that may be overlapping or colliding with one another.

Despite the utilization of sophisticated models, the precise identification and catego-

Table 2.1: Comparison of computational resources requires for feature extraction

	Computation Metrics		Accuracy ImageNet	
Model	Parameters (M)	FLOPS (G)	Top1	Top5
AlexNet	60.97	7.27	63.3	84.6
VGG16	138.4	154.7	74.4	91.9
ResNet-50	25.56	4.12	76.55	93.06
Swin Tiny	28	4.5	81.2	96
ConvNeXt Tiny	28	4.5	82.9	95.86

rization of objects can still pose challenges in particular scenarios, such as instances where objects are small, distant, or substantially obstructed. In order to address these obstacles, supplementary methodologies can be utilized, including but not limited to data augmentation, feature visualization, or fine-tuning the model on particular datasets or domains. Since the beginning of deep learning, the RCNN [21] family has been the preferred method of object detection due to its superior accuracy. The utilization of convolutional neural networks (CNNs) for object detection is predicated upon a number of fundamental principles. Convolutional Neural Networks (CNNs) possess the capability to acquire spatial hierarchies of features, rendering them highly appropriate for object detection of varying shapes and sizes within an image. In addition, the utilization of convolutional layers enables the identification of regional patterns within an image, whereas the pooling layers facilitate the reduction of the spatial dimensionality of the feature maps. Furthermore, the utilization of anchor boxes and non-maximum suppression methods facilitate the accurate localization of entities within an image.

2.2.1 Object Detection Techniques

2.2.1.1 Unifying Feature Extraction with FPN

Lie et al. designed Feature Pyramid Network (FPN) [47] to improve object detection in images. It is based on the idea of building a "pyramid" of features, with the lowest level corresponding to the lowest resolution and increasing in resolution as the pyramid ascends. The connection of the lowest-resolution feature map to the highest-resolution feature map is at the heart of FPN, creating a "top-down pathway" that allows high-resolution information to flow down to low-resolution feature maps. This enables the

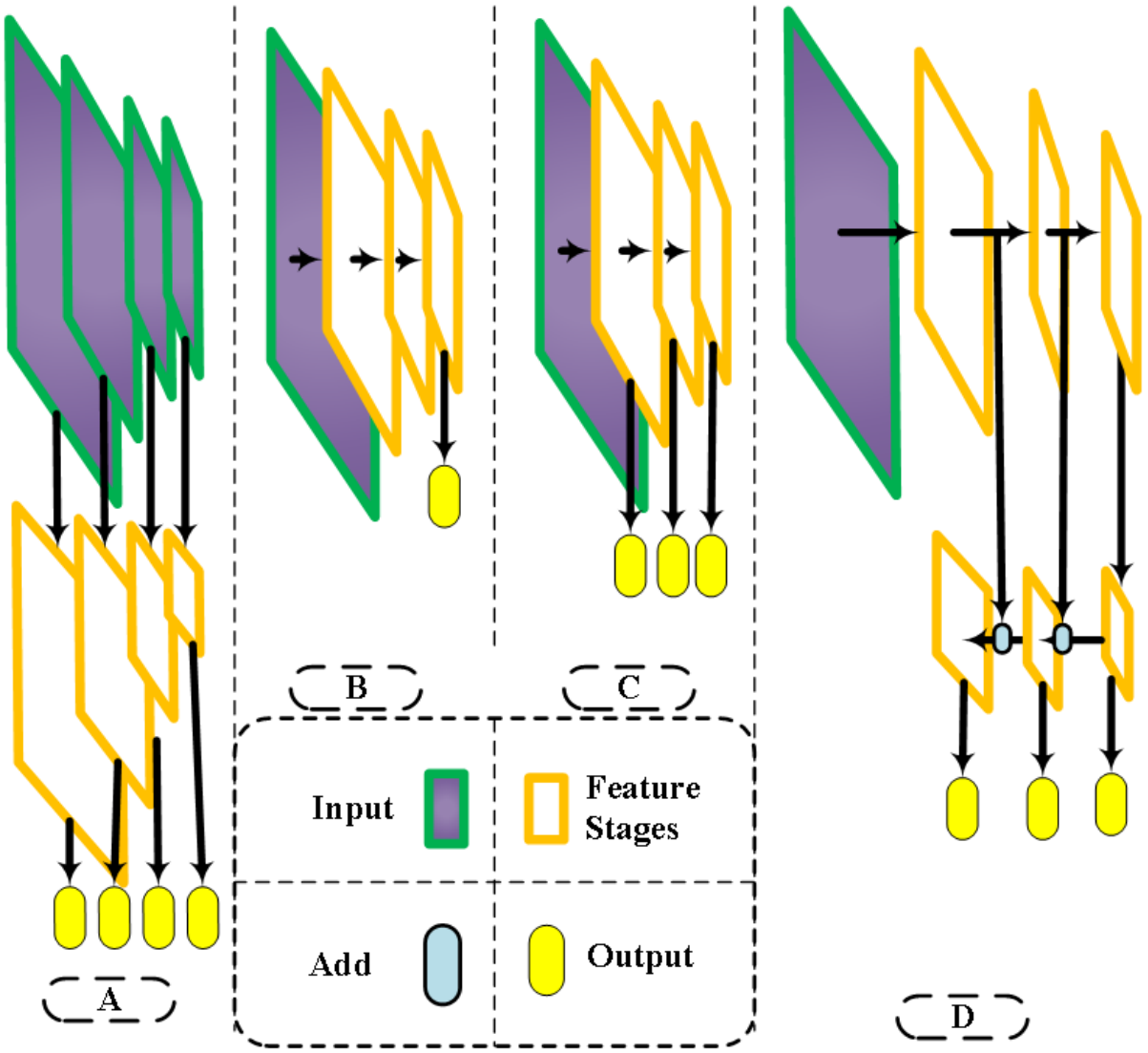

Figure 2.6: Feature Pyramid Network (FPN) [47]

network to have simultaneous access to the same set of objects and features at different scales.

As illustrated in Figure 2.6, creating a feature pyramid from an image pyramid(part 2.6.C) takes time because the features are calculated separately for each resolution of the image. FPN, on the other hand, enables the feature pyramid to be calculated with a single

pass of a ConvNet, which is much faster and more efficient. Previously, detection methods used only one size of the feature(part 2.6.B), sacrificing efficiency and accuracy for speed. The FPN, on the other hand, has a top-down pathway that allows it to merge features of different sizes for increased accuracy, giving it an advantage over the other approaches. Similarly, because the FPN combines different resolution feature maps, the feature pyramid it generates can be used as if it were an image pyramid, eliminating the need for a custom image pyramid construction(part 2.6.A).

The three necks, namely Path Aggregation Network (PAFPN), Weighted Bi-directional Feature Pyramid Network (BiFPN), and Recursive Feature Pyramid (RFP), are extensions of the fundamental Feature Pyramid Network (FPN) concept. Each of the three necks exhibits distinct modifications to the FPN architecture. For instance, PAFPN integrates FPNs with bottom-up path augmentation through its feature pyramid module [50]. BiFPN facilitates efficient multi-scale feature fusion [76]. RFP incorporates altered feedback features from FPN levels to the output of the underlying feature extraction stages, allowing for various perspectives on the pictures [66].

2.2.1.2 Exploring the Object Detection Landscape

In order to make the RCNN family of object detectors more efficient, the Region Proposal Network (RPN) was introduced in Faster RCNN [69]. The RPN has seen some success in speeding up the object localization process; however, it has its limitations too. In an attempt to build on this technology and make it faster, Cascade RCNN [4]was developed. This technique employed multi-cascaded Region of Interest (ROI) heads to reduce the overhead of region proposal generation further. Instead of a single region-proposal phase, it uses a cascade of region-proposal networks, allowing for more region proposals with higher accuracy. Additionally, Cascade RCNN uses a detection pipeline with two stages of boxes refinement. This improves the accuracy and precision of detection results since more iterations allow more chances to refine the boxes. The trade-off is the speed of the algorithm. Since it has more layers and more steps, it tends to be slower than Faster RCNN, but with increased accuracy.

In an effort to make object detection even faster and more efficient, YOLO (You Only Look Once) [68] was developed as an alternative to the RCNN family. From version one to seven, it's gone through various iterations, employing different backbones, making it smaller and anchorless, and making it a one-stage, end-to-end prediction method. Its accuracy is still not at the same level as the SOTA competitors, but the progress it's made since its inception has been remarkable.

2.2.2 Performance Evaluation in Object Detection

Metrics in object detection performance assessment provide a way to evaluate a model's efficacy and efficiency by quantifying how well it performs at detecting objects. Intersection Over Union, Average Precision, Average Recall, and the Difficulty Factor are some of the most used metrics for measuring the effectiveness of object detecting systems.

The Intersection over Union (IOU) metric is frequently employed in the assessment of spatial overlap between two regions. It is often represented as a percentage and is found by dividing the area of their intersection by the area of their union. The IOU metric is used to evaluate the precision with which an item is spotted in a picture. To determine AP, we divide the number of TP (true positive detections) by the sum of all detections (true positives plus false positives). The AP value represents the proportion of accurate identifications. The number of successfully recognized items (TP+FN) divided by the entire quantity of objects (TP+FP+FN) yields the Average Recall measure, which is mostly used for multiple object identification. It is a metric for determining a model's accuracy in identifying objects. The complexity and difficulty of an object's detection and identification are measured by its "difficulty factor" metrics. Higher accuracy and recall metrics are often needed for more complicated and demanding object identification models to achieve reliable detection. The statistical measure of Average Precision is deemed superior for optimization in instance segmentation due to its ability to consider false negatives. This feature is particularly crucial in the process of pruning segmentation models and guaranteeing the model's precision [19].

2.3 Instance Segmentation

Semantic segmentation is the detection of objects based on class and assigning pixels to a specific class of object without considering the importance of individual units in another word, we assign a class to pixels, and we don't separate them from the same class instances in the picture. In Instance Segmentation, we not only detect the pixel for each class, but we emphasize their differences to find the optimal category in that section.
Semantic segmentation necessitates the use of two techniques: efficiency and precision segmentation. Efficiency refers to the cost of the segmentation method, while precision refers to the accuracy of item localization and recognition. Memory and storage requirements, power and processing usage, model complexity, time to train and interface, and preprocessing affect segmentation efficiency. The precision of segmentation is affected by object size (particularly small objects), noise, crowd size, annotation, feature representation, geo-

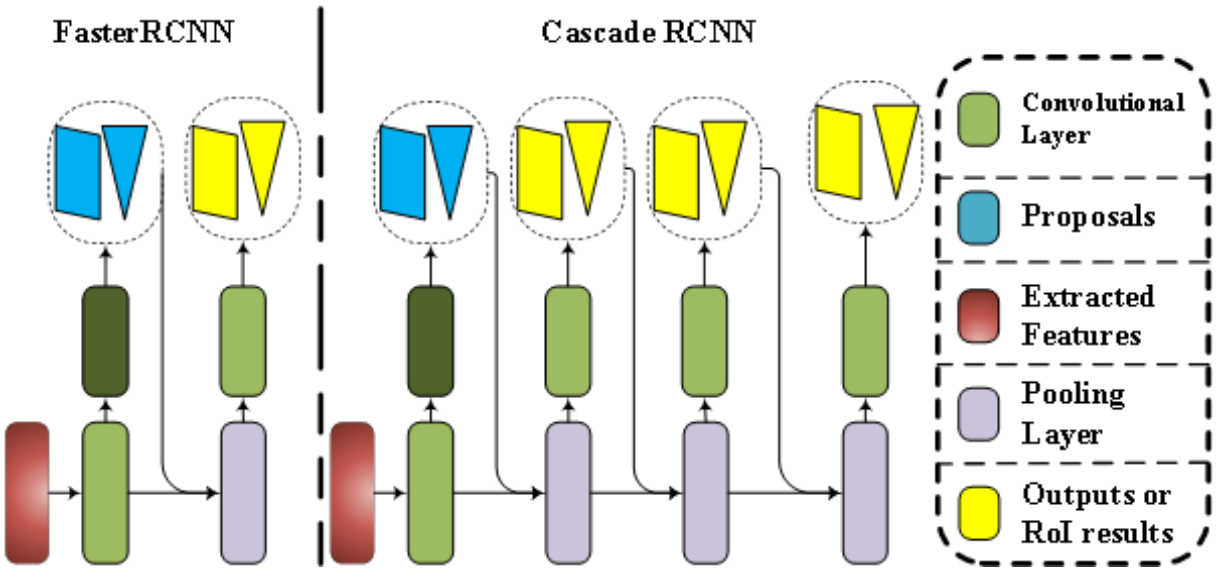

Figure 2.7: The evaluation of object detection models structure

metric transformations, handling of occlusions, and handling of picture degradations. [22]. In Instance Segmentation, Generous scopes of complications from semantic segmentation intrude with the object detection problems. The COCO dataset is a widely utilized resource for instance segmentation, comprising a total of 118,000 training images and 5,000 validation images, each accompanied by pixel-level segmentation masks [48].
To update box detection by segmenting the critical pixels of each object, He et al. created MRCNN [24] by integrating the various layers of Convolutional operations at the end of Faster RCNN. TheROIAlign misalignment solution and the bounding box region proposal network could provide helpful information to the final FCN. For example, the Final mask could employ spatial location extracted by features and bounding box prediction by the proposal. Nonetheless, this decoupling between the mask and bounding box branches occurs after ROI aligns, reducing competition and potential difficulties between the box and mask detection and allowing the FCN to execute a more accurate classification of each class. In the generalization step of feature extraction, Imagenet pretraining in the current model would enforce a definitive answer. Ultimately, we will still include classification, segmentation, and box detection. Bolya et al. in Yolact [3] found the solution by merging the prototyping masks and their coefficients linearly, dividing them into two convolutional-based branches and uniting them after non-max suppression. Their approach came from a naive understanding of box from the picture to employ RetinaNet. It's worth mentioning

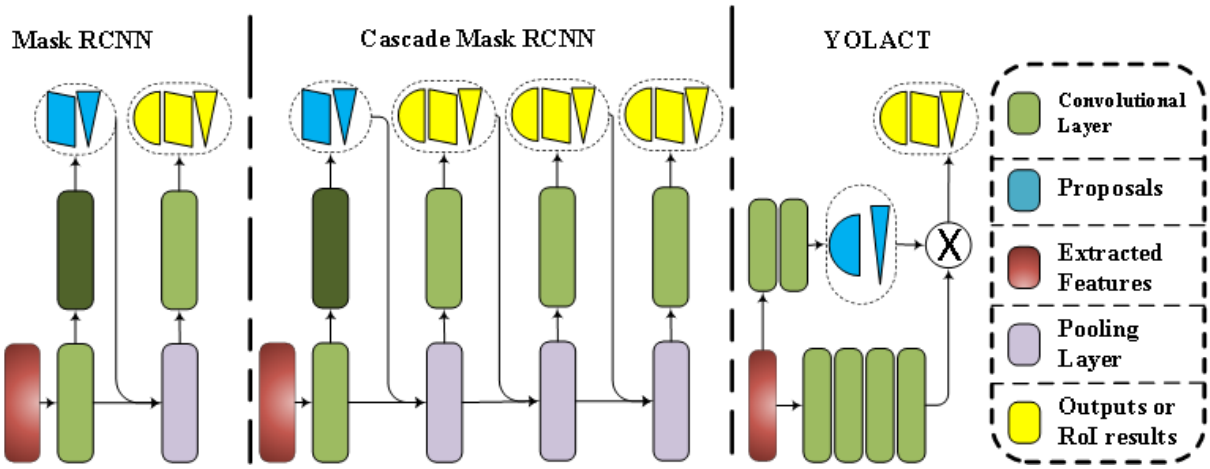

Figure 2.8: The evaluation of object segmentation models structure

that solo [80] and solov2 [81] solve this problem similar to the Yolact but without making the prototype, but they used the unified and high-resolution mask feature representation and combined them in more convolutional layers. In their recent work, they used a deconvolutional layer and modified the NMS algorithm. The Cascade [4] and HTC [5] are bringing the cascade idea from the feature extraction into the detection that manifests to more accrue and high computation cost. The cascade MaskRCNN (MRCNN) uses the same method as CascadeRCNN to Faster-RCNN. In the HTC, they reinforce semantic features as crucial as bounding box features, so they as another branch and used multiple convolutions to solve this in that branch. They are at the top of the line in accuracy and have worse throughput(image/sec) than MRCNN.

2.4 Pruning

Pruning tends to remove redundant parameters in different scales from the DNNs. This process usually takes place afterwards the Training phase and could be modified based on the central policies. Accuracy, speed, and sparsity would alter following the decision, search algorithm and removing the redundant sections from the architecture. Not only does trimming the network make a min-max problem for fine-tuning, but also it generates a divide-and-conquer situation in the searching redundancy. In Figure 1, The framework of the Pruning algorithm consists of three sections and seven potential segments. After appointing the objective of pruning, the details of pruning and tuning nodes would be

determined. Level, algorithm and criteria of trimming and fine-tuning and Decision making of pruning are other influenceable departments. [61]

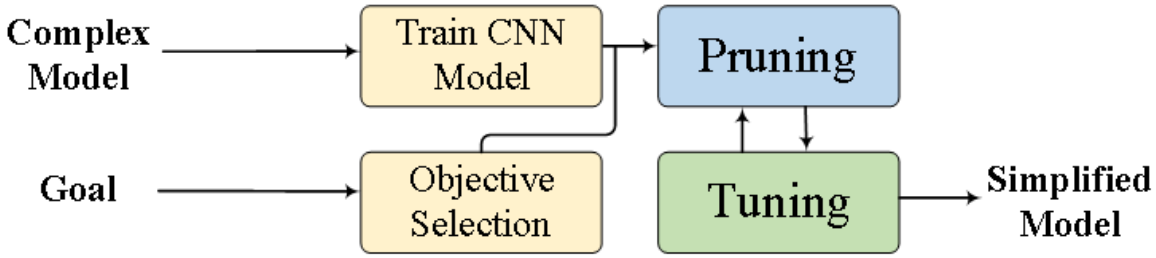

Figure 2.9: Framework of pruning models [61]

Before deep diving in the pruning history, Deep and "thin" networks are also suggested to be trained for a suitable balance between speed and accuracy(He et al. (2015) [25], Remero et al. (2014) [70]). Along with cutting down on running time, a related problem addressing memory preservation has drawn attention [9].

The brain mechanism in synaptic pruning could be manifested in NN, where they could heal and restore tolerable impair in neurons; Le Cun et al. initially proposed a system of Optimal Brain Damage [42] which employs a theoretically-accurate assessment of prominence to trim weights. Hassibi and Stork (1993) proposed the Optimal Brain Surgeon [23], which used second-order derivative information of every weight in the network for determining trimmable unimportant weights. In diversity networks(2016), Mariet and Sra proposed the compression method that extracts the sub-module from the diverse neurons in the network that doesn't require fine-tuning and retraining [56]. This approach employed only fully connected layers and made sparse connections.

2.4.1 Pruning Components

Pruning components refer to the various aspects of the pruning process that can be designed to enhance the results produced by the pruning procedure. It is possible to tailor the process of pruning to meet the requirements of a specific application better. The pruning algorithm, the pruning kernel, the pruning criteria, and the pruning techniques are all examples of components that have the potential to undergo modification.

The pruning kernel is the set of techniques used to reduce the complexity of a trained predictive model by removing unimportant weights or features. The pruning criteria are

used to assess the relevance of each weight or feature to the model's performance during the pruning process. The pruning techniques are the algorithms and methods used to implement the pruning process based on the criteria determined.

For instance, modifying the criteria for pruning could be helpful in bringing down the total amount of pruning that is required to achieve the desired effect. Altering the pruning algorithm might also help to accelerate the process and further reduce the amount of pruning that is required. In addition, altering the pruning kernel may make the process more precisely tailored to the requirements of individual applications. In the end, modifying the different aspects of the pruning process can help to improve the results of pruning.

2.4.1.1 Pruning Level

The level of pruning that takes place in a neural network is determined by the aim that is being pursued. Techniques for pruning might range from making slight adjustments to the network parameters to effecting significant structural alterations. In certain approaches, the removal of individual parameters or neurons might be necessary, whilst in others, the breakdown of larger layers into more manageable ones might be required. Merging layers or merging filters between layers is another strategy that can be utilized in the pruning process. There is a correlation between the level of pruning and the level of performance and computing capabilities it provides.

The process of shrinking the size of a neural network by deleting the weights that aren't necessary is referred to as unstructured network pruning. This method is used to develop neural networks that are more compact and efficient, with the potential to have higher levels of accuracy and reduced memory requirements. In most cases, the method entails examining the significance of the network weights and deleting the weights that have the least influence on the performance of the model. It is possible to cut the size of the model by as much as ninety percent by unstructured pruning.

The goal of structured pruning is to obtain a better balance between the number of computational resources used and the accuracy of parameter validation. This is accomplished by pruning the network in accordance with predefined patterns and repeating structures, such as channels or layers. Pattern recognition and optimization are both involved in the process of structured pruning, which differs from random pruning in that it follows specific patterns rather than cutting branches at random. It is possible to considerably cut down on the total number of parameters when using this approach by getting rid of entire channels or filters, as well as weight connections. This particular pruning method produces more accurate models despite having fewer parameters. Structured pruning, as opposed to

unstructured pruning, places a limit on the degree of sparsity that may be imposed on a network.

A number of different components, such as nodes, connections, channels, filters, layers, and blocks, are all candidates for elimination [61]. In the PFEC [44] and GREG [79] methods for pruning filters, or removing unnecessary filters, from a CNN are proposed.In CP [29], DCP [94] and FISHER [49] methods are proposed to reduce the number of channels in the Convolutional layer. In [64] and [85] pruning is achieved by reducing the number of connections.

Achieving the desired result is the sole determinant of the pruning strategy. Reducing the number of parameters in a neural network without sacrificing accuracy can be accomplished through connection, channel, and filter pruning. Block pruning is more stringent than other forms of pruning and can significantly reduce the number of parameters in a model, along with the time and effort required to run it. As a result, any of these methods can be used, depending on the relative importance placed on precision, model size, and computational expense.

2.4.1.2 Element Selection Principles

When picking pruning elements for a CNN, some of the primary parameters that we take into consideration are the filter size, the channel numbers, and the layer activations. When it comes to pruning the filter size, the quality of a pruning result can be evaluated based on the Gflop reduction and the percentage of difference in filter size to see which one can eliminate more without compromising the accuracy of the model. In order to prune the number of channels, we may first compare the number of channels present in each layer. Then, we can eliminate the channels that have poor performance and so lower the number of parameters as well as the computational complexity of the model. We have the option of using a weight pruning strategy in order to prune layer activations. With this strategy, we may arbitrarily choose a particular percentage or number of weights to be pruned and then analyze how this selection impacts the accuracy of the model.

These criteria are outlined in further detail in Figure 2.11. When it comes to pruning, the absolute value of the connection weights [42] and the L1-norm [44] are two of the most common criteria applied. In [28] and [43], a new set of pruning criteria is presented that demonstrates improved performance compared to the criteria used in earlier research. In [1], the criteria for trimming are applied to the context of medical image applications and put through a series of tests and evaluations. In [71], an altered version of the Linear Discriminant Analysis (LDA) is used in order to select the superfluous filters. For the

purpose of channel selection, Luo et al. [55] utilized L1-norm in conjunction with reconstruction error minimization. The method described in [31], which relies on absolute values of the weight, produces the superior result. Entropy, input and output norm, and Singular Value Decomposition (SVD) were all utilized in the process of node selection by He and colleagues [27]. In order to choose channels and nodes, Liu et al. [52] proposed the use of a scaling factor and a sparsity-induced penalty in addition to the L1 norm.

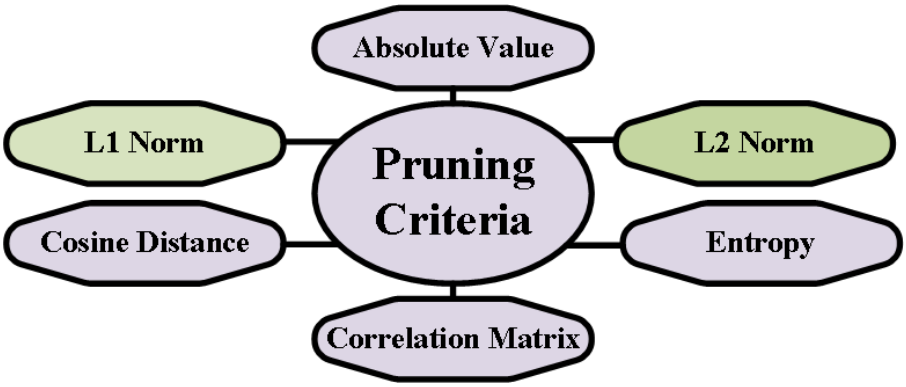

Figure 2.10: Pruning Principles [61]

The Frobenius norm was used to prune channels in He et al. work CP [29]. Using the weights' magnitudes, a binary mask indicating pruning elements is generated in [93]. New pruning criteria were defined by Singh et al. [74] using entropy and cosine similarity. Cosine distance is used to eliminate filters that aren't crucial in [28]. Each layer's output norm is then used to fine-tune the parameters of lower-level layers, as shown in [13]. By changing the shape of the Hessian matrix, a pruning binary mask is obtained, and this is how a pairwise correlation matrix is defined in [63].

In general, L1 norm produces more efficient and sparser solutions than L2. The L1 regularization performs better when many elements are substantially bigger than the others within the same feature map or kernel. The L2 regularization, on the other hand, performs best when the elementwise magnitudes are all roughly the same or when there are just a few elements with a much larger magnitude. The L1 regularization's capacity to significantly punish large element-wise magnitudes makes it a superior choice for the filter pruning problem, yielding quicker and sparser solutions.

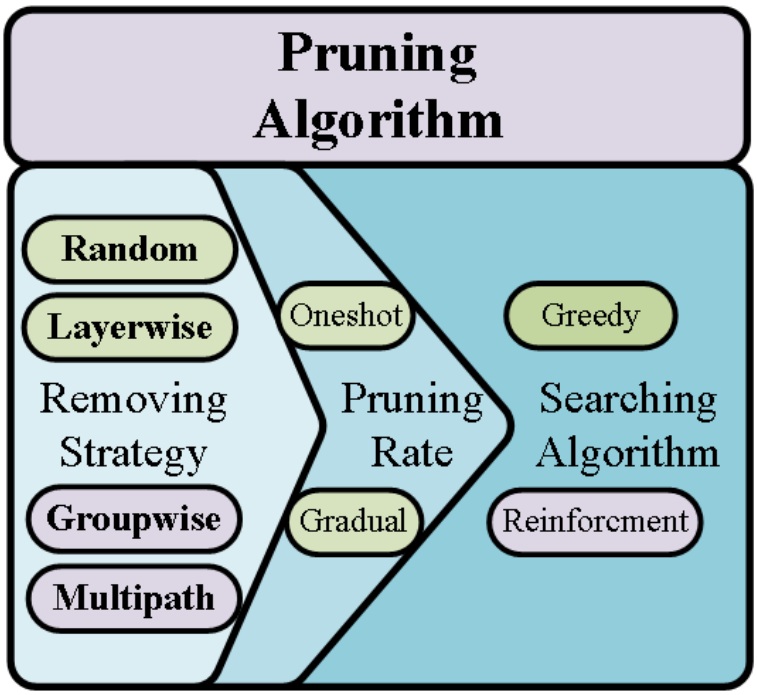

Figure 2.11: Pruning Algorithms [61]

2.4.1.3 Pruning Algorithm

Pruning algorithms are characterized by the element removal strategy used, the implemented search strategy to find the element to be removed, and the efficiency or speed of the procedure. The removing strategy, searching algorithm, and pruning speed are the main elements of the pruning algorithm. The pruning policy outlines the process for removing components from the model. The pruning technique takes into account the granularities, location, and sequence of element removal.

Granularities refer to the level of detail at which the pruning algorithm operates. The algorithm may remove individual weights or neurons or remove entire layers or blocks of the model. The granularity of the removal process significantly impacts the final result and should be chosen based on the specific requirements of the task at hand. A pruning algorithm may start by removing the small branches or components and work its way up, eventually removing bigger pieces. This way, the algorithm can effectively remove larger sections of the structure without considering every individual part.

Location refers to where the elements are removed from within the model. In this case,

the algorithm may remove elements from the model's beginning, middle, or end or remove elements randomly or in a targeted fashion. The location of element removal can also greatly impact the final result and should be chosen carefully.

The sequence of element removal refers to the order in which elements are removed from the model. For example, the algorithm may start by removing elements from the beginning of the model or prioritize elements deemed less important for the task. The sequence of element removal can greatly impact the final result, and different algorithms may prioritize different sequences. [61]. The timing and frequency of the implementations of gradual and one-shot pruning are different from one another. When performing gradual pruning, pruning happens at a slow rate and performs the task more regularly when training the network with multiple iterations between pruning sessions. In practice, this means that more weights are gradually eliminated while the neural network is given the opportunity to adapt and change its structure. On the other hand, one-shot pruning refers to the practice of performing multiple cuts in a single session. This eliminates a large number of weights all at once, but it also involves a significant amount of computation and there is a possibility that the neural network will not respond in the correct manner. Lebedev and Lempitsky (2016) and Zhou et al. (2016) approach in filter pruning modified regularization on neurons to represent group-sparse regularization in the middle of training [39] [92].

The major downside of PFEC [44] and GREG2 [79] is that they are unsuitable in networks with high neuron percentages because of more focus on Filter pruning rather than whole network simplification. More than 32 percent of parameters in segmentation models that heavily rely on the neural part could not be improved significantly by removing only filters.

The impacts of pruning at various rates will likewise vary in their manifestations. When the pruning rate is increased, more weights are eliminated, which ultimately results in a more dramatic alteration. This may lead to improved accuracy, but it also has the potential to cause overfitting. When the pruning rate is lowered, a smaller number of weights are eliminated, and the shift is less dramatic. This could result in a more consistent improvement in accuracy, but it could also result in a lower maximum accuracy. In the end, the dataset as well as the architecture of the CNN will determine which pruning rate is optimal.

2.4.1.4 Tuning and Further Decisions

After pruning, a model's accuracy may suffer because this process frequently causes a reduction in the model's capacity to understand and represent complex information, as

illustrated in figure 2.12. The removal of parameters can constrain the capacity of image analysis to identify a diverse array of visual characteristics, encompassing contours, patterns, and configurations. This may lead to a deterioration in the precision of image recognition and detection as a result of several factors, including the reduction of the network's potential to encompass a broad range of patterns in the data, a limitation of the model's capacity to extrapolate to novel data, and the reduced ability to manage data that is either incomplete or noisy. In other words, removing parameters from a model can make it less robust, which may result in the introduction of bias or overfitting of the model to the data used for training. In addition, even if a model is trained with a regularization technique, pruning may cause the model to fail to converge to the same regularized solution, which will result in a reduction in the accuracy of the model. The next step, after

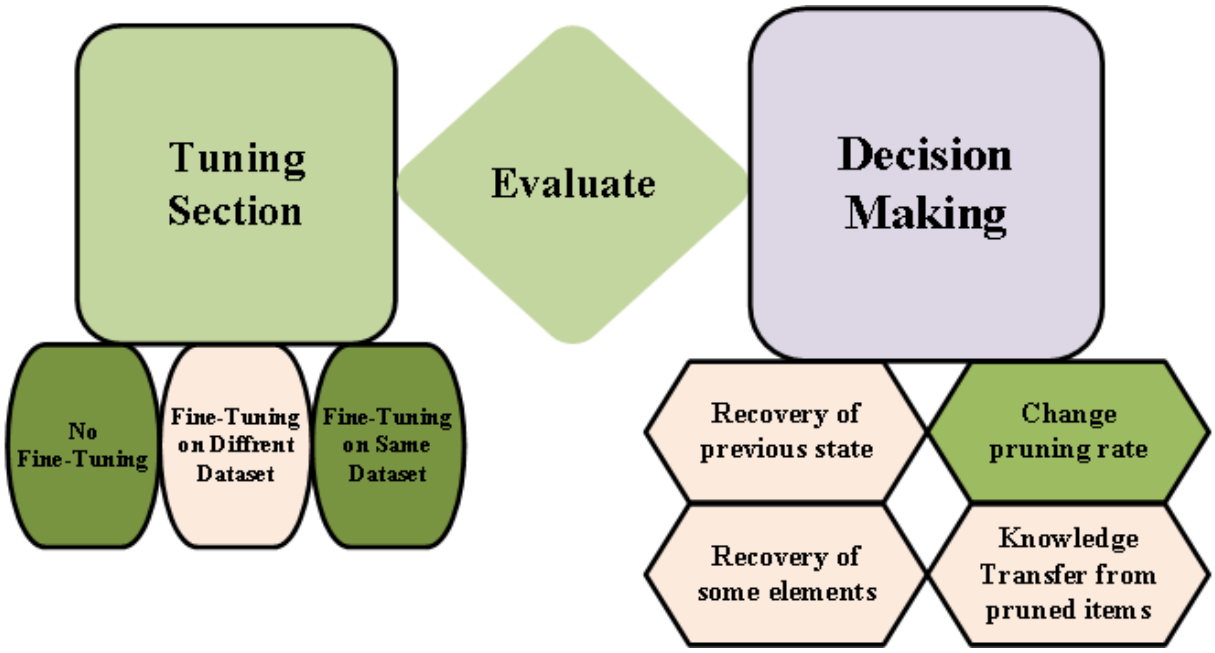

Figure 2.12: Decision making [61]

pruning, is to adjust the model's hyperparameters. This can be done by changing the learning rate, the number of epochs, the size of the batches, and other parameters. Also, it's important to test how well the model performs with different datasets to see how well it works with different kinds of inputs and tasks. If the model's accuracy was satisfactory, more decisions may need to be made, such as adding regularization or adding layers or neurons to the model. To make sure the model is tuned correctly, it is also important to

think about hyperparameter optimization techniques like cross-validation.

Following structural pruning, a decrease in accuracy is expected, necessitating the implementation of corrective measures known as Tuning. In PFEC [44], GREG [79] and PUFA [58] employed fine-tuning by training on the same dataset after pruning as the main approach. They suggested that pruned architectures can be trained on the same dataset for multiple epochs until fully converged, as post-pruning fine-tuning can increase accuracy and decrease training instability. In [34] proposed merging neurons with their neighbouring neurons as the major way to reduce the computational cost and improve model capacity at the same time.

The process of neuron merging entails replacing several neurons with a single neuron with randomly initialized weights. The fine-tuning process consisted of re-training the network after it had been pruned using layer-wise learning rate schedules. Compared to a network that had not been subjected to any training, we discovered that both neuron merging and fine-tuning led to increased accuracy. Regardless, The expected increase in accuracy using neuron merging may not be significant enough to be reported on the results table, as this gain would depend on the complexity of the network before pruning. In particular, the process of fine-tuning produced the greatest outcomes since it was able to maintain the network's original design the best. According to the evidence presented here, fine-tuning seems to be the most effective method for tweaking pruned networks.

2.5 Sensor Fusion

In Sensor Fusion main focus would be around adding more data and minimal change in the CNN structure. As using lidar and camera sensors framework used in MV3d [8] where lidar sensory projection in frontal view made performance enhancement. Region proposal generation's input and output go through the fusion that initially extracted features with different modalities and finally generated proposals with different extracted features modalities. The element-wise fusion mechanism effectively illustrated superior in classification and regression than no fusion.

The depth and RGB data were employed by FrustumPointNets [65] for extracting the precise location of objects in FrustrumPointNet that initiated segmenting 3d frustum using the 2d Regin proposal and depth knowledge. Value of points mentioned again in PointFusion [82] using camera and lidar data as employed 2d detector for RGB image and fusion with 3d point cloud from processed from raw point cloud data. AVOD [37] not only used the same mindset but also put an element-wise mean operation fusion mechanism

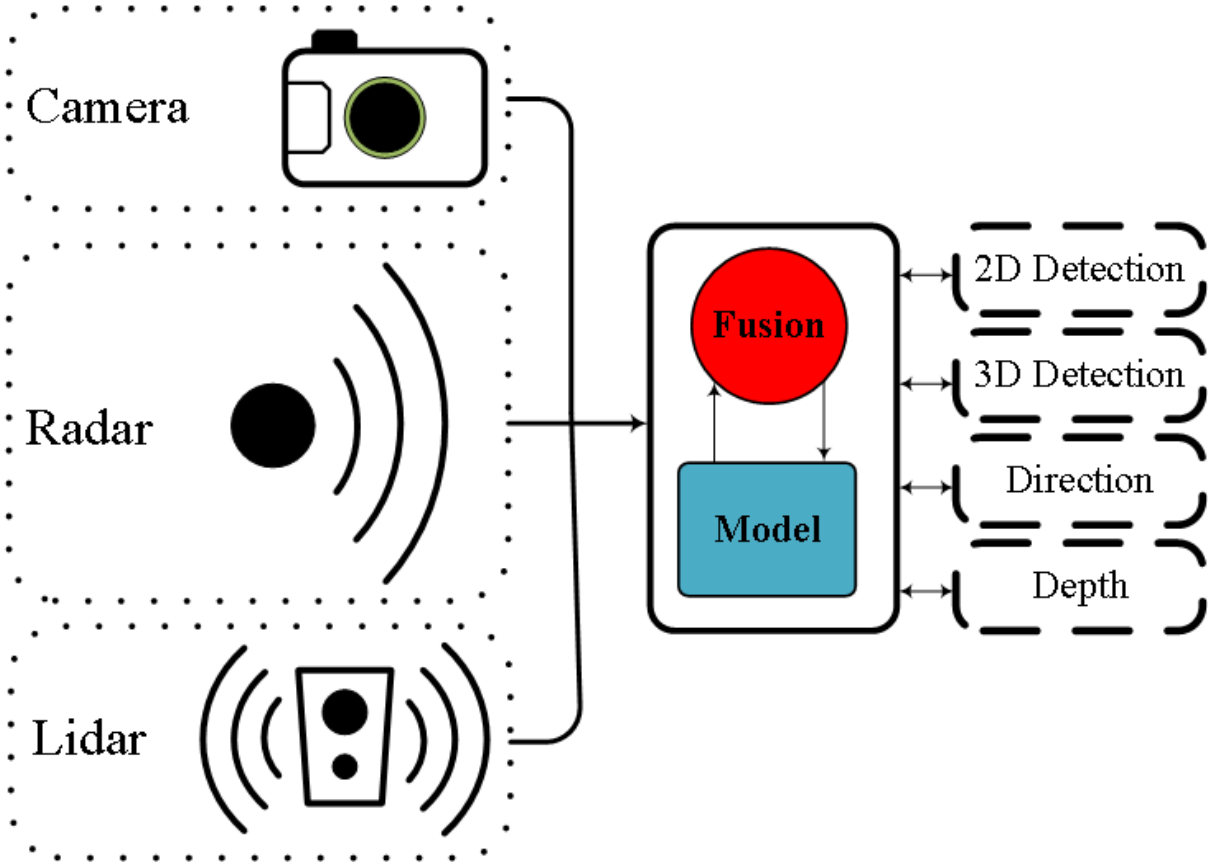

Figure 2.13: Multi-Sensor Fusion Framework

serially to fully connected layers modality-wise. At ContFuse [45], fusion(element-wise summation) happened from Frontal View(FV) features projection into bird eye view(BEV) and BEV extracted features.

2.5.1 Tuning Sensor Fusion for Enhanced Ecosystems

As illustrated in Figure 2.13, The models and operators of sensor fusion make use of a variety of sensors, such as cameras and lidar, to generate a virtual environment that takes into account the data from all of the sensors. Utilizing this virtual environment, we can accurately detect and categorize the objects that are present in the scene. Schuster et al. proposed a fusion technique employing multiple frames to improve scene flow estimates,

resulting in significant improvements for dual-frame algorithms and in substantial improvements for dual-frame algorithms and providing additional occlusion map estimates [72]. Objects in both the real world and in a virtual 3D environment can be located using a combination of 2D and 3D object detection [82] [37]. The robot is able to make decisions and interact with its surroundings thanks to a technique known as 3D semantic segmentation, which classifies the various objects it encounters in its surroundings [77]. The saliency detection process helps the robot understand what aspects of the environment it should be paying attention to by identifying which objects in the scene are important [75]. All of this information is compiled into a coherent understanding of the surrounding environment, which enables the robot to navigate and interact with it in a risk-free manner.

The goal of sensor fusion is to improve upon the results of previous measurements by combining data from multiple sensors. However, fusion can be tricky because of input inconsistencies. The complexity is amplified by the heterogeneous measurement accuracy profile across sources, the inherent uncertainty in the models, and the need for extensive preprocessing of the raw data. Achieving effective fusion requires first making sure that all data sources provide similar data types and resolutions. The data integration schemes and model parameterization of each source are also improved. Noise, such as missing or incorrect data, could be filtered out with the help of smoothing and filtering techniques. Finally, real-time performance is critical for keeping the fusion data current with environmental changes and improving accuracy. In other words, Sensor Fusion faces difficulties, such as projection issues that could not understand the image and feature loss, adjusting the parameter size or size of the network, normalization, noise in the picture, and certain calculation errors in the transformation [91].

2.5.2 Sensor Fusion Applications

Sensor fusion models have the potential to enhance both the security and the performance of trains. Sensors such as radar, cameras, and lidar can be used to locate potential obstructions that could prevent trains from moving forward. The sensor fusion model can then use the data from these sensors to provide real-time information on the track and nearby objects, which enables the onboard computer to adjust the speed accordingly and execute emergency stops as necessary [17].

By providing increased situational awareness to unmanned aerial vehicles (UAVs), sensor fusion models can be used to make UAV flights safer. Sensors like gyros, GPS, cameras, radar, and other similar devices can be used to detect objects in the environment and track the movements of other unmanned aerial vehicles (UAVs). Onboard computers can then

use this real-time information to help avoid collisions, maintain formation, and navigate through the environment more effectively [15].

The navigation and motion planning of unmanned underwater vehicles (UUVs) can be improved through the use of sensor fusion models. The model is able to create an in-depth and comprehensive map of the surrounding environment and identify the obstacles and paths by combining the data from a variety of sources, including cameras, sonar, laser sensors, pressure, magnetometers, and acoustic positioning data [60].

Sensor fusion models can be used to give industrial robots the ability to recognize objects and adapt their behaviour accordingly [35]. This ability allows the robots to perform their tasks more effectively. The robot is able to identify objects in its surroundings and respond to them in real-time by utilizing a variety of sensors, including vision, radar, and acoustic range finders. This could be put to use for a variety of tasks, such as loading and unloading pallets, picking up and assembling products, etc.

Sensor fusion models are an essential component of autonomous driving and have the potential to improve the overall road safety of vehicles through their application. Real-time signals can be transmitted to the vehicle's onboard computer from a variety of sensors, including cameras, radar, and lidar, which can be used to detect obstacles and other vehicles [83]. The onboard computer will then make the necessary adjustments to the vehicle's speed, direction, and other behaviours based on the information it has received.

Chapter 3

Compression using Pruning

In the field of CNN, one important technique is called "neural network pruning." Pruning is the process of reducing the number of parameters in a network, and it is commonly used to improve the performance of networks. Pruning helps to reduce the dimensionality of the model, which in turn makes it lighter and faster. In addition, it can also reduce the amount of overfitting that occurs. The process of transfer learning is made easier by pruning, which enables the model to concentrate on features that have a high degree of relevance. This has the potential to result in improved generalization performance as well as a further increase in accuracy.

The process of pruning a neural network can be challenging for a number of different reasons. First, because the behaviour of DNNs is complex and nonlinear, it can be difficult to define exactly which weights and neurons are contributing to a neural network's performance. In addition, the act of pruning a neural network may result in the elimination of useful patterns and a reduction in the accuracy of the network. Pruning can also involve a large amount of computational effort, and if it is not performed correctly, it can result in an unstable model. Last but not least, the process might be further complicated by the fact that changing hyperparameters can alter the output of a pruned network. This means that the exact same pruned network may have somewhat different behaviour when it is trained with different hyperparameters values.

The fundamental reasons why we prune networks are covered in Section 3.1, and the components of the algorithm that does the pruning are analyzed in Section 3.2 in greater depth. Section 3.3 explains implementation details. Section 3.4 provides an illustration of the efficiency of pruning in classification and instance segmentation models; Section 3.5 is a conclusion on the pruning outcomes.

3.1 Pruning Objectives

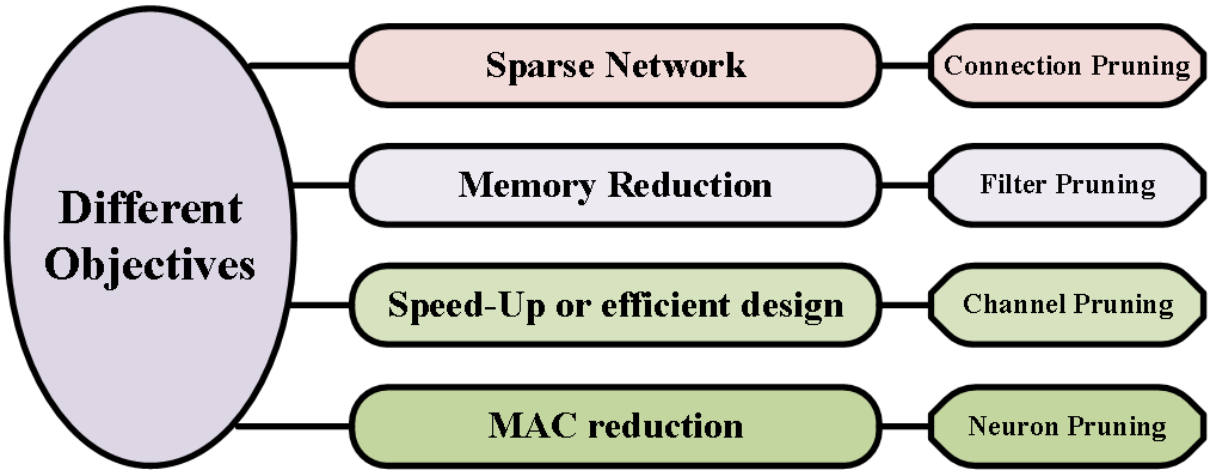

Figure 3.1: Pruning Objectives [61]

A CNN model requires a training stage in order to determine which of its variables and parameters are unnecessary or redundant. The training process can go on for a certain number of epochs or until the model reaches its optimal level of accuracy.

Once a CNN model has been trained, one may want to compress it in order to make it more computationally efficient and enable faster deployment. To accomplish the appropriate pruning and compression, it is important to understand the objectives of the compression. Different objectives have been investigated in previous studies, as shown in Figure 3.1. In [44], pruning is used to reduce the number of parameters. As a result the number of operations namely Multiply and Accumulate (MAC) is decreased; consequently, the goal may be to decrease the number of MAC operations. Another way to bring down the total number of MAC operations is to cut down on the total number of neurons.

In [52], pruning is employed to design a low-memory architecture compliant with a particular hardware implementation, achieved via filter pruning. In [12], pruning is employed as a way to cut down memory use by considering feature maps as the most significant energy-consuming factor for CNNs. Beyond this, pruning can enhance architecture, cut down the model's critical path, and optimize run-time performance. Hu et al . employed multiple objectives to achieve the desired outcome [31]. In [44], Li et al. employed the pruning of only the convolutional layers with the goal of decreasing computational power He et al. in [27] focused on pruning dense layers to reduce the model's complexity. Also, in [52], the entire model was condensed to generate a slender model with fewer operations.

3.2 Pruning Methods

The aim of pruning is to achieve the required trade-off between cost, size, and performance without sacrificing accuracy. To achieve this, different pruning algorithms are used, such as the Lottery Ticket Hypothesis, Pruning Filters for Efficient ConvNet(PFEC), Group Fisher Pruning, Neural Pruning with Growing Regularization and Pruning Filter with Attenuation.

In this section, we analyze the various pruning algorithms which are used to make CNNs more compact and efficient. We will look at the technical aspects of each algorithm and investigate the different sections of a CNN that can be targeted for optimization and pruning. We will also explore the trade-offs between structured and unstructured pruning and determine the efficiency of each algorithm in achieving the desired trade-offs. Further, we will delve into how each pruning algorithm is implemented and its efficacy in achieving the desired results. Finally, we will evaluate each pruning method and devise a quantitative analysis of each method's efficacy.

3.2.1 Pruning Filters for Efficient ConvNet(PFEC) [44]

The L1 norm is an appropriate consideration for deciding on filters in the absence of data because it boosts sparsity. The Pruning Filters for Efficient ConvNet(PFEC) [44] proposed by Li et al. operates in several steps to effectively prune a given number of filters in the i th convolutional layer. The first step is to calculate the sum of absolute kernel weights for each filter in the convolutional layer and store this value in s_j. Next, the filters are sorted according to the sum of their absolute kernel weights. Subsequently, the filters that possess the minimum sum values are eliminated, along with their associated feature maps. Furthermore, the kernels associated with the trimmed feature maps in the subsequent convolutional layer are eliminated as well. Ultimately, a new kernel matrix is generated to accommodate the ith and i + 1th layers, while the residual kernel weights are transferred onto the freshly developed model.

Altering MRCNN's implementation of the Pruning for Efficient ConvNet algorithm allows for the elimination of not only filters but also neurons according to the same criteria (L1 norm). As a result, various networks can now adopt a layer and block-based philosophy, with individual pruning rates and ratios being set for each block or section. By doing so, the network's structure and requirements can determine the optimal number of neurons and filters to use.

3.2.2 Pruning Using Filter Attenuation(PUFA) [58]

Mousa-Pasandi et al. employed Pruning Using Filter Attenuation(PUFA) [58] to improve the efficiency and accuracy of a deep neural network model. It starts with a training data collection, a model architecture, and the allowed accuracy drop T1 and pruning rate T2, as well as the attenuation parameter k. The method produces a trimmed model architecture. To reach this result, the method first trains the model for a few epochs before looping through computations and fine-tuning until the ultimately required accuracy and model complexity are obtained.

The procedure then calculates the L1 norm for each filter inside the loop by first increasing k by some constant a. The model is then fine-tuned for a few epochs after attenuating the k least important filters. Any filters having an L1 norm smaller than T2 are trimmed after recalculating the L1 norm. The accuracy loss is then assessed, and if it is smaller than T1, the process of fine-tuning is repeated. When the accuracy loss exceeds T1, the last pruned filters are retrieved, a new round of fine-tuning is performed, and the model complexity is calculated. The model is stored if the model complexity is appropriate. Otherwise, the loop is restarted by incrementing k.

Furthermore, in the context of the segmentation task discussed in section 3.4.2, our approach involved the utilization of neuron pruning techniques, specifically, the L1 norm filter and attenuation-based neuron pruning. This included increasing k and calculating the importance scores for each neuron. Those with scores below a certain level were attenuated, pruned the lowest ones, and the model was fine-tuned. The accuracy reduction was tested again, and if it was more than our permitted value, the neurons were restored and the model was saved. The model was fine-tuned after trimming to enhance accuracy even more.

3.2.2.1 Problem Formulation of Filter Pruning

Given a CNN composed of multiple layers, this problem statement seeks to explore the mathematical foundations of the filter pruning, which is used to accelerate the inference time of the CNN. Specifically, the research seeks to define the algorithms and methods used within filter pruning and provide mathematical equations that demonstrate the algorithm's evaluation of convolutional layers and reduction of network parameters.

Let W_{ij} denote each component of the weights matrix of size $m \times n$ and λ denote the regularization parameter for L1 norm-based pruning and μ denote the attenuation factor for weak components (e.g. smaller than some threshold).

Filter pruning that selects with L1 norm:

$$W_{ij} = \begin{cases} W_{ij} & \text{if } |W_{ij}| > \lambda \\ 0 & otherwise \end{cases} \tag{3.1}$$

Subsequent to attenuation of weak components:

$$W'_{ij} = \begin{cases} W_{ij} & \text{if } |W_{ij}| > \mu \\ \alpha W_{ij} & otherwise \end{cases} \tag{3.2}$$

Where α is the attenuation factor. Pruning lowest gradually:

$$W''_{ij} = \begin{cases} W'_{ij} & \text{if } |W'_{ij}| > \eta \\ 0 & otherwise \end{cases} \tag{3.3}$$

Where η is the threshold used for pruning. Let W_{ij} denote each component of the weights matrix of size $m \times n$. Then, for a given value of the regularization parameter λ, the matrix W can be pruned by first selecting with L1 norm such that components with values lower than λ are set to 0. Subsequently, components with values lower than μ are attenuated by a factor of α. Finally, the components with values lower than a given threshold η can be pruned gradually.

3.2.3 Neural Pruning via Growing Regularization(GREG) [79]

L2 regularization is a type of regularization used in convolutional neural networks (CNNs) to reduce the amount of overfitting in a model. L2 regularization applies a penalty to the weights of the neural network by adding a term to the loss function of the network. This penalizes large weights and helps to reduce overfitting.

L2 regularization modifies the loss function by adding a penalty which is the sum of the squares of the weights of the model. By increasing the L2 regularization to infinity, the model's weights are penalized too severely and pushed to close to zero, thereby preventing the model from being able to learn useful patterns from the data. Conversely, if the L2 regularization is set to close to zero, the model will be able to learn too much of the noise in the data, instead of the practical patterns, and would thereby be over-fitted.

Wang et al. employed Neural Pruning via Growing Regularization(GREG-2) which is an iterative pruning algorithm that was developed with the goal of effectively decreasing

the total number of network parameters. It accomplishes this goal by incorporating the principle of regularization into its method of pruning. The algorithm is able to perform more effective pruning and produce better model results as a result of the gradual increase in the level of regularization that it applies at each pruning step. To be more specific, the algorithm begins by setting a regularization ceiling (i.e. the maximum amount of regularization that can be applied to a layer) and a ceiling for picking. The regularization ceiling is the maximum amount of regularization that can be applied to a layer (i.e. the minimum amount of regularization that needs to be applied before a filter can be pruned). The process then iteratively updates the network weights by running a stochastic gradient descent, where the regularization is enforced, until certain criteria are met. This continues until a certain threshold is reached. During each iteration, the regularization of a subset of the filters is increased by an adjustable amount (the granularity), while the regularization of the remaining filters is set to a negative value. This is done so that the overall regularization of the filters can be improved (the original weight decay).

Finally, after a fixed number of iterations known as the interval, the set of filters that show a significant increase in regularization are eliminated while the remaining filters are retained in the cause. This process is known as pruning. Following the completion of the pruning process, the model will undergo fine-tuning in order to regain its accuracy. GREG-2 is able to selectively prune filters in an efficient manner, which ultimately results in better model performance. This ability is made possible by gradually increasing the levels of regularization that are used in this process. In addition, the algorithm can be further customized to provide more optimal pruning schedules by appropriately setting the interval and granularity values. This ensures even better performance overall.

3.2.4 Others

Girish et al. used Iterative Pruning for the Lottery Ticket Hypothesis (LTH) [19], a method for determining a neural network's best set of weights. It operates by randomly initializing the network's weights and specifying a pruning target percentage and number of pruning rounds. The network is then trained for N iterations using the current masked weights. Following that, a subset of the masked weights is set to 0, as determined by the pruning target percentage divided by the number of pruning rounds. The weights are reset to their initial values, and the process is repeated until all of the pruning rounds are finished. The optimal set of weights (winning tickets) should be found at the end of this process.

The Layer Grouping algorithm, which can detect coupled channels in residual network convolutional layers automatically, is used at the outset of Group Fisher Pruning (Fisher)

Algorithm 3.1 Pruning Schemes LTH [16] PFEC [44] Fisher [49] GREG [79] PUFA [58]

Input:
Training data
Pre-trained Model
Output:
Pruned model

1: Train model for a few epochs
2: Calculate the score (L1 norm for each Filter in Convs), (Abs for Weights in layers) or (group loss for each channel in Convs)
3: **while** model complexity is not acceptable **do**
4: **while** weights are not stabilized **do**
5: Increase the Regularization of selected filters, push weights to 0
6: **if** Regularization Ratio is acceptable **then**
7: stabilize by training 10000 iteration
8: **end if**
9: **end while**
10: Multiplying their weight with a learnable mask ▷ Attenuate the lowest filters
11: Prune lowest Filters, neurons, Weights or Channels
12: Fine-tuning in a few epochs
13: **end while**
14: Fine-tuning in a few epochs
15: **return** Pruned Model

only GREG We loop here until reaching Regularization ceiling.

[49]. By applying Depth-First Search (DFS) to the graph G and looking for any parent nodes in the desired groups, it is able to take in layers L and return groups of layers G containing convolutional/depthwise convolutional layers. [49]

A sample-wise gradient method is incorporated into Fisher [49] to rank the relative importance of individual or coupled channels. It employs Fisher Information, which measures the variation of a function's output associated with changes in its input, to determine the importance of each channel based on the gradient computation. The Layer Grouping algorithm is used to find all of the layers in the model that have coupled channels. It then iterates through forward passes to measure the loss and uses backpropagation to compute the gradients of the parameters. It also accumulates memory-normalized importance scores during this step by applying Taylor expansion to the network loss. The model parameters are then updated using gradient descent while the iteration index is incremented. Finally,

the algorithm prunes the least important channel at each prune interval (specified by the user) by setting the corresponding mask to 0 and resetting the importance scores. As a result, the algorithm is able to effectively prune the least important channels until the desired FLOPS are achieved [49].

The algorithm 3.1 illustrates a pruning plan is used different approaches. It trains a model on a given training dataset, and calculates scores, model loss and regularization ratios while ensuring the complexity remains acceptable by pruning the lowest filters, neurons, weights and channels. If regularization is acceptable, it then stabilizes the weights by training for additional iterations and attenuating the lowest filters on the GREG method and other methods don't. Finally, the model is fine-tuned in a few epochs and a pruned model is returned.

3.3 Implementation Details

Before and after using pruning methods in classification and instance segmentation tasks, significant consideration needs to be given to the implementation details and training methodologies for these approaches. Before and after executing the various pruning procedures, the resulting models need to be fine-tuned in order to achieve optimal performance. The various pruning approaches can be applied to a wide variety of backbone topologies. After that, a set of tests can be utilized to determine the most effective training procedures, which can subsequently be used to optimize accuracy and decrease pruning difficulties.

In this part, we give the implementation details based on utilizing various pruning methods in classification and instance segmentation performed on a variety of backbone topologies. These findings are presented based on the results of the classification. In addition, further experiments are carried out in order to examine these algorithms and evaluate the usefulness and efficiency of each pruning approach with regard to instance segmentation and object detection for a variety of architectural configurations.

3.3.1 Architectures and Datasets

We trained distinct CNNs for each dataset, applied pruning methods, and compared the resulting accuracy to that of a baseline model in order to evaluate the efficacy of the models' pruning approaches using the Cifar10 and MSCOCO2017 datasets. We employed many measures, including FLOPS and parameter reduction, to evaluate the performance of the pruning algorithms and the accuracy of the resulting models.

The Canadian Institute for Advanced Research(CIFAR) developed the Cifar10 dataset, which is a collection of labelled images. It comprises of 50,000 colour pictures with a resolution of 32 by 32 pixels, organized into 10 classes. These 50,000 images are divided into training and test sets of 10,000 each. The dataset contains ten different classes: aircraft, vehicle, bird, cat, deer, dog, frog, ship, and truck. PNG is the image format, and each picture has a resolution of 32 by 32 colour pixels.

For training in the classification domain, we employed SGD with a batch size of 128. Starting with a learning rate of 0.1 for the first 100 epochs, we decreased it to 0.01 for the next 50 epochs and finally to 0.001 for the final 50 epochs. The rate of weight loss was set to 0.0005.

Though Resnet-101's training time was longer than that of Resnet-56, it was able to reach its higher validation accuracy more quickly. We believe the distinction is because the Resnet-101 model has more parameters and layers.

An image benchmark known as MSCOCO2017 was made available by the Microsoft Cognitive Toolkit (CNTK) team. It consists of several tasks, such as object detection and instance segmentation, relating to the recognition and captioning of images. The training dataset has approximately 118,000 photos spanning 80 different item categories; the validation dataset and the test dataset, on the other hand, both comprise 5,000 images. The MSCOCO2017 is well-known for its comprehensive dataset as well as its rigorous assessment processes.

The objective of the instance segmentation task that is part of MSCOCO2017 is to segment individual objects that can be found in images. In light of this, it should be clear that, in contrast to object detection, which involves the identification of groups of objects, instance segmentation additionally necessitates the accurate masking and outlining of the contours of individual objects. Several approaches, such as pixel-wise class models and recognize-and-segment models, which involve recognition followed by segmentation, can be utilized in order to carry out this task. In addition, the MSCOCO2017 benchmark provides a task known as Object Detection and Panoptic Segmentation, the purpose of which is to jointly recognize and segment individual image components.

The accuracy of an object detection system can be measured in two different ways: by its average precision and its average recall. The proportion of accurate detections is referred to as average precision, while average recall evaluates the degree to which all of the important objects were identified. The Intersection Over Union (IOU) is a measure that is used to quantify the overlap that exists between two or more bounding boxes. It is also a useful metric for comparing the accuracy of one detection system to another; the higher the IOU, the better the accuracy of the prediction. There is a positive correlation between

average precision and average recall because both of these metrics measure the accuracy of predictions. Additionally, a high IOU corresponds to both a high average precision and a high average recall.

The Resnet50-FPN-MRCNN model was chosen as the starting point for the Instance Segmentation task in the MSCOCO2017. Resnet50, which was trained using the ImageNet dataset, is the foundation of this model. The training process for the model lasted for a total of 12 epochs, and the SGD optimization algorithm was used throughout, with the batch size set to 2. The learning rate was initially set to 0.02, but it was lowered to 0.002 after eight iterations of the learning process. The model additionally utilized a weight decay of 0.0001 and a momentum of 0.9. As part of the training procedure, there were 500 linear warmup iterations performed at a ratio of 0.001.

The Resnet50-FPN-Yolact model is the second one that is utilized in this training procedure. This model also makes use of a Resnet50. It was trained for 55 iterations using SGD, and the batch size was 8. The starting learning rate was 0.001. At epochs 20, 42, 49, and 52, there was a reduction in the rate of learning. In addition, the model included 500 linear warmup iterations with a ratio of 0.1 and used a weight decay of 0.0005 for it. The momentum was set at 0.9.

The ConvNeXt-tiny-FPN-MRCNN model is the third one that is utilized in this training procedure. ConvNeXt-tiny serves as the foundation for this model. It was trained for a total of 36 epochs making use of the AdamW optimization algorithm with a batch size of 2. The learning rate was initially set to 0.0002, and after 27 epochs, it was reduced to 0.00002 in order to achieve the desired result. In addition, the model went through a total of a thousand linear warmup iterations and used a weight decay of 0.05.

3.3.2 Fine-Tuning

When performing fine-tuning on the ResNet models following pruning, it is necessary to consider the training parameters carefully. Based on standard practice for fine-tuning pruned classification models in [38] and [79], we were able to accomplish what we set out to do by employing the SGD optimizer, with a batch size of 128, and a weight decay of 0.0005 per iteration. The learning rate was a dynamic parameter that was initially set at 0.01 until epoch 60, then decreased to 0.001 until epoch 90, and then decreased to 0.0001 until the end of the training process. In conclusion, a momentum of 0.9 was utilized throughout the process of training in order to guarantee more accurate revisions of the model parameters.

Based on fine-tuning practice that is employed in [49], a 12-epoch cycle was utilized during our process of fine-tuning the Resnet50-FPN-MRCNN. The initial learning rate was set to 0.002, and then it was decreased to 0.0002 after eight epochs had passed. The rate of weight decay that we used was 0.0001, and the momentum that we used was 0.9. We began our training by performing 500 linear warmup iterations at a ratio of 0.001 in order to ensure a seamless transition. A batch size of two was used for the training of the model.

During this step of the fine-tuning process for ResNet50-FPN-Yolact, the SGD optimizer was used, and the batch size was set to 2, and the learning rate was set to 0.001. After eight iterations of training, this learning rate was brought down to 0.0001 percent. In addition to the learning rate, a weight decay of 0.0005 and a momentum of 0.9 were utilized in this analysis. In order to provide additional assistance with the process of fine-tuning, 500 linear warmup iterations with a ratio of 0.1 were carried out. This process of fine-tuning took place over the course of a total of twelve epochs.

The initial learning rate for the ConvNeXt-tiny-FPN-MRCNN was 0.0002, and the fine-tuning procedure consisted of 12 epochs with a batch size of 2. Following the eighth epoch, the rate of learning was slowed down to 0.00002 per second. A linear warmup of 1000 iterations was performed with a ratio of 0.1, which assists the network in adjusting to the training parameters during the subsequent training phase. The AdamW optimization, which is a momentum-based algorithm, is used throughout the process. This algorithm's goals are to improve accuracy while simultaneously reducing instances of overfitting. After each epoch, the performance of the model is assessed, and the training procedure is repeated as many times as necessary until it reaches an acceptable level.

3.4 Experimental Results

In this part, we give the outcomes of utilizing various pruning methods in the classification based on a variety of backbone topologies. These findings are presented based on the results of the classification. In addition, further experiments are carried out in order to examine these algorithms and evaluate the usefulness and efficiency of each pruning approach with regard to Instance Segmentation and Object Detection for a variety of architectural configurations.

The classification accuracy was determined using 10 classes of cifar 10, and the results on instance segmentation and object identification were determined using the mean average precision and mean average recall from the mask pixel or the bounding box. The failure findings and the related visuals on instance segmentation are bringing on additional

discussions. When compared with the baseline model and the other algorithms, our failure outcomes were comparable. It is important to note that the display of failure pictures is based on demonstrating how handicapped the network gets when a specific compression strategy has been applied.

The primary issues that need answers are how each pruning strategy compares to other compression methods and how efficient it is in the sequence. How the design of an effective pruning technique might have an effect on the final output mask of the picture, as well as how the fine-tuning could potentially resound in large parameter decreases. Lastly, we want to compare the strength and weaknesses of each method in real-world scenarios and applications.

In the first subsection, we discuss the results obtained after pruning classification models. In the second subsection, we discuss instance segmentation models of compression. And in the third and final subsection, we provide an overview of the failure results and potential takeaways regarding the design of suitable frameworks.

Table 3.1: Accuracy comparison of different pruning methods on Cifar10 with ResNet56 & ResNet101

Model Arch	Pruning Algorithm	Accuracy (%)	Parameter Reduction (%)	FLOPS Reduction (%)
R56	Baseline	93.36	0	0
	PFEC [44]	91.96	46.65	50
	GREG2 [79]	**92.65**	46.65	50
	PUFA [58]	92.40	46.65	50
	PUFA + GREG2	92.55	46.65	50
R101	Baseline	93.99	0	0
	PFEC [44]	93.59	42.6	45.7
	GREG2 [79]	**93.9**	42.6	45.7
	PUFA + GREG2	93.63	42.6	45.7

3.4.1 Pruning Results in Classification

Fine-tuning is a common practice in pruning research, employing conventional training algorithms in epochs. In the ResNet56 model, we initially finetuned the model using SGD after training 200 epochs; after the pruning section, we had 120 epochs for fine-tuning with the same optimizer [79]. As observed in table 3.1, the filter pruning algorithms, The GREG appeared superior, gains by 0.69 percent higher than fixed regularizer and one shot pruner and 0.25 percent higher than PUFA pruner with fixed regularizer.

Using Nvidia GeForce RTX 2080 Ti, the ResNet56 has an accuracy of 93.36 percent based on the training method that was provided to it, and it can train at a pace of 50 frames per second while taking an hour to do so. The 27 blocks are learned by ResNet101 in under two hours, and it achieves an accuracy of 93.88 percent while maintaining a frame rate of 40.

The L1 Oneshot pruning method is harmful to potentially relevant filters as a result of ignoring the significance of gradients as a factor in the pruning process. The real reason for the fluctuations is that training ResNet56 on a constant gradient makes the GREG [79] information in the network easier to decode, eliminating the need for attuning the network. Nevertheless, there is a point where removing the Oneshot could damage the complete accuracy; consequently, making use of the attenuation and gradually pruning would improve the foundational model.

A 0.15 percentage point improvement in ResNet56 and 0.27 in ResNet101 may be attributed to the utilization of Attenuation in conjunction with Growing regularization compared to using only Attenuation. In order to carry out a comparison that was objective, we placed all of the pruning ratios in the layers that had the same number of overall; the results in greater pruning percent would fluctuate more. We utilized ResNet56; in addition, the pruning ratio is roughly 75, 75, and 32 percent in each step of the model; and the speed increase is approximately 2.55 times the speed of the basic model.

3.4.2 Pruning Results in Instance Segmentation

In the ResNet50 model, initially, we trained the model using SGD for 24 epochs then, after the pruning section, we had 12 epochs for fine-tuning using SGD [6]. As illustrated in figure 3.3, L1 norm filter and network pruning in MRCNN ResNet50 after 50 percent pruning surpasses L2 norm pruning by 2 percent. This is due to the fact that the L1 norm filter is able to capture outliers and pick weights with more importance than others. The network pruning process is able to cut down on the amount of duplication that exists

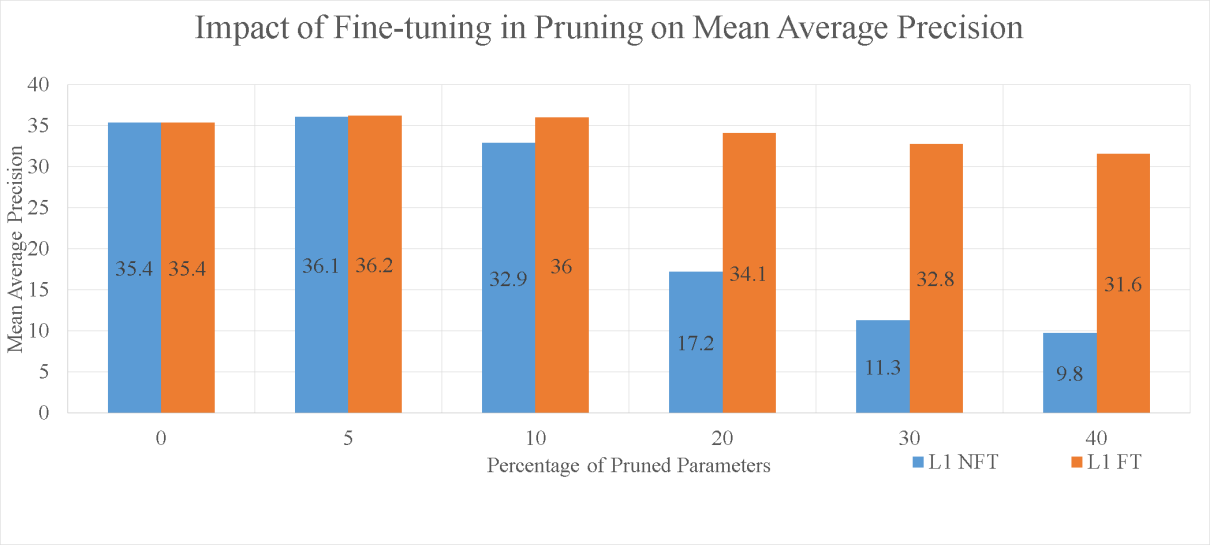

Figure 3.2: Fine-Tuning vs No Fine-Tuning for filter and neuron pruning Resnet50-FPN-MRCNN using L1 norm one-shot

inside the network and as a result, reduce the total number of parameters, all without causing a major drop in the network's overall performance. This contributes to the general simplification of the design, which ultimately results in a model that is both quicker and more effective. In addition, in comparison to the L2 norm approach, the L1 norm method is more computationally efficient, which contributes to the improvement in performance that this method achieves.

As shown in figure 3.2, After applying L1 norm filter and network pruning to MRCNN ResNet50, the performance increases by 20 percent when fine-tuning is applied. This is due to the reduced computational complexity of the network after pruning which allows for more efficient use of the network's resources and thus results in improved accuracy. Furthermore, the fine-tuning helps the network adjust to the changes brought about by the pruning, further optimizing the model's performance. The combination of L1 norm with attenuation filter and network pruning in ResNet50-FPN-MRCNN , ResNet50-FPN-Yolact and ConvNeXt-tiny MRCNN results in improved accuracy than just applying L1 norm alone. This can be explained in two ways. Firstly, attenuation filters reduce the number of weights in a CNN by decreasing the weight of certain parameters. This allows for larger proportion of more important parameters in the model, and also avoids over-fitting. Secondly, network pruning reduces the number of neurons in a CNN, which again allows for better overall performance as the model can focus on the important parameters while

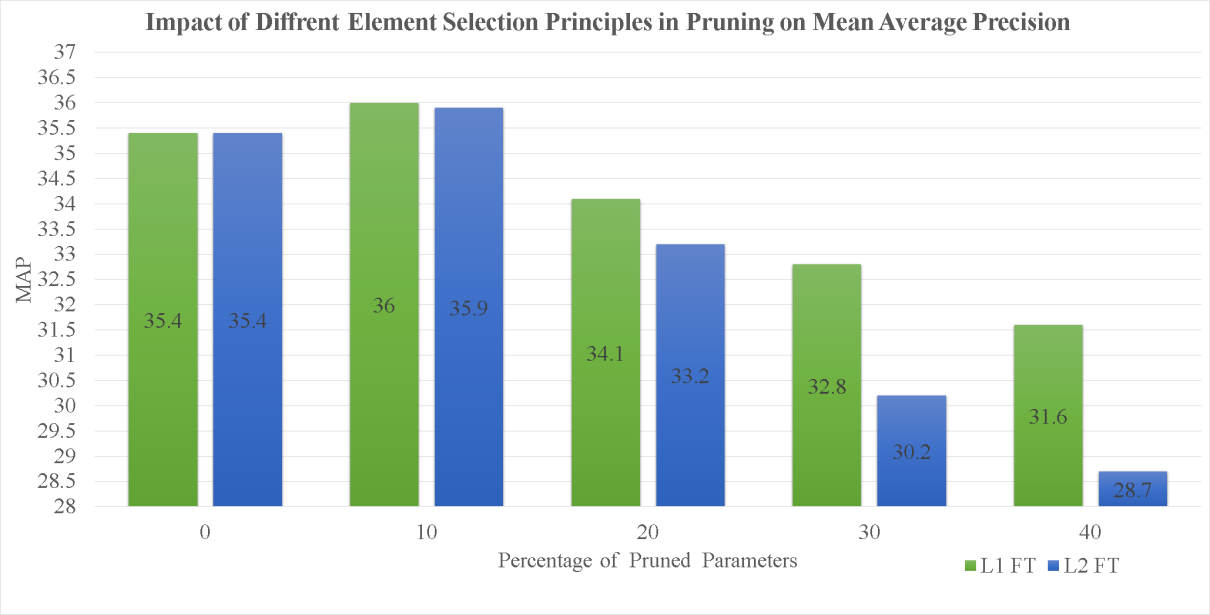

Figure 3.3: L1 vs L2 norm for filter and neuron pruning Resnet50-FPN-MRCNN

ignoring the less important ones. This combination of L1 norm with attenuation filter and network pruning can therefore improve accuracy by up to 2 percent, thus outperforming the simple L1 norm alone. Figure 3.4 shows that the L1 norm pruning with attenuation method outperforms the L1 norm one-shot pruning method when applied to the testing image in the instance segmentation task. This indicates that the quality of masks and boxes on the pruning results remains high even after a 50 percent reduction in parameters. Furthermore, L1 norm pruning with attenuation has comparable masks and boxes to the baseline while still capturing minor details.

Lottery Ticket Hypothesis (LTH) weight pruning is able to achieve better accuracy than L1 norm filter pruning after a reduction of 50 percent of the network's parameters. This is because LTH weight pruning is able to prune network parameters with greater precision and speed by taking into account the significance of each parameter with statistical certainty. LTH weight pruning is a slower by 4 Frames Per Second (FPS) and more complicated technique for weight pruning when compared to the L1 norm filter pruning, which is much simpler and more efficient. However, LTH weight pruning does increase accuracy.

The Fisher channel pruning technique is an efficient method for lowering the number of parameters used in the ResNet50-FPN-MRCNN model. In order for it to function properly, unnecessary layers and channels that do not contribute to accuracy are eliminated.

The model is able to make do with less memory and processing resources because to the elimination of these layers and channels, and it is still capable of producing accurate results. Fisher channel pruning results in a reduction of 50 percent of the parameters, which in turn leads to an improvement of 7 percent in accuracy and a speedup of 0.5 frames per second.

Table 3.2: Average Precision (AP%) comparison of different pruning methods on MSCOCO2017 dataset, pruning methods: L1 refers to L1 norm Filter and neuron pruning, L1+B refers to L1 norm block and neuron, Device: Nvidia RTX 2080 Ti

Model Design	Pruning Algorithm	AP^{Mask} (%)	AP^{Box} (%)	Parameters(M) PRP(↓ %)	FLOPS(G) ORP(↓ %)	Speed (FPS)
R50 FPN MRCNN	Baseline	34.7	38.2	44(0)	169(0)	14.5
	L1	24.43	29.67	22(50)	80(52)	18.5
	LTH [19]	**35.3**	**38.4**	44(sparsity:50)	169(0)	14.6
	Fisher [49]	28.3	31.2	22(50)	90.2(46.7)	15
	L1+Att	25.6	25.6	22(50)	80(52)	18.7
	L1+B	16.6	21.6	22(50)	**78(53)**	**20.1**
R50 FPN YOLACT	Baseline	28.2	31.2	35.3	64.11	24.2
	L1	25.7	27.2	26.4(25)	50(25)	25.1
	L1+Att	**26.1**	**27.9**	26.4(25)	50(25)	25.1
	L1	15.5	16.1	17.6(50)	29.55(46.1)	27.6
	L1+Att	**17.2**	**19.6**	17.6(50)	29.55(46.1)	**27.6**
ConvNeXt-T FPN MRCNN	Baseline	41.7	46.2	48.1	271.85	19.8
	L1	35.3	37.1	24.05(50)	128(47.2)	20.3
	L1+Att	**36.4**	**39.5**	24.05(50)	128(47.2)	**20.3**

3.4.3 Discussion

In this discussion, we will compare and contrast the benefits and drawbacks of various pruning techniques in terms of speed, real-world implementation, as well as algorithm time and memory complexity. It is feasible to determine the most appropriate method for a provided application and ensure that the benefits of pruning are realized while minimizing any negative impacts that may have on model performance if one understands the strengths and weaknesses of each technique. The technique of L1 norm block and neuron pruning is

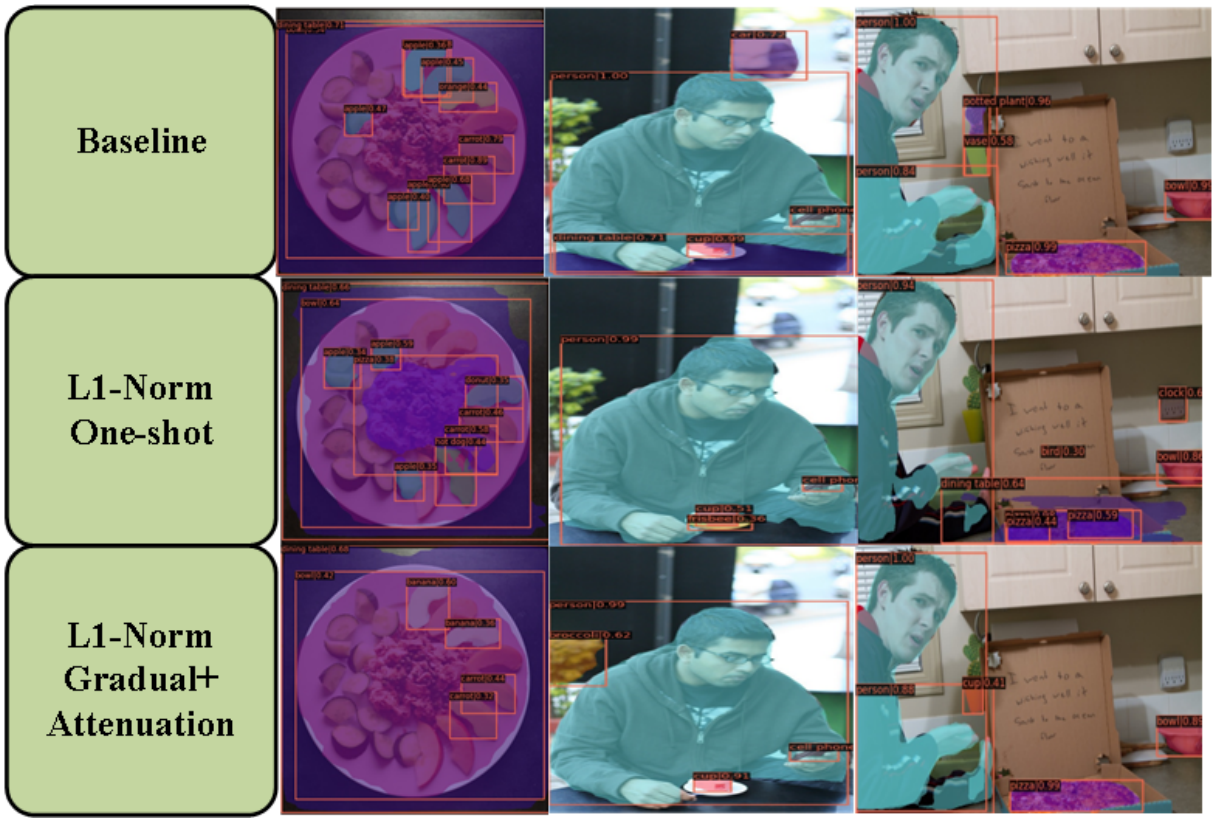

Figure 3.4: ResNet50-FPN-MRCNN testing results after pruning

a method of pruning that exhibits the highest speed, resulting in a 5.6 frame improvement and a 53% reduction in floating point operations. Nevertheless, it is not recommended due to its substantial 18.1% decrease in Average Precision, rendering it impractical for the majority of real-world scenarios. Despite its potential for increased speed, this method should be abandoned unless prioritizing precision is not of paramount importance.

L1 norm one-shot Filter and neuron pruning involve removing a large number of filters or neurons from a model all at once, using the L1 norm as the criterion for selecting which filters or neurons to keep and which to remove. One of the main benefits of this method is its simplicity. It is relatively easy to implement and requires only a single pruning step, making it relatively quick compared to other pruning methods. This method is straightforward in terms of real-world implementation and can be easily integrated into a machine-learning pipeline. The act of pruning leads to a decrease of 52% in floating point operations, thereby potentially enhancing the model's speed by 4 frames. Nevertheless, the approach exhibits

a drawback in that it has the potential to cause disturbance to the model's performance, leading to a decrease of 10% in the mean precision. Hence, it is advised that some flexibility for implementation in practical scenarios where speed is a crucial factor and a marginal compromise in precision can be accommodated. In general, using the L1 norm one-shot filter and neuron pruning technique can be considered a valuable approach for diminishing the dimensions of deep neural networks. However, it is advisable to exercise caution when implementing this method to prevent any adverse effects on the model's efficacy.

The L1 norm filter and neuron pruning with attenuation approach involves a gradual elimination of filters or neurons from a model based on the L1 norm as the selection criterion. Additionally, the remaining filters or neurons are attenuated to account for the removed ones. The aforementioned approach exhibits the second highest speed, with a notable enhancement of four frames, and a reduction of 52 percent in floating point operations. However, it ranks third from the bottom in terms of Average Precision, with a reduction of 9 percent. This approach is suggested for practical applications where speed is essential, precision is of secondary importance, and a less intrusive pruning technique is favoured over L1 norm one-time pruning. The principal drawback of this approach lies in its greater resource demands in contrast to the l1 norm one-shot method. Nonetheless, this approach remains uncomplicated to execute and can be seamlessly incorporated into the machine learning pipeline.

L2 norm Filter and neuron pruning is very similar to L1 norm pruning; however, rather than using the L1 norm as the selection criterion for which filters or neurons should be pruned, filter and neuron pruning uses the L2 norm. When compared to L1 norm pruning, it, unfortunately, did not perform better. Despite its potential benefits, the approach ultimately fell short of expectations in terms of maintaining model performance. This method is not overly complicated to implement in the real world, and it is simple enough to be incorporated into a machine-learning pipeline without much difficulty. Compared to selecting filters or neurons to prune using the L1 norm, the process of using the L2 norm can be more computationally intensive.

The lottery ticket hypothesis (LTH) weight and connection pruning is a pruning method used to reduce the number of parameters in trained neural networks. In terms of metrics, LTH connection pruning has the advantage of achieving the best Average Precision with a 0.6 percent gain compared to the baseline model. However, there is no improvement in terms of speed, as it is the same as the baseline model with 0 Frame improvement. This method is recommended in real-world applications if accuracy is crucial and speed is not a major concern. LTH pruning is computationally efficient, allowing for a reduction in algorithm time and memory requirements, making it suitable for deployment in resource-constrained environments. However, LTH pruning is data-dependent, making it

challenging to apply to other datasets, and time-consuming to apply in practical settings, as it necessitates training and pruning for each new dataset. LTH pruning also uses a greedy algorithm, which might not be able to locate the overall ideal solution, and there may be no speedup for large datasets and models.

The Fisher channel and neuron pruning technique is a channel pruning methodology that involves the categorization of filters or neurons into channels, predicated on their Fisher information score. Empirical evidence suggests that the implementation of this methodology yields a decrease in the dimensions of the model, while simultaneously preserving its efficacy. This technique demonstrated notable results in terms of metrics, achieving the fourth position in speed enhancement, with a 0.6 frame improvement and a 53% reduction in floating point operations. Furthermore, it attained the second highest outcome in the Average Precision metric, exhibiting a decrease of 6.4%. This methodology is suggested for practical implementations in which precision and speed hold significance. It is noteworthy to mention that the implementation of this technique may present difficulties and demand a considerable duration of time owing to its algorithmic complexity and memory consumption during the computation of the backward gradient in the pruning process.

Regarding the questions presented in section 1.4, it has been observed that in instance segmentation, the implementation of connection pruning can result in enhanced precision. Conversely, filter and neuron pruning techniques have been linked to faster processing, whereas channel and neuron pruning methods offer an optimal balance between structural pruning and efficacy. The response to the second inquiry is as follows. Research has shown that filter pruning algorithms utilizing the L1 norm exhibit superior efficacy compared to those utilizing the L2 norm. Fine-tuning has been observed to enhance the accuracy of a pruned model by up to 20 points while simultaneously reducing the parameters by 40% in filter pruning. The combination of these techniques holds the potential to enhance the efficacy and precision of instance segmentation models, thereby rendering them more suitable for practical applications.

3.5 Summary of Pruning

Pruning is a common method that is utilized to cut down on the overall size of deep learning models. It is utilized to reduce the size of well-trained models without significantly compromising their overall performance in the process. This is accomplished by pruning algorithms, which strip the model of any weights or neurons that are unnecessary, redundant, or otherwise irrelevant. The majority of the time, this method is put to use for

activities such as object detection or classification. In this chapter, we investigate three models that are considered to be state-of-the-art in the field of instance segmentation and analyze the performance of these models both before and after they have been pruned.

The results of a classification analysis indicate that the GREG method is superior to the others. Under the same speed-up situation, it achieved a 0.69 percent accuracy improvement over the fixed regularizers and one-shot pruner and a 0.25 percent higher rate of accuracy than the PUFA fixed regularizer. This study takes a complete look at the results of pruning methods on the ResNet56 and ResNet101 models. Additionally, it offers valuable insights into the impact of pruning techniques on both the network's performance and computational efficiency.

The ConvNeXt Tiny MRCNN, the Resnet50 MRCNN, and the Resnet50 Yolact are the three models. On the COCO dataset, the models were analyzed for their performance. The precision of LTH unstructured connection pruning surpasses that of structured L1 norm filter and neuron pruning due to its ability to selectively remove individual connections from neural networks based on their weights, enabling more precise pruning decisions. Conversely, the implementation of Fisher channel pruning with high precision revealed that reducing the number of channels did not yield the expected acceleration.

It was also discovered that using the L1 norm filter and neuron pruning with attenuation produced better results for all three models than simply employing the L1 norm filter pruning, the L2 filter pruning, and the L1 block Pruning. Despite the significant reduction in model size resulting from the pruning, the performance of the L1 norm filter and neuron pruning with the attenuation method remained satisfactory. This demonstrates how important it is to research various pruning algorithms and the effectiveness of those algorithms for a variety of tasks.

In general, the findings for the three models demonstrated that pruning is a workable alternative for decreasing the size of the model without significantly impacting its performance. However, L1 norm filter pruning with attenuation is unlikely to be sufficient on its own to build a model that is faster than its counterpart. The results demonstrate that it remains essential it is to pursue the investigation of a variety of pruning algorithms and the results they produce.

Chapter 4

Sensor Fusion for Detection

The detection of moving objects in two-dimensional scenes is a challenging problem that is typically solved by combining the information from multiple sensors. The term "sensor fusion" refers to a collection of algorithms and processes that are used to combine or fuse the data captured by multiple sensor into a single output or representation. This data may have originated from a camera, radar, sonar, or lidar, as well as any other type of sensors. The objective is to obtain a more robust and more precise comprehension of the situation is attainable through the consolidation of the information provided by numerous sensors.

Sensor fusion provides adaptability and scalability, it consists an excellent framework for the planning, development, and implementation of a wide variety of object detection tasks, including those used in robotics, autonomous vehicles, and surveillance systems. In general, any task that is related to the detection of moving objects can be made more accurate by using the appropriate combination of sensors and sensor fusion operators.

Due to the need for combining images, point clouds and other types of data to calculate information accurately, fusion is a necessary component of 3D object and scene detection research. Building upon existing 2D detection abilities, researchers need to bridge the gap between 2D and 3D to achieve accurate computed results. Fusion has its own challenges, as explored in Section 4.1, which focuses on key differences between early and mid-level fusion and their practical applications. Section 4.2 examines the challenges which occur due to differences between fusion operators and section 4.3 explains implementation details. Following this, Section 4.4 features experimental results achieved by the fusion mechanism. Finally, Section 4.5 concludes the results and shows the overall effectiveness of the fusion approach.

4.1 Architectural Setting for Fusion

To accomplish 2D road object detection, we choose a multi-sensor fusion architecture that accepts lidar and camera inputs. We conducted our tests on a reference network. We assessed the performance by swapping out different operators that fuses the two input modalities. We display the selected sensor fusion network's design in Figure 4.1. Our objective is to develop a model that attains a high level of precision while maintaining simplicity.

To identify 2D objects, the camera stream analyses a coloured input image. The drawbacks of utilising a camera alone include the effects of poor lighting, adverse weather, noise, reflections, and a lack of depth cues. An image-based object detector will perform poorly in these difficult circumstances. The purpose of the lidar stream is to deliver 3D data in the form of 3D point clouds to supplement the visual data. The chosen approach involved integrating the lidar data onto the camera view by superimposing depth, intensity, and height (Depth Height Intensity (DHI)) representations, thereby achieving a consistent representation of the modalities. The sensor fusion head of this early fusion network is

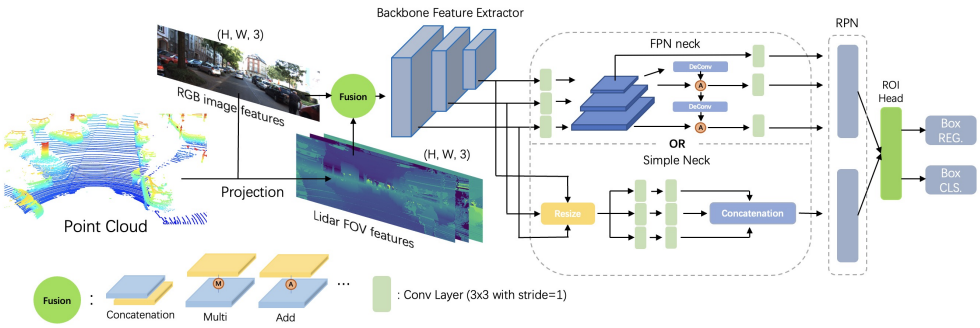

Figure 4.1: The structure of our sensor-fusion framework aims to comprehend the efficacy of various combination operators in a 2D detection task.

created by combining the camera and lidar frontal views using a fusion operator. The outcome of the fusion operation is then used to do inference in the 2D object detection. The Faster-RCNN [20] and Cascade-RCNN [4] detection frameworks, among others, served as inspiration for the design of the 2D detection network. For the purpose of producing object proposals, Faster-RCNN uses a region proposal network (RPN) and a region-of-interest (ROI) detection head. Each region proposal is used to generate a fixed-length feature vector, from which a classification score and predicted bounding box is calculated. The processed proposals from RPN and extracted features from the neck are inputs of the

ROI head. Cascade-RCNN is based on a sequential arrangement of three identical ROI heads, each aiding the next ROI head to have a better understanding of objects' boxes. To utilizing the proper neck for our structure Simple Neck and Feature Pyramid Network (FPN) neck [47] has been employed. FPN has 256 filters and Simple Neck has 96 filters in each stage, however, the Simple Neck output has 288 channels after concatenation of the output of different stages with the same shape. For the feature extraction backbone, ResNet30 [26] and Swin Transformer tiny [51] with three and four stages have been used. After initial convolutional layer with kernel size of 7 and 32 filters, The ResNet30 architecture comprises three stages, with each stage consisting of several basic blocks. In the first stage, there are 32 filters present in each layer, while the second stage has 64 filters in each layer. The last stage of the architecture contains 128 filters in each layer, and has a total of 6 basic blocks. On the other side four stages in Swin Transformer Tiny using window size of 7 and embedding size based on 96×2^i with i referring to stage index from zero to three. Most of the studies are using mid-level fusion that utilizes another network to process the data of lidar points and then fuse it to RGB-processed Feature Maps. We used the same backbone networks with Early and Mid-level fusion in that manner in which Mid-Level Fusion combine extracted features from each stage of the backbone. Figure 4.2 indicates the policy of applying this task.

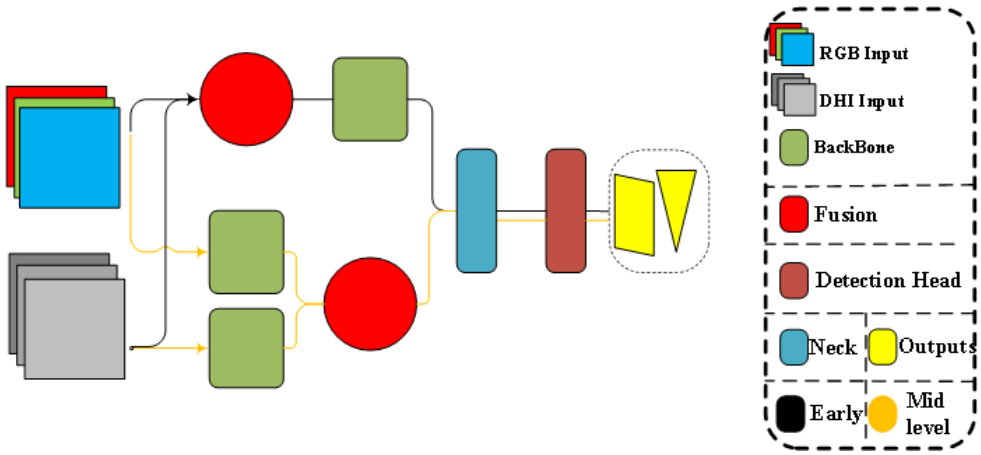

Figure 4.2: Black: Early fusion refers to the process of mixing inputs prior to their transmission to the base network during the initial phase. Orange: mid-level fusion involves the initial routing of inputs to two distinct base networks, followed by integrating the resulting features.

4.1.1 Structural Substitute in Feature Processing

FPN (Feature Pyramid Networks) contains a total of 256 filters in every stage, divided into 3-4 different sizes for different feature map shapes. It has a special lateral connection which connects the output of different stages with the same shape. On the other hand, Simple Neck has a simpler structure with only 96 filters in each stage and can have better performance compared to Vanilla network architectures [57]. The simple neck output consists of 288 channels after concatenation of the output of different stages with the same shape instead of 256 filters in 3-4 shapes in FPN. This is achieved by using a single feature map size in Simple Neck that matches the size of the smallest feature map size in FPN. This helps Simple Neck to achieve better performance than FPN. In addition, using only one feature map size in Simple Neck reduces the overall complexity of the architecture and makes it easier to implement.

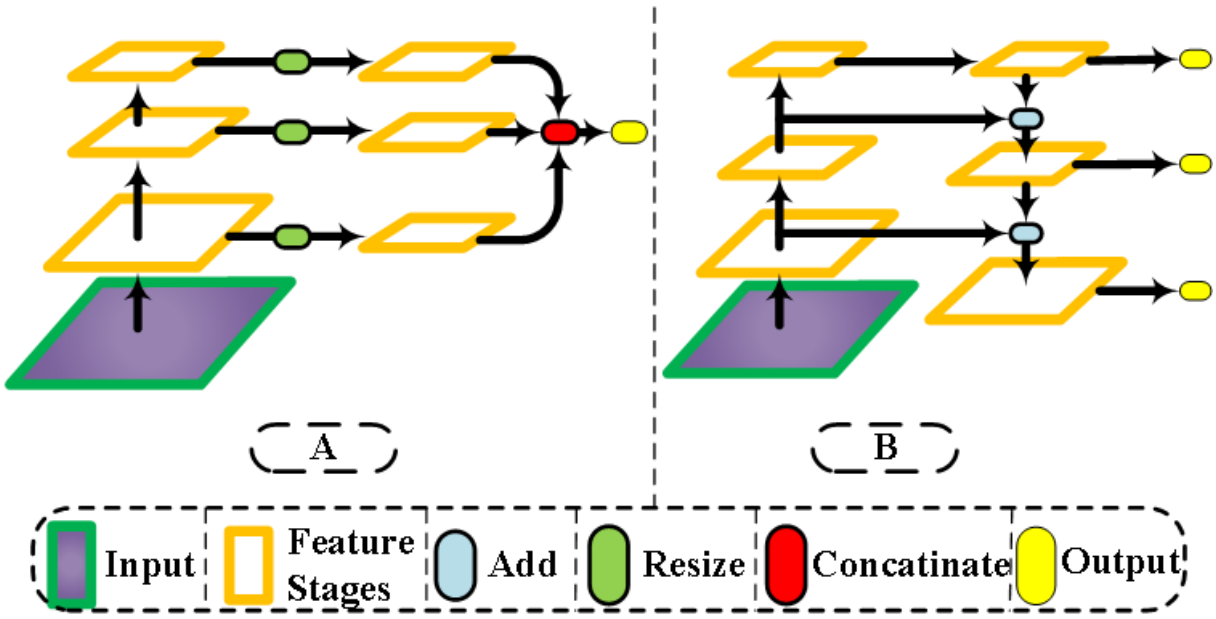

Figure 4.3: FPN vs. SN

4.1.2 Architectural Modification in Mid-Level Fusion

As illustrated in Figure 4.2, our implementation of mid-level fusion places an emphasis on utilizing two of the same backbones to detect features in an image and fusing them before those features reach the neck (a bottleneck in the network used as a bridge between earlier layers and the later layers). This method increases the input size of the neck in the concatenation approach of fusion, which refers to the process of combining two different input images into one single output image.

4.2 Fusions Operators

Sensor Fusions Operators are the primary tools used to combine and understand data from a variety of sensors. They allow for complex interactions between various digital information sources.

4.2.1 Simple Fusions Operators

Concatenation is a simple fusion operator that combines two or more input feature maps into one output feature map by appending one after the other. Element-wise summation is also a simple fusion operator that adds the values of corresponding elements, feature by feature, in two or more input feature maps into one output feature map. Multiplication is another simple fusion operator that multiplies the values of corresponding elements, feature by feature, in two or more input feature maps into one output feature map. XOR is a fixed fusion operator that creates an output feature map by comparing two input feature maps for any two corresponding elements and outputting a 0 or 1 depending on the result of the comparison. Mean and Max are both fixed fusion operators that create an output feature map composed of the average or maximum of all of the values of the corresponding entries in the input feature maps.

These simple and fixed fusion operators might not be robust enough to capture all of the details from feature maps and modalities, as they lack the complexity necessary to take into area multiple non-linear relations. Generally, a more sophisticated fusion algorithm would be needed to capture all the details of feature maps and modalities.

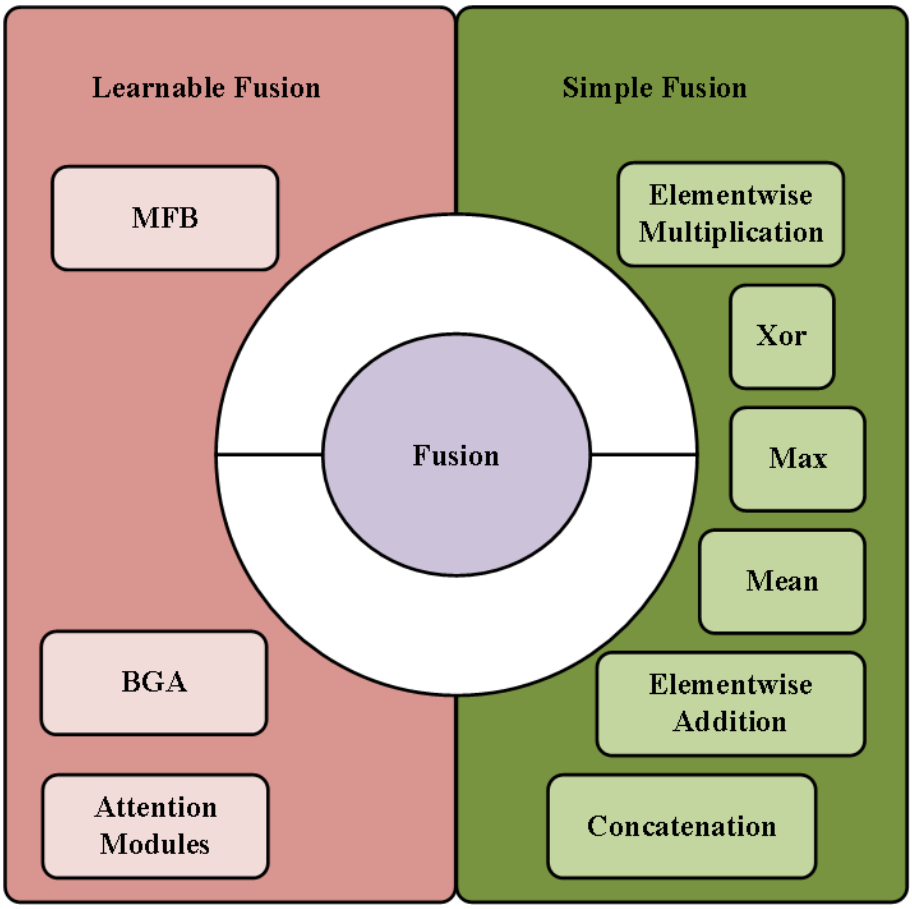

Figure 4.4: Simple vs Learnable Fusion Operators

4.2.2 Learnable Fusion Operators

Learnable fusion operators differ from simple fusion operators in that they are learned from the target data itself rather than relying on predetermined sets of operations. This allows sensor-learnable fusion operators to adapt to different problem scenarios and produce better classification results. Additionally, since the rules governing the fusion process are learned from the data, they often provide an additional level of accuracy and robustness.

Bilinear pooling is proposed by [86] as a powerful fusion mechanism but impractical from a computational perspective in image data. They then proposed a moresuitable mechanism based on factorized bilinear pooling (MFB) which is more robust than the

Steps	MFB	BG
Input	$\mathcal{F}_a = \text{Conv}_{3\rightarrow 6}(\mathcal{I}_{rgb})$ $\mathcal{F}_b = \text{Conv}_{3\rightarrow 6}(\mathcal{I}_{lidar})$ (4.1)	
Mid	$\mathcal{F}_{out1} = \mathcal{F}_{b1} \times \mathcal{F}_{a2} + \mathcal{F}_a$ $\mathcal{F}_{out2} = \mathcal{F}_{b2} \times \mathcal{F}_{a1} + \mathcal{F}_b$ $\mathcal{F}_{mul} = \mathcal{F}_{out1} \times \mathcal{F}_{out2}$ $\mathcal{F}_{add} = \mathcal{F}_{out1} + \mathcal{F}_{out2}$ (4.2)	$\mathcal{F}_{out1} = (\sigma(\mathcal{F}_{a1}) \times \mathcal{F}_{a2}) + \mathcal{F}_a$ $\mathcal{F}_{out2} = (\sigma(\mathcal{F}_{b1}) \times \mathcal{F}_{b2}) + \mathcal{F}_b$ (4.3)
Final	$\mathcal{F}_{p1} = \text{Conv}_{6\rightarrow 6}(\mathcal{F}_{mul})$ $\mathcal{F}_{p2} = \text{Concat}(\mathcal{F}_{p1}, \mathcal{F}_{add})$ (4.4) $\mathcal{F}_{final} = \text{Norm}(\text{Conv}_{12\rightarrow 3}(\mathcal{F}_{p2}))$	$\mathcal{F}_{p1} = \text{Concat}(\mathcal{F}_{out1}, \mathcal{F}_{out2})$ $\mathcal{F}_{final} = \text{Conv}_{12\rightarrow 3}(p1)$ (4.5)

Table 4.1: Learnable fusion mechanisms: MFB and BGF.

basic version. Rahman et al. in [67] utilized MFB for the detection of activities in videos.

Bilateral Guided Aggregation (BGA) used in [84] and [11] to succeed in semantic segmentation tasks, keep the semantic features of the objects in the data and mix frontal view capabilities with bird's eye view capabilities in a camera-lidar fusion network. The main innovation of BGA is bringing cross-model embedding attention ideas to implementable sub-models.

We deployed both MFB and BGF, discussing different implementations and their computational costs in the system. Both fusion mechanisms are illustrated in Figure 4.5 and formalized in Table 4.1.

Multi-modal Factorized Bilinear Pooling: RGB image ($\mathcal{I}_{rgb}$) and lidar features ($\mathcal{I}_{lidar}$) are inputs of this module in the early Fusion. In Equation 4.1 A convolutional layer converts the number of channels of each input from 3 to 6 where ($\mathcal{F}_a$) and ($\mathcal{F}_b$) are the products. In the middle of this fusion Equation 4.2, $\{\mathcal{F}_{a1}, \mathcal{F}_{a2}, \mathcal{F}_{b1}, \mathcal{F}_{b2}\}$; Convolutional layers are responsible for the formation of these filters, and the size of the channel which is 6 stays unchanged; also addition and multiplication are enforced to $\mathcal{F}_{out1}$ and $\mathcal{F}_{out2}$. In the final part Equation 4.4, the product obtained is utilized as an input to a convolutional layer that consists of six filters' output, concatenated with previous addition results and charged into a last convolutional layer which diminishes the channel size from 12 to 3, after which the figures go through power as well as L_2 normalization.

Bilateral Guided Fusion: As illustrated in the Table 4.1 and Figure 4.5 initial step

1 is similar to MFB, input channels are initially transformed into a high-dimensional space via convoluting (Equation 4.1), then given to additional convolutions that generate four feature maps: $\{\mathcal{F}_{a1}, \mathcal{F}_{a2}, \mathcal{F}_{b1}, \mathcal{F}_{b2}\}$. However, a combination of all four feature maps employs Equation 4.5 with a mixture of the sigmoid, addition, and multiplication to build results with three channels.

4.3 Implementation Details

We will discuss here the steps that are involved in our sensor fusion 2D object detection experiments using RGB and lidar data sources. The preprocessing that was done in order to make the lidar point cloud projected in the same size as the RGB image, the dataset that was used, and the metrics that were used to evaluate the various detectors will all be covered in this section. In addition to that, we will go over the various training methods that are utilized for the detectors.

4.3.1 Projection of Lidar Points

The process of projecting a three-dimensional lidar point cloud onto a frontal view can be broken down into a few steps, as follows:

1. Extract a projection matrix that could be obtained by KITTI's annotation denoted by P, which will be used to represent both the intrinsic and extrinsic parameters of the

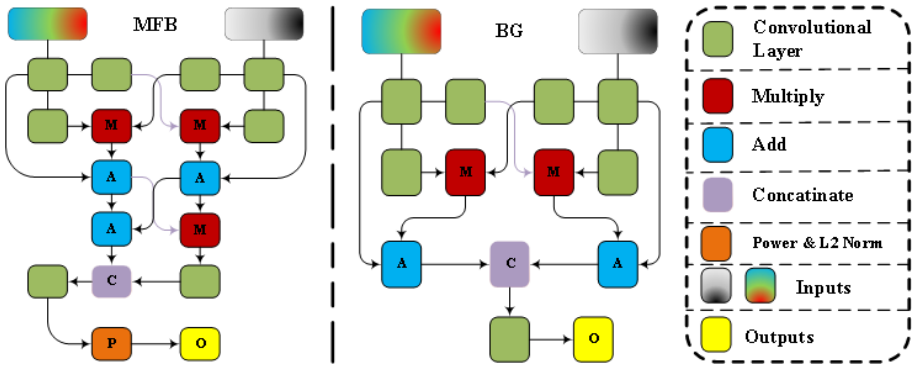

Figure 4.5: The structure of two learnable fusion mechanisms: MFB and BGF.

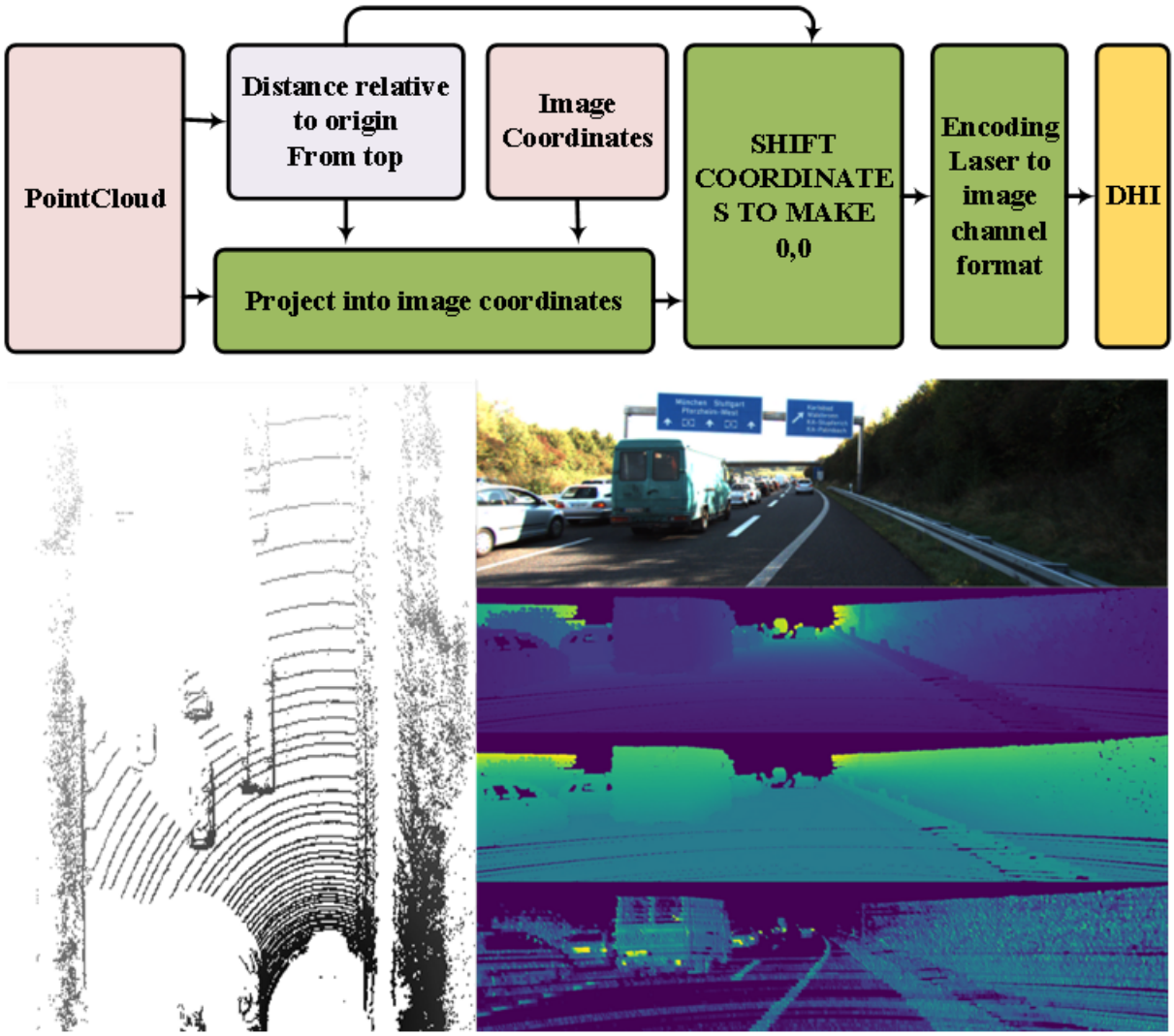

Figure 4.6: Decoding Channels

lidar sensor for each frame. The definition of the projection matrix is as follows:

$$P_{4\times4} = \begin{pmatrix} f_{11} & 0 & c_u & -f_u b_x \\ 0 & f_v & c_v & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.6}$$

where u and v are points within the camera, x, y, and z are points within the point cloud, and b is a baseline that is determined by the camera.

2. Transform the 3D lidar point cloud coordinates from the sensor frame to the camera frame. This can be done by multiplying each point in the point cloud by the extrinsic matrix which is included in the dataset, which represents the transformation from the lidar sensor frame to the camera frame.

3. In order to project the modified point cloud onto the image plane, multiply each point with the projection matrix P. The following equation can be used to illustrate this concept, which can be applied to every individual point in the point cloud in equation 4.7, where X_img represents the projected point in two dimensions in the image plane and X_cam represents the transformed point in three dimensions in the camera frame.

$$X_{img} = P * X_{cam} \tag{4.7}$$

4. After this transformation, the point will have x,y,z intensity values, using the x and y to discard the points that are out of the image plane. The distance between the lidar sensor and a particular point in the environment can be thought of as a measurement of depth on a relative scale. The calculation for it involves taking the z coordinate of a particular point, then dividing that number by 70 because the depth in the image is 70 meters. The reflective quality of a point in the environment can be evaluated using a metric called intensity. To determine it, take the x coordinate of the point that is being used in the calculation. A point's height in relation to its surroundings can be measured using height as a relative measurement to 4 meters height from the ground. It can be computed by beginning with the y coordinate of a particular point, deducting 1, and then dividing the result by 4.

4.3.2 Datasets and Metrics

Our study involves a comprehensive examination of the impact of early sensor fusion systems; through the training and evaluating of four distinct models utilizing the KITTI 2D object detection dataset [18]. This benchmark contains data collected in autonomous driving scenarios, with instance labels provided for object categories such as cars, pedestrians, cyclists, among others. To carry out our experiments, we rely on KITTI's validation set, following the same split suggested in [7].

We measure the detection accuracy of each model using the average-precision (AP%) metric, and report results for three different detection difficulty levels (easy, moderate,

hard). To further study the behavior and performance of the models, we present an ablation study for the four different approaches, to identify the impact that each of their components has on the overall detection accuracy.

4.3.3 Training Strategy

ResNet-based model variants are trained using the popular SGD algorithm. SGD is an iterative optimization algorithm which updates the parameters of the network in each iteration. Specifically, the optimizer applies a linear search on the weights of the network in order to minimize the value of the loss function used for training. Meanwhile, Swin Transformer-based model is trained using the recently introduced decoupled AdamW [54] algorithm. This optimizer performs the bias correction on the first and the second moments, thus allowing the model to be trained faster. Additionally, the AdamW optimizer can be used to reduce memory requirements when training deep architectures.

All of the architectures were designed to perform a two-dimensional detection task encompassing the classes of "Car", "Pedestrian", and "Cyclist". The training of the model was conducted over a period of 30 epochs, during which horizontal flipping was implemented as a data augmentation technique. This strategy increases the amount of training data, which in turn leads to better generalization and improved performance of the model. Apart from this, several regularization techniques such as weight decay and dropout, are also utilized in order to reduce the overfitting of the model.

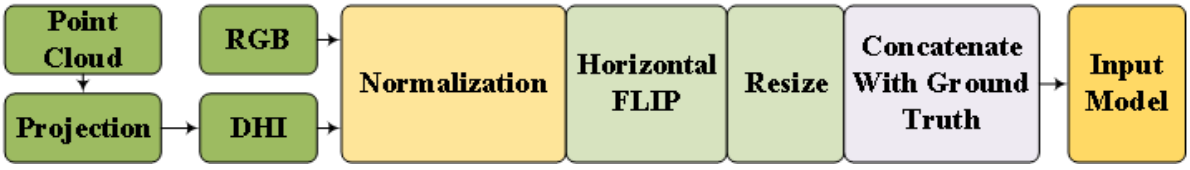

Figure 4.7: Pre-training Stages

4.4 Experimental Results

This section delves into the ramifications of diverse operators utilized for early fusion in relation to distinct 2D detection networks. Additionally, supplementary investigations are conducted to evaluate mid-level fusion and ascertain the precision and efficacy of individual fusion operators [62].

4.4.1 Evaluation of Early Sensor Fusion

Table 4.2 presents a comparison of the influence of five distinct early fusion operators on the detection precision of four 2D object detectors. The initial condition involved the utilization of "RGB" as the baseline for detection, whereby no lidar information was incorporated, thereby precluding early fusion operation. Different types of fusion operations are available, such as (1) "Add" for element-wise addition, (2) "Concat" for concatenation, (3) "Multi" for element-wise multiplication, (4) "BGF" for bilaterally-guided fusion, and (5) "MFB" for multi-modal factorized bilinear pooling.

In the initial round of assessments utilizing the R30 Simple Neck Faster-RCNN detector, the operator MFB demonstrated superior performance compared to other operators in the early fusion across all levels of complexity and the three pertinent categories. In comparison to the fusion operator ranked second, the MFB demonstrated improvements in the moderate difficulty level for cars, pedestrians, and cyclists, with increments of (+4.37%), (+3.76%), and (+5.37%), respectively. The R30 FPN Faster-RCNN detector was utilized in the subsequent set of experiments. The results indicate that the concatenation of features yielded superior outcomes for the pedestrian and cyclist categories. In contrast, the BGF exhibited better performance for the car category and the easy difficulty level for the cyclist class. Feature concatenation and BGF can be deemed equivalent. In examining the moderate difficulty level, it was observed that the BGF exhibited a superior performance by (+0.68%) for the car category. However, it demonstrated a lower performance of (-0.42%) and (-2.06%) for the pedestrian and cyclist classes compared to Concatenating features. The study found that when combined with R30-FPN Cascade-RCNN, the fusion method known as BGF outperformed other fusion methods across all difficulty levels for both car and pedestrian classes. Additionally, the fusion method MFB demonstrated superior performance compared to other methods for the Cyclist class across all difficulty levels.

The Swin Transformer Tiny FPN Cascade-RCNN detector was examined, and it was observed that the car class outcomes utilizing MFB fusion surpassed those of other early fusion techniques. Similarly, the investigation on pedestrian classification demonstrated that the technique of feature concatenation exceeded alternative methods, exhibiting patterns akin to those observed in ResNet30-FPN Faster-RCNN. Finally, the investigation of the cyclist class exhibited a comparable tendency to that of the ResNet30-FPN Faster-RCNN case, wherein the BGF beat the remaining approaches.

Based on the previous findings, it is evident that fusion techniques that are MFB and BGF exhibit a superiority over simple fusion methods in the context of car detection. In general, the combination of features through concatenation yields satisfactory outcomes

Object Detector	Fusion Process	Car (AP_{70}%)			Pedestrian(AP_{50}%)			Cyclist(AP_{50}%)		
		Easy	Mod	Hard	Easy	Mod	Hard	Easy	Mod	Hard
R30 Simple Neck Faster RCNN	RGB	78.04	56.03	47.25	29.04	24.55	20.93	1.54	0.77	1.19
	Add	77.45	59.18	48.34	30.73	23.25	19.52	2.28	1.20	1.22
	Concat	83.48	65.74	56.05	36.63	27.60	23.97	6.95	3.89	4.22
	Multi	77.09	56.39	47.51	28.98	22.18	18.71	9.44	3.93	3.98
	BGF	84.82	66.73	58.44	34.09	28.36	24.76	14.16	7.58	7.24
	MFB	**85.52**	**71.10**	**59.20**	**37.20**	**32.12**	**29.98**	**16.83**	**12.95**	**10.17**
R30 FPN Faster RCNN	RGB	89.92	81.26	72.56	63.18	53.12	46.18	19.17	12.64	12.41
	Add	92.57	81.88	71.70	54.27	45.32	38.28	30.07	24.33	22.99
	Concat	93.48	84.21	74.42	**65.59**	**55.41**	**47.94**	40.40	**32.50**	**31.24**
	Multi	87.39	70.25	62.60	58.18	48.91	42.20	28.37	21.26	19.91
	BGF	**93.79**	**84.89**	**76.15**	64.57	54.99	47.45	**43.89**	30.44	29.11
	MFB	92.93	82.37	72.23	64.12	53.15	47.11	38.14	25.15	24.60
R30 FPN Cascade RCNN	RGB	92.49	84.38	73.75	62.33	52.89	44.90	31.71	19.56	18.99
	Add	93.82	85.42	75.33	63.62	52.61	44.31	42.91	24.45	23.02
	Concat	94.49	87.50	77.74	70.21	58.98	50.41	55.07	35.91	34.72
	Multi	90.90	77.23	67.53	64.85	54.44	46.38	35.13	19.55	18.93
	BGF	**94.90**	**88.40**	**78.38**	**71.25**	**61.11**	**52.48**	**57.42**	37.31	35.33
	MFB	93.21	85.34	76.93	66.88	56.13	48.53	61.74	**38.69**	**37.11**
Swin FPN Cascade RCNN	RGB	93.85	81.66	71.56	58.98	50.02	43.12	31.51	18.28	17.48
	Add	90.35	77.76	67.78	59.50	50.70	43.29	32.37	19.31	19.35
	Concat	94.87	86.36	77.76	**71.30**	**60.83**	**52.05**	44.66	29.35	28.15
	Multi	91.49	76.05	66.00	60.73	50.62	43.50	19.68	11.14	11.56
	BGF	95.38	85.81	77.68	67.66	56.78	47.90	**51.66**	**31.90**	**30.22**
	MFB	**96.57**	**88.12**	**78.13**	67.20	58.10	50.11	35.25	20.77	20.19

Table 4.2: Examining the precision in two-dimensional detection when utilizing diverse fusion operators at the beginning phases with versions of Faster-RCNN and Cascade-RCNN.

and outperforms fusion techniques that are learnable for specific pedestrian classes in certain scenarios.

4.4.2 Evaluation of Mid-level Sensor Fusion

The testing of mid-level fusion involves the concurrent execution of two distinct ResNet30 models, followed by integrating the resulting features. Table 4.3 presents a comparison between the outcomes of early and mid-level fusion utilizing the identical base architecture of the ResNet30 FPN Faster-RCNN 2D object detector. The results indicate that, for car, cyclist and pedestrian classes, simpler methods, such as element-wise addition or feature

Object Detector	Fusion Operator	Car (AP_{70}%)			Pedestrian (AP_{50}%)			Cyclist(AP_{50}%)		
		Easy	Mod	Hard	Easy	Mod	Hard	Easy	Mod	Hard
R30 FPN Faster RCNN **Early**	RGB	89.92	81.26	72.56	63.18	53.12	46.18	19.17	12.64	12.41
	Add	92.57	81.88	71.70	54.27	45.32	38.28	30.07	24.33	22.99
	Concat	93.48	84.21	74.42	**65.59**	**55.41**	**47.94**	40.40	**32.50**	**31.24**
	Multi	87.39	70.25	62.60	58.18	48.91	42.20	28.37	21.26	19.91
	BGF	**93.79**	**84.89**	**76.15**	64.57	54.99	47.45	**43.89**	30.44	29.11
	MFB	92.93	82.37	72.23	64.12	53.15	47.11	38.14	25.15	24.60
R30 FPN Faster RCNN **Mid-level**	RGB	90.44	79.97	70.12	53.40	46.13	39.33	23.83	14.88	14.65
	Add	93.14	**83.80**	**73.88**	**67.84**	**59.38**	**51.40**	**40.66**	**29.69**	**28.23**
	Concat	**93.97**	83.77	73.65	67.12	58.18	50.53	31.10	24.56	23.26
	Multi	93.70	82.87	72.64	63.64	54.25	46.46	28.78	20.01	19.43
	BGF	93.41	82.50	72.44	60.75	51.91	44.72	29.10	19.71	19.15
	MFB	93.75	82.87	72.08	59.80	50.54	43.00	18.52	14.02	13.52

Table 4.3: The average precision for 2d detection of the ResNet30 FPN Faster-RCNN object detector was compared between two approaches: early fusion, where fusing was done with the input before the ResNet30 model, and mid-level fusion, where fusing was done with the outputs of two stream ResNet30 models.

concatenation, outperform more complex learnable fusion mechanisms in mid-level fusion. In the context of pedestrian detection, it has been observed that mid-level fusion utilizing addition outperforms early fusion. Specifically, the former approach for the pedestrian class results in an improvement of (+2.25%), (+3.97%), and (+3.46%) in the easy, moderate, and hard difficulties, respectively.

4.4.3 Complexity Analysis

4.4.3.1 Computational Cost

Table 4.4 analyses the five object detectors currently available without any form of fusion. The computational complexity of performing element-wise addition and multiplication operations on raw RGB data is equivalent. ResNet30-FPN Faster-RCNN exhibits a relatively smaller footprint in terms of learnable parameters, totaling to 18.4M, as opposed to Swin Transformer Tiny FPN Cascade-RCNN, which has a capacity of 72.5M.In addition, it has been observed that ResNet30 Simple Neck Faster-RCNN exhibits a higher level of efficiency, with a four-fold smaller in GFLOPS compared to Swin Transformer Tiny FPN Cascade-RCNN. Furthermore, ResNet30 Simple Neck Faster-RCNN demonstrates a 4.4

times faster FPS rate. However, it is worth noting that its performance in the Car Class category under Easy difficulty is comparatively lower, with a 15 percent deficit.

Table 4.4: Assessment of the parameter numbers, floating point operations per second (FLOPS) of various models as well as their speed based on frame per second on inference on NVidia RTX 2080ti. The precision results were evaluated on the Car class and easy Difficulty in KITTI 2d Detection.

	Computation Metrics		Speed	Performance
Model	Parameters (Million)	FLOPS (Giga)	Rtx 2080 Ti (FPS)	Car Easy
R30 SN Faster-RCNN	18.88	**79.44**	**46.2**	78.04
R30 FPN Faster-RCNN	**18.4**	191.71	23.1	89.92
2 Stream R30 FPN Faster-RCNN	20.87	217.78	20.2	90.44
R30 FPN Cascade-RCNN	46.2	219.51	15.8	92.49
Swin Tiny FPN Cascade-RCNN	72.5	336.13	10.8	**93.85**

The correlation between the number of parameters and GFLOPS in different fusion techniques is noteworthy. Figure 4.8 examines the differences between concatenation and two learnable fusion techniques, MFB and BGF. The evaluation is conducted using the aforementioned techniques for three distinct structures, namely ResNet30 Simple Neck Faster-RCNN, ResNet30 FPN Faster-RCNN, and Swin Transformer Tiny FPN Cascade-RCNN. The concatenation process generally does not introduce additional parameters to the underlying architecture, except the Swin Transformer Tiny FPN Cascade-RCNN. On the other hand, MFB and BGF. introduce additional parameters to the framework. The results of concatenation in terms of increasing GFLOPS exhibit a comparable trend to both MFB and BGF. and occasionally surpass them.

4.4.3.2 Kernel Size of MFB

This analysis aimed to evaluate the effectiveness of two distinct kernel sizes (1×1 and 3×3) by utilizing a ResNet30 FPN Faster-RCNN framework. Table 4.5 demonstrates that the employment of a larger kernel size leads to increased precision for the car, pedestrian, and cyclist classes, except for the pedestrian classification in moderate difficulty conditions.

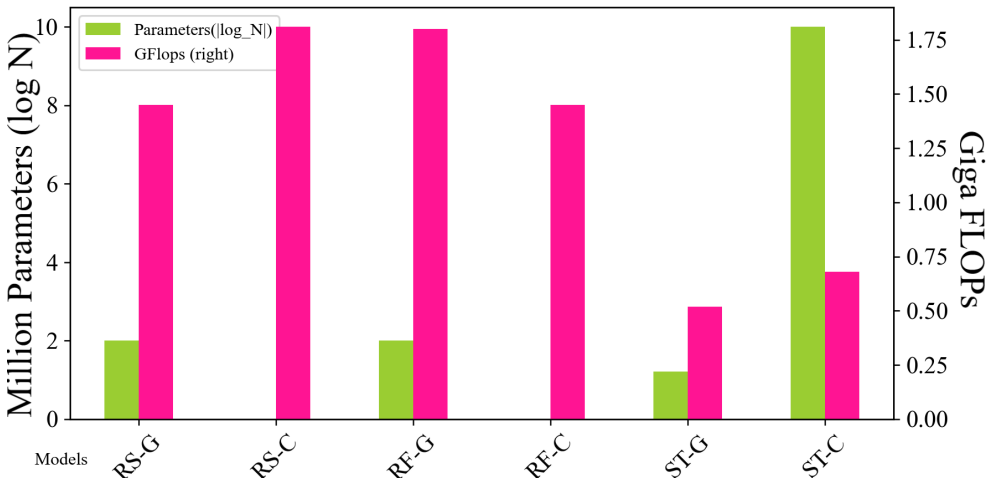

Figure 4.8: Detecting a correlation between the logarithmically scaled quantity of parameters and the corresponding GFLOPS output when utilizing various fusion operators. Although incorporating learnable fusion mechanisms increases the number of learnable parameters within the architecture, their GFLOPS remains comparable and, in certain instances, superior to feature concatenation. "R" contributes to the ResNet backbone using the Faster-RCNN model, "S" to Simple Neck, "F" to utilizing FPN, "ST" to using Swin Transformer tiny, "G" to learnable fusion, and "C" to Concatenation.

Table 4.5: Analyzing the precision of various kernel sizes in MFB Fusion module on the ResNet30 FPN Faster-RCNN architecture, investigations were carried out using two different kernel sizes: K_1 as 1x1, and K_3 as 3x3. Experiments included both mid-level and early-level fusions.

Exp	K_1	K_3	Mid	Car $AP_{70\%}$			Pedestrian $AP_{50\%}$			Cyclist $AP_{50\%}$		
				Easy	Mod	Hard	Easy	Mod	Hard	Easy	Mod	Hard
(a)	√		√	92.93	82.01	71.42	58.8	50.43	42.11	16.04	10.65	10.33
(b)		√	√	**93.75**	**82.87**	72.08	59.8	50.54	43.00	18.52	14.02	13.52
(c)	√			92.71	81.48	71.82	63.66	**54.12**	46.28	31.72	23.90	22.82
(d)		√		92.93	82.37	**72.23**	**64.12**	53.15	**47.11**	**38.14**	**25.15**	**24.60**

4.4.4 Discussion

The findings depicted in Figures 4.9 and 4.10 demonstrate that the BGF early fusion technique, which combines RGB and DHI, exhibited superior performance in detecting pedestrians and cars in the visual scene compared to the sole utilization of the RGB camera image. Despite encountering challenges in detecting objects in the environment, the fusion algorithm exhibited superior performance in distinguishing pedestrians and cars, indicating enhanced accuracy.

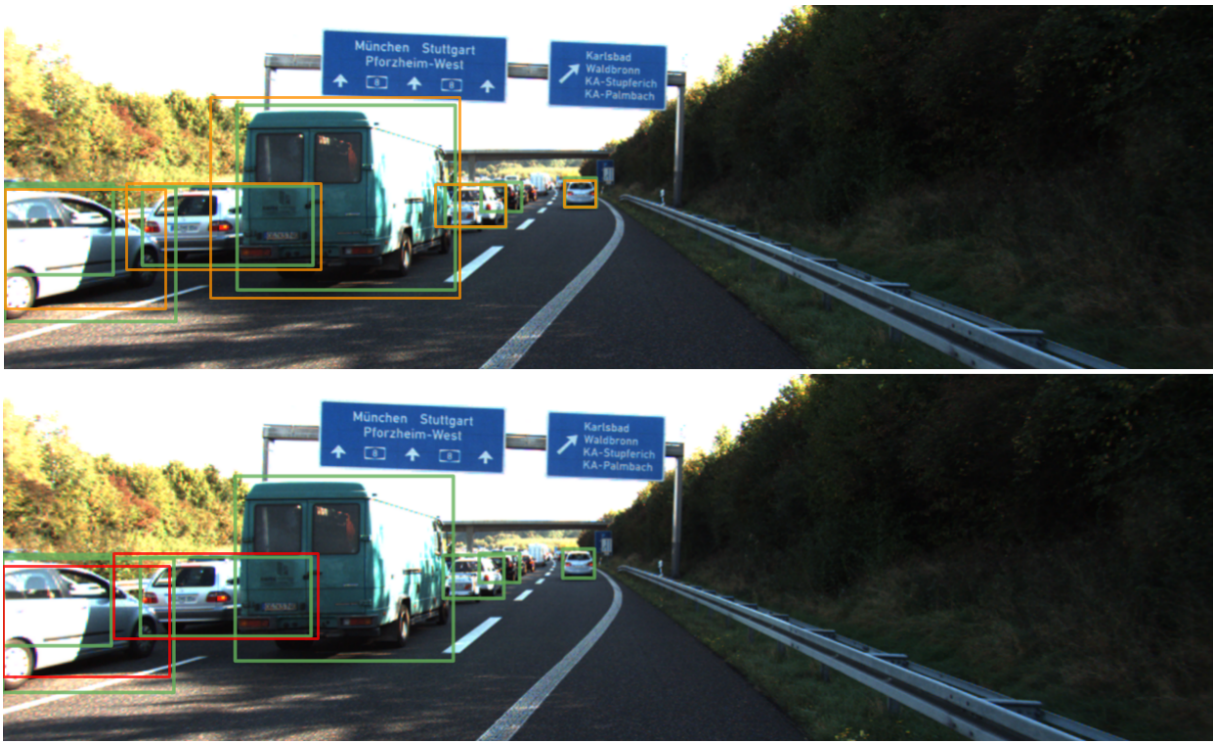

Figure 4.9: In the complex Car scene, RGB results in red and BGF results in orange.

To address the inquiries presented in Section 1.4, the response to the initial question is as follows: The primary differentiation between early and mid-level fusion in the context of this multi-model sensor fusion implementation pertains to the specific stage at which the fusion process occurs. The process of early fusion entails amalgamating unprocessed sensor data from various sources into a solitary representation. On the other hand, mid-level fusion involves merging intermediate-level characteristics that have been extracted from each modality before being fused together. The responses to the second and third

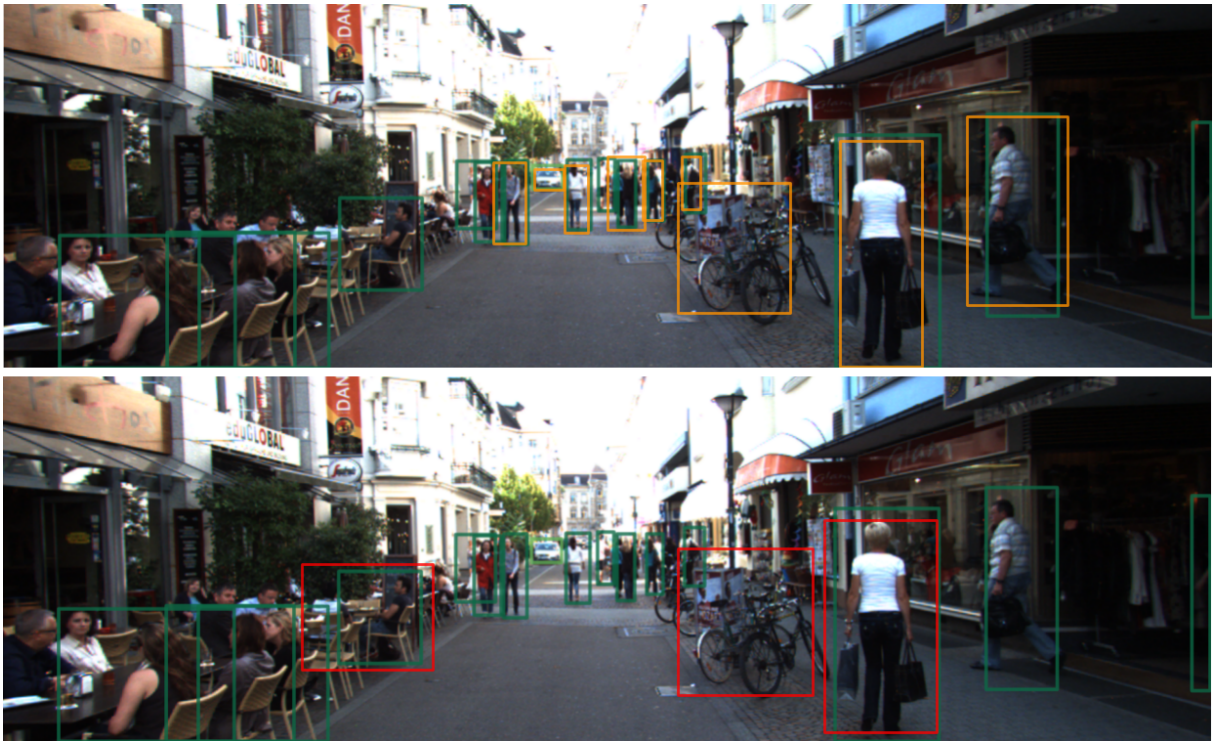

Figure 4.10: In the complex pedestrian scene, RGB results in red and BGF results in orange.

questions are as follows: The precision of various architectures in early sensor fusion can be significantly impacted by choice of fusion operators. The use of simple operators such as element-wise multiplication can often lead to a loss of information. The utilization of learnable fusion operators has the potential to capture intricate intermodal interactions. However, this approach also entails increased computational complexities and necessitates a larger corpus of training data to prevent overfitting. In response to the last question, it is worth noting that the decision to employ basic fusion techniques such as addition and multiplication, as opposed to more complex learnable fusion operators and concatenation, can significantly impact both the training process and the ultimate efficacy of the fusion model. Generally speaking, the former methods tend to be faster, but this choice should be made with careful consideration of the specific goals and requirements of the project at hand.

4.5 Summary of Fusion

In this work, we focused on the use of fusion operators for the task of multi-modal 2D object detection. Fusion operators are techniques used to fuse the features of distinct input modalities or types of data to improve a machine-learning model's precision. In the context of multi-modal 2D object detection, these modalities might include visual features extracted from images or video and other types of data such as text or audio.

The study conducted an in-depth investigation of the efficacy and complexity of diverse fusion operators in different architectures for 2D detection. We found that multi-modal representations that leverage early fusion, or the combination of input features at the earliest stage of the network, provided more interaction between input features and were more effective than mid-level fusion of high-level feature maps. This suggests that early fusion is a more effective way to combine the features of different input modalities for this task.

We also found that the use of MFB, a particular type of fusion operator, resulted in enhancements in performance when implemented in conjunction with a Transformer-based 2D object detector. This suggests that MFB is a particularly effective fusion operator for this task.

However, The element-wise multiplication of two input features negatively impacted the performance of the network. This is because element-wise multiplication causes the gradients, or the mathematical quantities that determine how the model's parameters should be adjusted during training, to vary dramatically in the upstream network. This can make it difficult for the model to learn effectively, resulting in poor performance. In contrast, We found that mid-level fusion using a simple element-wise addition fusion resulted in higher performance. This suggests that element-wise addition is a more effective fusion operator than element-wise multiplication for mid-level fusion in this task.

Overall, this work provides valuable insights into the effectiveness and efficiency of different fusion operators for the task of Multi-modality (multi-sensor) 2D object detection. We indicate that early fusion can be more effective than mid-level fusion and that MFB and element-wise addition are particularly effective fusion operators for this task under certain classes and difficulties. The utilization of element-wise multiplication is not recommended due to its negative impact on the performance of the model.

Chapter 5

Conclusion

In this study, the pruning technique was applied to three state-of-the-art models in the field of instance segmentation and the results were analyzed. The results showed that the Neural Pruning via Growing Regularization (GREG) method was superior to the other methods in terms of accuracy improvement in the classification tasks. The attenuation Network Pruning method was superior to the other methods regarding accuracy improvement in the detection tasks. The pruning process caused a drastic reduction in the models' size while still maintaining good performance. However, the decrease in channel size did not result in the expected increase in speed. Instead, the study emphasized the importance of researching various pruning algorithms and the results they produce.

The study also investigated the use of fusion operators for multi-modal 2D object detection. The results showed that early fusion provided better interaction between input features and was more effective than mid-level fusion. The use of multi-modal factorized bilinear pooling (MFB) and Bilateral Guided Fusion (BGF) resulted in performance improvements when used with a Transformer-based 2D object detector. On the other hand, using element-wise multiplication negatively impacted the network's performance. Finally, the study suggests that element-wise addition is a more effective fusion operator for mid-level fusion.

Multi-modal fusion and neural network pruning can improve machine learning models' robustness, interpretability, and generalization performance while modelling brain functions and providing insights into the neural mechanisms underlying perception, cognition, and behaviour. Pruning comes from brain forgetting, synaptic pruning, and sensor fusion comes from fusion. These strategies may increase machine learning model robustness by lowering the effect of noisy or irrelevant information, interpretability by reducing complex-

ity, and generalization performance by reducing complexity and efficiency. However, data and model concerns may arise and need careful thought and implementation. In addition, these methods may be used to represent brain activities, including sensory integration and brain plasticity.

There is still much room for improvement when it comes to pruning algorithms. Future work could focus on exploring new pruning methods that could improve upon the accuracy and speed of the pruned models. For example, researchers could investigate the use of reinforcement learning algorithms to guide the pruning process or try combining different pruning methods to achieve even better results. Another avenue for future work could be to focus on pruning models for other tasks, such as natural language processing or reinforcement learning, to see if the results are transferable across domains.

It would be interesting to evaluate other fusion operators yet to be considered in this study. Furthermore, the study could be extended to other tasks or modalities, and the effects of the fusion operators could be analyzed in more detail. Additionally, it would be worthwhile to investigate the impact of the fusion operators on the training process and the overall complexity of the models. It would be worth investigating the use of fusion operators for other tasks, such as instance segmentation, object tracking or semantic segmentation. This could provide valuable insights into the generalizability of the results found in this study and the potential for using fusion operators in other domains.

In conclusion, this study provides valuable insights into the effectiveness and efficiency of different fusion operators for the task of multi-modal 2D object detection and pruning for object detection. The results suggest that early fusion, MFB, BGF and element-wise addition are particularly effective fusion operators for this task, while the use of element-wise multiplication should be avoided. Further research is needed to fully understand the potential and limitations of these fusion operators.

References

[1] M Augasta and Thangairulappan Kathirvalavakumar. Pruning algorithms of neural networks—a comparative study. *Open Computer Science*, 3(3):105–115, 2013.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9157–9166, 2019.

[4] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

[5] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4974–4983, 2019.

[6] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[7] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals using stereo imagery for accurate object class detection. *IEEE transactions on pattern analysis and machine intelligence*, 40(5):1259–1272, 2017.

[8] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[9] Maxwell D Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

[10] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.

[11] Jiajun Deng, Wengang Zhou, Yanyong Zhang, and Houqiang Li. From multi-view to hollow-3d: Hallucinated hollow-3d r-cnn for 3d object detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(12):4722–4734, 2021.

[12] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *International Conference on Machine Learning*, pages 1607–1616. PMLR, 2019.

[13] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[15] Pradeep Duraisamy, Venkatesh Babu Sakthi Narayanan, Ramya Patturajan, and Kumararaja Veerasamy. Multi-sensor fusion methods for unmanned aerial vehicles to detect environment using deep learning techniques. In *Computational Intelligence for Unmanned Aerial Vehicles Communication Networks*, pages 263–273. Springer, 2022.

[16] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[17] Hongfei Gao, Yongzhen Huang, Haoran Li, and Qichao Zhang. Multi-sensor fusion perception system in train. In *2021 IEEE 10th Data Driven Control and Learning Systems Conference (DDCLS)*, pages 1171–1176. IEEE, 2021.

[18] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[19] Sharath Girish, Shishira R Maiya, Kamal Gupta, Hao Chen, Larry S Davis, and Abhinav Shrivastava. The lottery ticket hypothesis for object recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 762–771, 2021.

[20] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[21] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[22] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. A survey on instance segmentation: state of the art. *International journal of multimedia information retrieval*, 9(3):171–189, 2020.

[23] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

[24] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[25] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5353–5360, 2015.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[27] Tianxing He, Yuchen Fan, Yanmin Qian, Tian Tan, and Kai Yu. Reshaping deep neural network for fast decoding by node-pruning. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 245–249. IEEE, 2014.

[28] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4340–4349, 2019.

[29] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[30] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[31] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[32] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 646–661. Springer, 2016.

[33] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[34] Woojeong Kim, Suhyun Kim, Mincheol Park, and Geunseok Jeon. Neuron merging: Compensating for pruned neurons. *Advances in Neural Information Processing Systems*, 33:585–595, 2020.

[35] Heiko Koch, Alexander König, Alexandra Weigl-Seitz, Karl Kleinmann, and Jozef Suchỳ. Force, acceleration and vision sensor fusion for contour following tasks with an industrial robot. In *2011 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pages 1–6. IEEE, 2011.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[37] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *2018*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1–8. IEEE, 2018.

[38] Duong H Le and Binh-Son Hua. Network pruning that matters: A case study on retraining variants. *arXiv preprint arXiv:2105.03193*, 2021.

[39] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016.

[40] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[42] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[43] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[44] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[45] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *Proceedings of the European conference on computer vision (ECCV)*, pages 641–656, 2018.

[46] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[47] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[48] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[49] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021.

[50] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[51] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[52] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[53] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022.

[54] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[55] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[56] Zelda Mariet and Suvrit Sra. Diversity networks: Neural network compression using determinantal point processes. *arXiv preprint arXiv:1511.05077*, 2015.

[57] Yahya Massoud. *Sensor fusion for 3D object detection for autonomous vehicles*. PhD thesis, Université d'Ottawa/University of Ottawa, 2021.

[58] Morteza Mousa-Pasandi, Mohsen Hajabdollahi, Nader Karimi, Shadrokh Samavi, and Shahram Shirani. Convolutional neural network pruning using filter attenuation. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 2905–2909. IEEE, 2020.

[59] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? *Advances in neural information processing systems*, 32, 2019.

[60] Tudor Nicosevici, Rafael Garcia, Marc Carreras, and Miguel Villanueva. A review of sensor fusion techniques for underwater vehicle navigation. *Oceans' 04 MTS/IEEE Techno-Ocean'04 (IEEE Cat. No. 04CH37600)*, 3:1600–1605, 2004.

[61] Morteza Mousa Pasandi, Mohsen Hajabdollahi, Nader Karimi, and Shadrokh Samavi. Modeling of pruning techniques for deep neural networks simplification. *arXiv preprint arXiv:2001.04062*, 2020.

[62] Morteza Mousa Pasandi, Tianran Liu, Yahya Massoud, and Robert Laganière. Sensor fusion operators for multimodal 2d object detection. In *Advances in Visual Computing: 17th International Symposium, ISVC 2022, San Diego, CA, USA, October 3–5, 2022, Proceedings, Part I*, pages 184–195. Springer, 2022.

[63] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In *International Conference on Machine Learning*, pages 5113–5122. PMLR, 2019.

[64] Lutz Prechelt. Connection pruning with static and adaptive pruning schedules. *Neurocomputing*, 16(1):49–61, 1997.

[65] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 918–927, 2018.

[66] Siyuan Qiao, Liang-Chieh Chen, and Alan Yuille. Detectors: Detecting objects with recursive feature pyramid and switchable atrous convolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10213–10224, 2021.

[67] Md Atiqur Rahman and Robert Laganière. Mid-level fusion for end-to-end temporal activity detection in untrimmed video. In *BMVC*, 2020.

[68] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[69] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[70] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[71] Ting Rui, Junhua Zou, You Zhou, Jianchao Fei, and Chengsong Yang. Convolutional neural network simplification based on feature maps selection. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 1207–1210. IEEE, 2016.

[72] René Schuster, Christian Unger, and Didier Stricker. A deep temporal fusion framework for scene flow using a learnable motion model and occlusions. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 247–255, 2021.

[73] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[74] Arshdeep Singh, Padmanabhan Rajan, and Arnav Bhavsar. Deep hidden analysis: A statistical framework to prune feature maps. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 820–824. IEEE, 2019.

[75] Peng Sun, Wenhu Zhang, Songyuan Li, Yilin Guo, Congli Song, and Xi Li. Learnable depth-sensitive attention for deep rgb-d saliency detection with multi-modal fusion architecture search. *International Journal of Computer Vision*, 130(11):2822–2841, 2022.

[76] M Tan, R Pang, and QV Le. Efficientdet: scalable and efficient object detection. arxiv. *arXiv preprint arXiv:1911.09070*, 10, 2019.

[77] Anirud Thyagharajan, Benjamin Ummenhofer, Prashant Laddha, Om Ji Omer, and Sreenivas Subramoney. Segment-fusion: Hierarchical context fusion for robust 3d semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1236–1245, 2022.

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[79] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020.

[80] X Wang, T Kong, C Shen, Y Jiang, and L Li. Solo: segmenting objects by locations. arxiv preprint arxiv: 191204488. 2019.

[81] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. Solov2: Dynamic and fast instance segmentation. *Advances in Neural information processing systems*, 33:17721–17732, 2020.

[82] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 244–253, 2018.

[83] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6):2140, 2021.

[84] Changqian Yu, Changxin Gao, Jingbo Wang, Gang Yu, Chunhua Shen, and Nong Sang. Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation. *International Journal of Computer Vision*, 129(11):3051–3068, 2021.

[85] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9194–9203, 2018.

[86] Zhou Yu, Jun Yu, Jianping Fan, and Dacheng Tao. Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In *ICCV*, pages 1839–1848, 2017.

[87] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[88] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[89] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

[90] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13001–13008, 2020.

[91] Ziyuan Zhong, Zhisheng Hu, Shengjian Guo, Xinyang Zhang, Zhenyu Zhong, and Baishakhi Ray. Detecting safety problems of multi-sensor fusion in autonomous driving. *arXiv preprint arXiv:2109.06404*, 2021.

[92] Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *European conference on computer vision*, pages 662–677. Springer, 2016.

[93] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[94] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. *Advances in neural information processing systems*, 31, 2018.

APPENDICES

Appendix A

Mathematical Analysis of Pruning

In this section, we will investigate the mathematical explanation for the effectiveness of CNN pruning by utilizing probabilistic distributions. Probabilistic distributions are a crucial component in the process of establishing methods for pruning. This is because they provide a framework that allows for the accurate and reliable measurement of the value of particular connections in the process of accurately maintaining the network.

A.1 Mathematical Formulation

CNNs are formidable designs for deep learning that are exceptionally effective when it comes to resolving complicated problems that include enormous quantities of data. Traditional CNNs, on the other hand, suffer from high computational complexity and high dimensional parameterization because of their size. This means that training them takes a substantial amount of time and resources and that there are a huge number of parameters that need to be optimized. This may be formally represented as an $|O(N^2)|$ complexity in the number of parameters,N, and an $|O(N^3)|$ complexity in terms of the amount of time required for training and the computational needs.

When starting with a CNN model that has been trained, the process of pruning comprises sorting the parameters in order of importance score S_{ijk} for every operation node f_{ijk} for each layer i, filter channel j, and spatial location k. After that, the model is simplified by iteratively deleting the unnecessary parameters with the lowest relevance score while still satisfying the model performance constraints. That is to say, the weights that have an importance score of $S_{ijk} < t$ are the ones that are removed, where t is a pre-defined threshold.

Let us assume that the set of n weights in a particular CNN is denoted by the variable$A = x_1, x_2, ...x_n$, and that the set of corresponding importance scores for the weights is denoted by the variable$S = s_1, s_2, ..., s_n$. Let us indicate the collection of randomly picked weights with the notation$R = r_1, r_2, ..., r_n$. Understanding where attention is directed during pruning allows us to formulate an expression for the pace of pruning employed by both importance-based and random pruning:

$$\lambda_{imp} = \frac{\sum_{i=1}^{n} s_i}{\sum_{i=1}^{n} x_i}, \lambda_{ran} = \frac{\sum_{i=1}^{n} r_i}{\sum_{i=1}^{n} x_i} \tag{A.1}$$

The effectiveness of importance-based pruning may be compared to that of random pruning by determining the distribution of the ratio between the two pruning rates. The function that best characterizes this probability is as follows:

$$P(\frac{\lambda_{ran}}{\lambda_{imp}}) = \frac{\sum_{i=1}^{n} P(r_i/s_i) \times P(x_i)}{\sum_{i=1}^{n} P(r_i \times x_i)} \tag{A.2}$$

Given that the importance scores are typically higher for the weights that contribute the most to the overall performance of the CNN, it follows that the probability of importance-based pruning being superior to random pruning is higher than the probability of random pruning being superior to importance-based pruning. The next two subsections will discuss the mathematical adequacy of pruning algorithms based on robustness using an L1 versus L2 norm, as well as the outcomes of using pruning with attenuation versus one-shot removing of weights or neurons. We will delve into the mathematics of the algorithms used in each strategy and discuss the advantages and disadvantages of each approach.

A.1.1 Adequacy of L1 Norm

The L1 norm has been used as an element selection criterion in filter and neuron pruning due to its robustness in dealing with noisy data sets. The L1 norm is also known as the least absolute deviations or the Manhattan distance due to its physical interpretation as the sum of the absolute Euclidean distances from each point to the origin. The L1 norm can be calculated for any vector or matrix with elements x_i as follows: $||x||_1 = \sum_{i=1}^{n} |x_i|$ This norm is especially useful for sparse vector or matrix representation where many of the components are zero, thus punishing the large values of L1 more than the small ones. The sparser the vector, the smaller the L1 norm. This makes the L1 norm an attractive choice for selecting elements for filter and neuron pruning.

The robustness of the L1 norm can be mathematically proven by considering two absolute non-zero values x_i and x_j, such that $|x_i| >> |x_j|$. The L1 norm of this vector can be expressed as $\sum_{i=1}^{n} |x_i| = s + |x_i| + |x_j|$, where $s = \sum_{i \neq i,j} |x_i|$.

The L2 norm, which is the squared Euclidean distance, of the same vector is $\sum_{i=1}^{n} x_i^2 = (s + |x_i| + |x_j|)^2 = (s^2 + 2s|x_i| + 2s|x_j| + |x_i|^2 + |x_j|^2)$. This can be further simplied to $|x_i|^2 + 2s|x_i| + 2s|x_j|$.

Therefore it can be seen that the L1 norm is more robust to the outlier value x_i as the magnitude of the outlier has an insignificant effect on the norm whereas, in the L2 norm, the magnitude of the outlier has a significant effect on the Norm.

This robustness of the L1 norm to outlier values makes it an attractive criterion for filter and neuron pruning, where large outlier values need to be ignored. Additionally, the simplicity and efficiency of the L1 norm make it a popular choice in many applications. All these properties make the L1 norm an ideal element selection criterion in the filter and neuron pruning.

A.1.2 Adequacy of Attenuation

Let W represent the weight matrix of an L1-Norm filter and neuron pruning model, A represent the associated adjacency matrix, P the he threshold pruning value, and w_{ij} the weight matrix elements.

The One-Shot pruning algorithm will create a sparse matrix of the same size as the original weight matrix, W', such that:

$$W'_{ij} = \begin{cases} W_{ij}, & \text{if } |w_{ij}| > P \\ 0, & \text{otherwise} \end{cases} \tag{A.3}$$

Attenuation of weak parts or gradual pruning on the other hand adjusts the weights:

$$W_{ij} = \begin{cases} 0, & \text{if } |w_{ij}| \leq P \\ w_{ij} \times (1 - q), & \text{otherwise} \end{cases} \tag{A.4}$$

where q is a particular pruning rate.

To prove that attenuation of weak parts and gradual pruning are superior to One-shot pruning, we must show that $||W_{ij}||'_1 < ||W_{ij}||_1$ when employing attenuation or gradual pruning.

Given the lemma that $||W_{ij}||'_1 = \sum_{i,j=1}^{N} |W'_{ij}| = \sum_{i,j=1}^{N} (|W_{ij}| \times \mathbb{I}(|w_{ij}| > P))$, we can say the following:

$$\begin{aligned} ||W_{ij}||_1 &= \sum_{i,j=1}^{N} |W_{ij}| \\ ||W_{ij}||'_1 &= \sum_{i,j=1}^{N} (|W_{ij}| \times \mathbb{I}(|w_{ij}| > P)) \end{aligned} \tag{A.5}$$

Now, let A be the adjacency matrix of the weights matrix W. We can derive the following equation using the triangle inequality:

$$\begin{aligned} ||W_{ij}||_1 - ||W_{ij}||'_1 &= \sum_{i,j=1}^{N} |W_{ij}| - \sum_{i,j=1}^{N} (|W_{ij}| \times \mathbb{I}(|w_{ij}| > P)) \\ &\leq \sum_{i,j=1}^{N} |W_{ij}| - \sum_{i,j=1}^{N} (|W_{ij}| \times \mathbb{I}(A_{ij} \neq 0)) \\ &= \sum_{i,j=1}^{N} |W_{ij}|(1 - \mathbb{I}(A_{ij} \neq 0)) \\ &< 0 \end{aligned} \tag{A.6}$$

Therefore we can conclude that $||W_{ij}||'_1 < ||W_{ij}||_1$ when employing attenuation or gradual pruning algorithms and therefore they are superior to the One-Shot pruning algorithm.