Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

Кваліфікаційна робота
освітній ступінь – бакалавр
на тему:

# “Dropout for Neural Networks Pruning”
# «Dropout як метод обрізки нейронних мереж»

Виконала: студентка 4-го року навчання
освітньої програми «Прикладна математика»
спеціальності 113 Прикладна математика
Семенець Дарина Віталіївна
Керівник: Швай Н. О.,
кандидат фіз.-мат. наук, доцент

Кваліфікаційна робота захищена з оцінкою

_____

Секретар ЕК

_____

(підпис)

«_____» _____20___р.

Київ – 2025

Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
доцент, кандидат фіз.-мат. наук
_____*Чорней Р.К.*
(підпис)
“____”_____2025

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

для кваліфікаційної роботи

студенту 4-го курсу, факультету інформатики

Семенець Дарини Віталіївни

**Тема:** Dropout for Neural Networks Pruning / Dropout як метод обрізки нейронних мереж

Зміст кваліфікаційної роботи:

Abstract
Introduction
1. Literature Review
2. Methodology and Experimental Setup
3. Experiments and Results
Conclusion
References

Дата видачі “____”_____2025 Керівник _____
(підпис)

Завдання отримав _____
(підпис)

# Графік підготовки кваліфікаційної роботи до захисту

Графік узгоджено «_____» _____2025р.

Науковий керівник _____
*(ПІБ)*

Виконавець кваліфікаційної роботи _____
*(ПІБ)*

| № з/п | Перелік робіт | Термін виконан- ня етапу | Підпис наукового керівника | Дата озна-йомлення наукового керівника | Примітка |
|---|---|---|---|---|---|
| 1. | Отримання теми квалі-фікаційної роботи. | 30.09.2024 | | | |
| 2. | Ознайомлення з темою кваліфікаційної роботи. | 07.10.2024 | | | |
| 3. | Розробка плану та стру-ктури роботи. | 21.10.2024 | | | |
| 4. | Робота з науковою лі-тературою, опис основ-них означень. Написан-ня вступу та анотації. | 28.10.2024 | | | |
| 5. | Дослідження Dropout для обрізки нейронних мереж . | 04.11.2024 | | | |
| 6. | Робота над текстовим оформленням теорети-чної частини та одержа-них результатів. | 18.02.2025 | | | |
| 7. | Попередній аналіз ква-ліфікаційної роботи. Виправлення помилок. | 17.05.2025 | | | |
| 8. | Попередній захист ква-ліфікаційної роботи. | 23.05.2025 | | | |
| 9. | Захист кваліфікаційної роботи. | 06.06.2025 | | | |

# Table of Contents

**ABSTRACT** 7

**INTRODUCTION** 8

**CHAPTER 1. LITERATURE REVIEW** 10

**1.1 NEURAL NETWORK COMPRESSION** 10
1.1.1 CONVOLUTIONAL NEURAL NETWORKS AND THE PROBLEM OF SCALING 10
1.1.2 OVERVIEW OF KEY APPROACHES: PRUNING, QUANTIZATION, KNOWLEDGE DISTILLATION 11
1.1.2.1 Pruning: 11
1.1.2.2 Quantization: 12
1.1.2.3 Knowledge Distillation: 12
1.1.3 MODEL COMPRESSION EVALUATION METRICS 13
**1.2 PRUNING: TYPES AND EVOLUTION OF APPROACHES** 15
1.2.1 CLASSIFICATION OF PRUNING METHODS STRUCTURED, UNSTRUCTURED AND SEMI-STRUCTURED TYPES OF PRUNING 15
1.2.1.1 Unstructured pruning: 15
1.2.1.2 Structured pruning: 16
1.2.1.3 Semi-structured Pruning: 16
1.2.2 MAIN PRUNING STRATEGIES 16
1.2.2.1 Magnitude-based pruning 17
1.2.2.1.1 L2-norm-based Structured Pruning 17
1.2.2.2 Gradient-based pruning 18
1.2.2.3 Regularization-based pruning 18
1.2.3 TIMING OF PRUNING: PRE-TRAINING PRUNING, DURING TRAINING PRUNING AND POST-TRAINING PRUNING 19
**1.3 THEORETICAL AND MATHEMATICAL VIEW OF DROPOUT** 20
1.3.1 INTRODUCTION TO DROPOUT 20
1.3.1.1 Standard Dropout 20
1.3.1.2 Spatial Dropout 21
1.3.2.1 Variational Inference Perspective: 22
1.3.2.2 Dropout Objective as Approximate Variational Loss: 22
1.3.2.3 Predictive Uncertainly and Monte Carlo Dropout 23

**CHAPTER 2. METHODOLOGY AND EXPERIMENTAL SETUP** 24

**2.1 RESEARCH PROBLEM AND METHODS** 24
2.1.1 PROBLEM STATEMENT 24
2.1.2 DROPOUT-BASED PRUNING 25
2.1.2.1 Custom Dropout Implementations 25
2.1.2.1.1 Custom Dropout for Linear Layers 25
2.1.2.1.2 Spatial Dropout for Convolutional Layers 26
2.1.2.2 Dropout Mask Stages 26
2.1.3 L2-NORM PRUNING 28
**2.2 SYSTEM CONFIGURATION** 29
2.2.1 MODEL DESIGN 29
2.2.2 DATA DESCRIPTION AND PREPROCESSING 29
2.2.3 TRAINING PHASE 30

**CHAPTER 3. EXPERIMENTS AND RESULTS 31**

**3.1 EXPERIMENT PREPARATION 31**
3.1.1 DATASET DESCRIPTION AND PREPROCESSING 31
3.1.2 MODEL DESIGN 32
**3.2 EXPERIMENT 1: DROPOUT-BASED VS. L2-NORM PRUNING 33**
3.2.1 FOCUS AND RESEARCH QUESTIONS 33
3.2.2 EXPERIMENT 1.A: COMPARISON OF DROPOUT-BASED VS. L2-NORM PRUNING (NO MASK CONTROL) 33
3.2.2.1 Initial Training Setup 33
3.2.2.2 Results for Dropout-based Pruning 34
3.2.2.3 Results for L2-Norm Structured Pruning 35
3.2.2.4 Result Comparison 36
3.2.2.5 Fine-Tuning After Pruning: Enhancement of Model Performance 37
3.2.3 EXPERIMENT 1.B: COMPARISON. OF DROPOUT-BASED VS. L2-NORM PRUNING (WITH CONTROL MASK POOL) 38
3.2.3.1 Initial Training Setup 38
3.2.3.2 Result Comparison 39
3.2.2.5 Fine-Tuning After Pruning: Enhancement of Model Performance 40
**3.3 EXPERIMENT 2: STABILITY AND VARIABILITY OF DROPOUT-BASED PRUNING 41**
3.3.1 FOCUS AND RESEARCH QUESTION 41
3.3.2 FULL RANGE OF DROPOUT RATES AND MASKS WITH SEED-CONTROLLED TRAINING AND UNCONTROLLED TRAINING 41

**CONCLUSION 43**

**REFERENCES: 44**

## Abstract

In this study, the hypothesis is examined whether Dropout masks can be used for structural pruning without further evaluating the importance of individual filters or weight. It was decided to compare a Dropout-based approach, which is based on the utilization of binary Dropout masks, with a classical L2-Norm-based pruning method. For this task, we manually designed an architecture of a convolutional neural network with a custom Dropout. The research undergoes the following phases: designing a mask generation mechanism, preprocessing data, training of model, implementing of pruning algorithms, and conducting experiments using the Imagenette2 dataset. Our idea is to determine whether Dropout pruning can offer a reliable alternative to traditional methods, especially under different levels of sparsity and stochasticity.

Keywords: Pruning, Dropout, L2, Convolutional Neural Network, Algorithm, Training, Binary Mask, Seed.

# Introduction

Although CNNs have shown steady progress in the plethora of machine learning tasks. It caused increase in model sizes, high memory usage and longer training time.

The deployment of deep networks in environments with limited resources, such as mobile devices, real-time applications and etc, becomes a very tough mission due to these issues. Researchers have proposed model compression methods to tackle these problems. One of the most popular approaches among scientists is pruning, which aims to remove superfluous parameters.

On the one hand, typical traditional pruning methods, such as L1/L2 norms, gradient sensitivity or regularization (for more information refer to Section 1.2.2) algorithms, rely on computing importance scores for weights, filters or neurons. Nevertheless, they require extra computational expenses during the pruning process and often fine-tuning to recover accuracy.

On the contrary, Dropout offers a simple way to reduce overfitting. As a stochastic regularization technique, it randomly zeroes out neurons or even feature maps during training. This behavior promotes sparsity within the network, which in turn resembles the effect of pruning.

This leads to the question:

*Can the dropout binary masks be used after training to carry out pruning, without evaluating the importance of individual weights or filters?*

**Objective of Study:**

We explore the potential of stochastic dropout masks for pruning in convolutional neural networks (CNNs). The study aims:

- to inspect whether stochastic Dropout binary masks can be used for structured pruning of CNNs

- to compare the effectiveness of dropout-based pruning with custom L2-norm pruning method in terms of sparsity and model accuracy
- to analyze the variability and stability of the Dropout pruning results with respect to different dropout rates and random seeds

**Research Tasks and Expected Outcomes:**

To address the main objectives, this study designs and trains convolutional neural networks with tailored dropout layers. It then applies both dropout-based and L2-structured pruning techniques to compare their performance under varying dropout rates, random seeds and sparsity ratio.

The aim is to assess how dropout-based pruning impacts model size and accuracy, and how stable the results are across different configurations. Additionally, that another goal is to explore whether dropout-guided pruning can be used reliably in practice.

# Chapter 1. Literature Review

## 1.1 Neural Network Compression

### 1.1.1 Convolutional Neural Networks and the Problem of Scaling

**Convolutional neural networks (CNNs)** are a class of artificial neural networks designed for processing data such as images. CNNs are composed of three main types of layers: convolutional, pooling and fully connected layers. Whereas the convolutional layers extract spatial hierarchies of features through learnable filters, pooling one reduce dimension and fully connected layers make final classification.

The powerful side of CNNs is their ability to capture spatial and local correlations in data thanks to using shared weights and receptive fields. This allows CNNs to learn complex patterns, which makes them a standard tool in image classification, object detection and other computer vision tasks. [1]

However, CNNs tend to become deeper and wider due to the substantial increase in complexity of tasks. This may lead to a massive growth in the number of parameters and required computational resources. For example, modern architectures such as LLaMA and PaLM incorporate millions of parameters, which makes them too resource-intensive in the learning and inference processes. [2]

This increase in scale leads to three major challenges:

- **Memory consumption:**

"CNNs require substantial memory to store the data necessary to backpropagate the error (gradients of the activations with respect to the loss), and information about the computation of the gradients of the model parameters." [3]

- **Energy efficiency**:

As stated in the article "Carbon Emissions and Large Neural Network Training" a model like GPT-3 (with 175 billion parameters) consumes 1.287 megawatt-hours of electricity and generates 552 tons of carbon emissions [4].

- **Infrastructure Costs:**

High-performance hardware, such as GPU or TPUs, is essential, as it is the only way to handle the computational demands of large-scale CNNs. It typically involves significant investment-whether through cloud-based or on-premises systems.

In response to the discussed issues model compression techniques have been developed. LeCun et al. (1989) were the first to propose the Optimal Brain Damage method, whose concept is to prune network weights based on their "saliency" score to reduce redundancy without serious harming performance [5]. This approach gave a rise for a boarder field of model compression: pruning, quantization and knowledge distillation.

### 1.1.2 Overview of Key Approaches: Pruning, Quantization, Knowledge Distillation

All of the model compression techniques have a common target, but the ways in which they achieve it differ significantly. Among the most widely used methods are pruning, quantization and knowledge distillation. They represent three principal areas in the field of neural network compression.

#### 1.1.2.1 Pruning:

Pruning itself refers to the selective removal of neural network parameters (neurons and weights), which do not influence much to the network's prediction. This process results in a more efficient and compressed model. One of the common pruning techniques is neuron pruning (depicted in Figure 2), which involves deleting entire neurons or filters from the network.

Another one (depicted in Figure 1) is weight pruning. It identifies weights with minimal influence (mostly that with the smallest absolute values) and sets them to zero. A described method makes a sparse weight matrix that substantially decreases model size and computational demands [6].

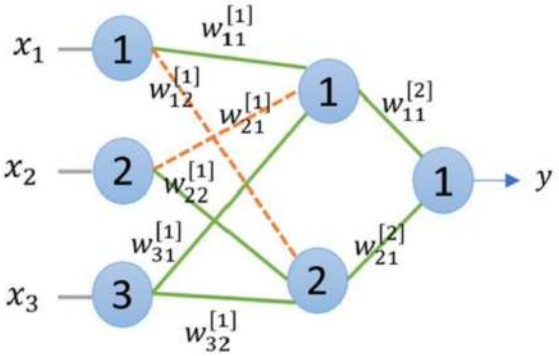


Figure 1. Weight pruning [7]

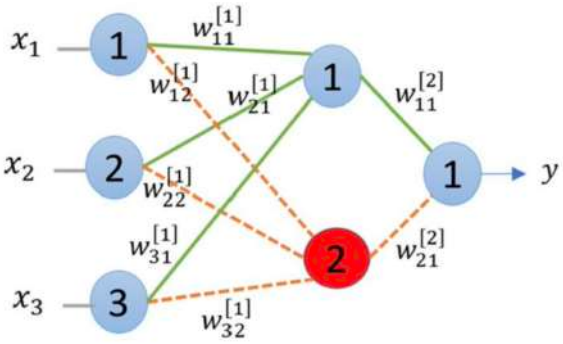


Figure 2. Neuron pruning [7]

1.1.2.2 Quantization:

Quantization is another key approach of neural network compression. It reduces the numerical precision of model weights and activations. Instead of the standart 32-bit floating point representation, parameters are converted into lower-bit formats (FP16 (16-bit floating point), INT8 (8-bit integer) or even binary values). This approach decreases the memory footprint required to store the model and optimizes the runtime performance of the model [8].

Quantization is typically performed using one of two strategies: post-training quantization and quantization-aware training (known as QAT). The first method reduces the model size and precision after full training of the model. Although such technique is extremely effective, using it may lead to a drop in accuracy. In contrast, QAT takes the effect of quantization into account during training, which helps maintain the model accuracy even with low precision weights [9].

1.1.2.3 Knowledge Distillation:

In 2015 Hinton et al. introduced a new method of model compression, called knowledge distillation [10]. The concept of this approach is to teach a smaller model (*the student*) to replicate the behaviour of a larger, more accurate model (*the teacher*). In comparison to the larger model, smaller one is not trained directly using only ground-truth labels. Instead, it is supervised using the probability distributions of the teacher model (known

as soft labels). These labels carry important information, which is a representation of teacher model's confidence about different options, as illustrated in Figure 3.

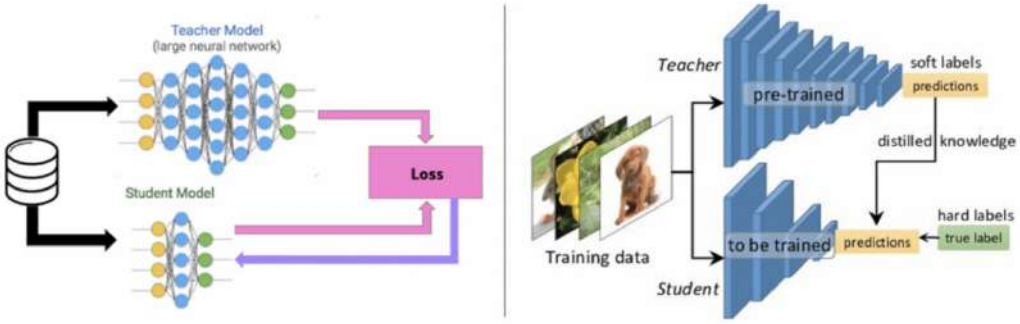


Figure 3. Knowledge Distillation [7]

The crucial goal is to retain the essential information and decision-making capabilities necessary for specific tasks by extracting the knowledge from a large-scale model into a smaller counterpart. Although a smaller model may not match the teacher's overall performance, it can still achieve similar results. [6]

### 1.1.3 Model Compression Evaluation Metrics

Accuracy is often the primary metric used when evaluating a compressed CNN. However, to fully understand the impact of compression, it is also crucial to take into account more specific factors like memory usage, inference time and proportion of parameters removed.

**Accuracy:**

Accuracy is defined as the proportion of correctly predicted examples over the total number of predictions made by the model. It is typically expressed as in (1) formula:

$$(1)\ Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} \times 100\%$$

**Top-k accuracy:**

Top-k accuracy is a metric that measures the proportion of cases in which the correct class label is included among the top k most probable predictions made by the model. Top-1 and Top-5 concepts are regularly used values, where for Top-1 correct label must be the first prediction and for Top-5 correct label must appear among the top five. This accuracy provides a broader view of a model's performance by evaluating its ability to yield reasonable guesses under high uncertainly [11].

**Pruned Ratio**

The pruned ratio (also called "sparsity") metric quantifies the degree of model compression achieved through pruning. It can be defined as the proportion of parameters removed from the original (unpruned model), which is expressed in (2) formula:

$$(2)\ \ Pruned\ Ratio = \frac{N_{original} - N_{remaining}}{N_{original}}$$

Where $N_{original}$ is the total number of parameters in the original model, and, in turn, $N_{remaining}$ is the number of parameters that remain after pruning. This metric helps to evaluate balance between compression efficiency and the preservation of model performance. [12]

## 1.2 Pruning: Types and Evolution of Approaches

### 1.2.1 Classification of Pruning Methods Structured, Unstructured and Semi-structured types of Pruning

Pruning methods can be generally classified based on the granularity at which parameters are removed. The three main categories are unstructured pruning, which deletes specific weights, structured pruning, which removes groups of parameters such as filters, channels or neurons, and semi-structured pruning, which drops weights according to a fixed pattern (for example, a few weights in each row).

#### 1.2.1.1 Unstructured pruning:

Unstructured pruning is the most finely grained method. It deletes individual weigths from the network without regard to their location in the tensor structure. Typically, this process is done by identifying weights with the smallest absolute values and setting them to zero.

Let $W = w_0, w_1, \dots w_k$ denote the set of weights in a neural network and let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be the training dataset. Given a budget $k$ for the number of non-zero weights, the unstructured pruning problem can be formulated as the following optimization task [13]:

$$(3)\ \min_{W} \mathcal{L}(W; \mathcal{D}) = \min_{W} \frac{1}{N} \sum_{i=1}^{N} \mathrm{l}\big(W; (\boldsymbol{x_i}, y_i)\big), \quad \text{subject to} \quad |W|_0 \le k$$

In practice, a binary mask $M$ is often used to indicate which weights are kept and which are pruned. This leads to an equivalent formulation (4):

$$(4)\ \min_{W,M} \mathcal{L}(W \odot M; \mathcal{D}) = \min_{W,M} \frac{1}{N} \sum_{i=1}^{N} \mathrm{l}\big(W \odot M; (\boldsymbol{x_i}, y_i)\big), \text{subject to} \quad |M|_0 \le k$$

Here, ⊙ - element- wise multiplication. The model is then fine-tuned or retrained using a fixed mask, which prevents the pruned weights from being updated.

1.2.1.2 Structured pruning:

The concept of structured pruning is based on removing entire structural components of a neural network. The given method was designed to maintain a structure that can be efficiently executed on standard hardware without the need for specialized sparse computation libraries.

Let $S = \{s_1, s_2, \dots, s_L\}$ represent the set of parameter groups in each layer (channels or filters). Structured pruning selects a smaller subset $S' \subseteq S$ that reduces model size while keeping performance degradation minimal, under a given pruning restriction[13]:

$$(5)\ S' = \{s'_1, s'_2, \dots, s'_L\}, \quad \text{where} \quad s'_i \subseteq s_i, \quad i \in \{1, \dots, L\}$$

Instead of removing individual weights, this method removes whole parameter blocks. As a result, it produces compart and hardware-friendly models.

1.2.1.3 Semi-structured Pruning:

Semi-structured pruning is a less-explored approach. It applies sparsity in a structural Pattern, sitting between unstructured and structured pruning in tersm of granularity. Since the sparsity patterns in this method are less aggressive than in structured, the negative impact on model accuracy is generally lower. However, it is important to claim that the potential of semi-structed pruning remains under-explored, especially in modern transformer-based architectures [14].

1.2.2 Main Pruning Strategies

The pruning process in neural networks involves not only selecting the structural level (the differences between these types are described in Section 1.2.1) but also choosing a defensible approach for determining the importance of individual parameters. In other words, one must decide which weights or structures are less significant for the model's prediction and can be easily removed. This brings us to pruning strategies. These methods are used to assess and rank parameters for removal. In this paper we defined three strategies, which are considered the most important in both academic research and

practical utilization: **magnitude-based pruning, gradient-based pruning**, and **regularization-based pruning**.

#### 1.2.2.1 Magnitude-based pruning

Magnitude-based pruning relies on the assumption that weights with smaller absolute values contribute less to the model's output and are considered safe to delete without a huge effect on the model's overall performance. This approach was described by Cheng et. al [13].

Formally, a binary mask $m_i$ is assigned based on the norm of each weight or weight group.

$$(6)\ m_i = 1 \text{ if } |w_i|_1 \geq a, \text{ else } 0$$

In formula (6) $a$ is a predefined threshold. Although the classical version is based only on weight magnitudes, it inspired more advanced methods. The good example is the Wanda method [15], a one-shot pruning technique designed for LLMs (large language models). The algorithm calculates the importance score, additionally using the strength of the input activation:

$$(7) \quad g_{ij} = \left|w_{ij}\right| * \left\|x_j\right\|_2$$

This makes the method slightly more accurate while still keeping it simple to use.

##### 1.2.2.1.1 L2-norm-based Structured Pruning

A commonly used structured pruning technique is based on the L2 norm of filter weights. In this approach, the importance of each filter is measured by its L2 norm, and filters with the smallest values are removed. Li et al. [16] introduced this method, which is a structured variant of magnitude-based pruning and it is typically applied after full model training.

$$(8)\ \text{Score}_{\text{L2}}(F_i) = |F_i|_2$$

1.2.2.2 Gradient-based pruning

Gradient-based pruning is a strategy that selects parameters for removal based on their estimated impact on the loss function, as derived from gradient information. Molchanov et al. [12] were first to propose a criterion that approximates the change in loss resulting from removing a feature map, using a first-order Taylor expansion.

The pruning saliency of a feature map $h_i$ is defined as in formula (9):

$$\Theta_{\mathrm{TE}}(h_i) = \left|\frac{\partial C}{\partial h_i} \cdot h_i\right| \quad (9)$$

Where $C$ is the cost function and the importance of a feuture map $h_i$ is calculated by multiplying its by value with how much it affects the loss. Parameters with lower $\Theta_{\mathrm{TE}}(h_i)$ scores are considered less important and are pruned first.

This approach is efficient because it uses gradients from backpropagation and works well across layers, especially when combined with normalization [12].

1.2.2.3 Regularization-based pruning

Regularization-based pruning introduces sparsity by modifying the loss function to penalize less relevant weights during training. The key difference that has this approach is that allowance to suppress unimportant parameters dynamically, as part of optimization process.

In [17], the authors propose an extended loss function with weight-specific regularization (10):

$$\hat{L}(\overline{w}) = L(\overline{w}) + \lambda \sum_{n,i,j} \left(I_{n,i,j} \cdot \left|w_{i,j}^{n}\right|^2\right) \quad (10)$$

Where $I_{n,i,j}$ is an irrelevance score based on gradient magnitude:

$$I_{n,i,j} = \exp\left(-\left|\frac{\partial L}{\partial w_{i,j}^{n}}\right|\right) \quad (11)$$

Weights with small gradients receive stronger penalties, which them toward zero.

### 1.2.3 Timing of Pruning: Pre-training pruning, During training pruning and Post-training pruning

The timing of pruning affects the training cost and the final model performance. Based on when the pruning process is applied, we distinguish three paradigms: pre-training pruning, during training pruning and post-training pruning [13].

**Pre-training pruning** is when pruning is applied before the training phase. The model is typically pruned at initialization and the sparse network is trained from scratch. This method is appropriate for reducing initial computational cost, but pruning effectiveness may suffer in the absence of learning signal. Often, pre-training relies on structural assumptions due to the lack of weight information at this stage.

**Pruning during training** refers to methods where the model is gradually pruned while it is being trained. Since the sparsity changes throughout training, this approach lets the network adapt to pruning on the fly. However, it introduces obstacles like a need to handle pruning and training at the same time.

In **post-training pruning**, the model is first fully trained and only then it can be pruned (optionally fine-tuned). This is the most commonly used techniques, since it saves model accuracy while simplifying the pruning process. Hence, pruning decisions can rely on actual weights and activation information, making this method powerful and effective.

| ***Strategy*** | **Pre-training phase** | **During training** | **Past-training phase** | **Explanation** |
|---|---|---|---|---|
| *Magnitude-based* | Rarely used, not effective | YES | Most common | Easiest to apply after training phase |
| *Gradient-based* | NO | YES | Possible, but less reliable | Requires access to gradients, therefore during training |
| *Regularization-based* | NO | YES | NO | Regularization acts only during training |

Table 1. Implementation phases of pruning techniques

# 1.3 Theoretical and Mathematical View of Dropout

## 1.3.1 Introduction to Dropout

### 1.3.1.1 Standard Dropout

Initially, dropout is regulation technique, which prevents overfitting in neural networks. Most precisely, the primary idea is to randomly "drop" a subset of neurons during each training iteration (the essence of the process is depicted in Figure 4). Since such technique helps prevent units from co-adapting too strongly to each other, this forces the network to learn inessential representations and improves generalization.

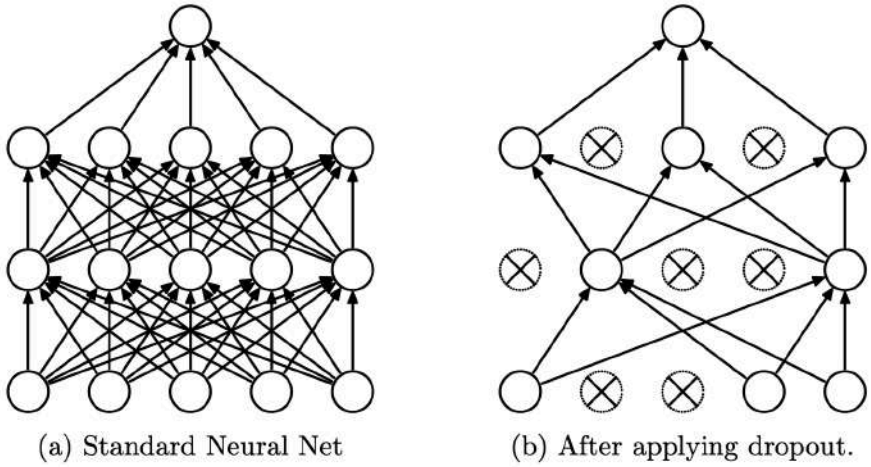


Figure 4. a) A standard neural net with 2 hidden layers,
b) A neural net after appliance of dropout [18]

From a high-level perspective, dropout can be represented as training an ensemble of many subnetworks that share parameters. Each subnetwork is formed by applying a random binary mask to the activations and during inference the full network is used with scaled weights. This approach works like averaging many models but is much faster to compute.

Srivastava et al. [18] proposed mathematical interpretation. For a given layer input $x \in R^d$ dropout applies masking using a binary vector $z$, where each element is sampled independently from a Bernoulli distribution:

$$\tilde{x} = x \odot z, \quad z_i \sim \text{Bernoulli}(p) \tag{12}$$

where p is the probability of keeping a unit active. The mask $z$ zeroes out some of the components of the input vector. During training the output is scaled as:

$$(13)\ \ output = \frac{f(W(x \odot z))}{1-p}$$

This scaling ensures that the expected output remains the same at test time when all units are active.

#### 1.3.1.2 Spatial Dropout

Spatial Dropout (also known as Dropout2d) is a variant of standard dropout made specifically for CNNs. The idea is not randomly drop individual activations, but drops entire feature maps (channels). This helps reduce the dependency between channels.[19] Mathematical formulation can be represented as:

Let $x \in R^{C \times H \times W}$ be the input tensor, where $C$ is the number of channels. Spatial Dropout is applied as follows:

$$(14)\quad \tilde{x}_{c,:,:} = x_{c,:,:} \cdot z_c, \quad z_c \sim \text{Bernoulli}(p)$$

where

- $\tilde{x}$ is the output after applying Spatial Dropout
- $z_c \in \{0, 1\}$ is a random binary variable shared across all spatial positions in channel $c$
- $p$ is the probability of keeping a channel active.

### 1.3.2 Dropout as Variational Approximation of Bayesian Inference

While dropout was originally introduced as a heuristic regularization technique to reduce overfitting in DNNs, Gal and Ghahramani [20] showed that it can be reinterpreted as a form of approximate Bayesian inference in deep models. In particular, they demonstrated that applying dropout during training is mathematically equivalent to preforming variational inference in a probabilistic model with a specific Bernoulli variational distribution over the weights.

1.3.2.1 Variational Inference Perspective:

In Bayesian neural networks the goal it to estimate the posterior distribution over weights $p(\omega \mid \mathcal{D})$, where $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$ is the training dataset. Variational inference is used to approximate it with a smaller distribution $q(\omega)$ (as computing exactly is insoluble) by minimizing the variational lower bound. This is represented in formula (12):

$$(15) \quad \mathcal{L}_{\text{ELBO}} = \sum_{n=1}^{N} E_{q(\omega)}[\log p(y_n \mid x_n, \omega)] - \text{KL}\big(q(\omega) \mid p(\omega)\big)$$

In the case of dropout itself, the variational distribution $q(\omega)$ is defined via Bernoulli-distributed multiplicative masks, which are applied to the weights or activations. For a weight matrix $W \in R^{K \times D}$, the variational form is represented in formula (16):

$$(16) \quad W = M \cdot \text{diag}(z), \quad z_j \sim \text{Bernoulli}(p)$$

Where *M* are the variational parameters and y are randomly selected binary values for each neuron.

1.3.2.2 Dropout Objective as Approximate Variational Loss:

Gal and Ghahramani [20] show that the standard dropout training objective can be viewed as a Monte Carlo estimate of the ELBO with a single stochastic sample $w^{(t)} \sim q(w)$. This leads to the practical loss function:

$$(17) \mathcal{L}_{GP-MC} \approx \frac{1}{N} \sum_{n=1}^{N} \frac{-\log p\left(y_n \mid x_n, \omega^{(n)}\right)}{\tau} + \sum_{i=1}^{L} \left( \frac{p_i \cdot l^2}{2\tau N} \|M_i\|_2^2 + \frac{l^2}{2\tau N} \|m_i\|_2^2 \right)$$

- $\boldsymbol{\tau}$ is the model prediction
- $\boldsymbol{l}$ is the prior length scale
- $\boldsymbol{M_i}$ are variational parameters
- $\boldsymbol{p_i}$ is the dropout retention probability for layer $\boldsymbol{i}$

This expression looks similar to the standard dropout loss with L2 regularization, but now it has a clear theoretical meaning as an approximate.

1.3.2.3 Predictive Uncertainly and Monte Carlo Dropout

By treating Dropout as a variational approximation, one can use Monte Carlo sampling to estimate how uncertain the model is in the predictions. Monte Carlo Dropout method involves performing multiple stochastic forward passes at test time dropout enabled and aggregating results.

To estimate the model's predictive mean for a new input $x^*$, we sample $T$ different sets of dropout masks and average the outputs [20]:

$$(18)\quad E_{q(y^*|x^*)}(y^*) \approx \frac{1}{T}\sum_{t=1}^{T} \hat{y}^*(x^*, W_1^t, \dots, W_L^t)$$

To obtain the predictive variance, we use the second raw moment of the output and subtract the squared mean:

$$(19)\quad \mathrm{Var}_{q(y^*|x^*)}(y^*) \approx \tau^{-1} I_D + \frac{1}{T}\sum_{t=1}^{T} \hat{y}^*\left(x^*, (\boldsymbol{W_1^t}, \dots, \boldsymbol{W_L^t})\right)^T \hat{y}^*(x^*, W_1^t, \dots, W_L^t)^T - E_{q(y^*|x^*)}(y^*)^T E_{q(y^*|x^*)}(y^*)$$

Where

- $\hat{y}^*(x^*, \boldsymbol{W_1^t}, \dots, \boldsymbol{W_L^t})^T$ is the prediction at the *t*-th stochastic forward pass with dropout
- $\tau^{-1}\boldsymbol{I_D}$ is a fixed value that represents the noise in the model's outputs
- the second term is the empirical second moment
- the third term subtracts the squared predictive mean

# Chapter 2. Methodology and Experimental Setup

## 2.1 Research Problem and Methods

### 2.1.1 Problem Statement

Consider a neural network with parameters $\theta$, consisting of $L$ layers:

$$(20) \qquad \theta = \{\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(L)}\}$$

Here, $\theta^{(l)}$ refers to the parameters of the $l$-th layer. When training with Dropout random binary masks are applied to the activations throughout the forward pass. For layer $l$ , we denote a binary mask by $m^{(l)} \in \{0,1\}^{d_l}$, where $d_l$ is the number of output neurons or channels in that layer.

The set of all masks generated during training can be represented as:

$$(21) \quad M = \{m_i{}^{(1)}, m_i{}^{(2)}, \dots, m_i{}^{(L)}\}_{i=1}^{k}$$

where K is the number of training iterations or distinct seeds used. [22]

**Main Idea: Dropout-based Pruning**

The core hypothesis of this work is that Dropout masks can be repurposed for pruning after training.

Instead of computing importance scores for parameters, we use a binary mask $m^{(l)}$ for each layer to guide structured pruning. Specifically, we define the pruned parameters of each layer as:

$$(22) \quad \theta_{pruned}^{(l)} = \theta^{(l)} \odot m^{(l)}$$

When the binary mask contains zeros, it removes corresponding filters or neurons.

In convolutional layers, where weights have shape $\theta^{(l)} \in R^{C_{out} \times C_{in} \times k \times k}$, a mask $m^{(l)} \in \{0,1\}^{C_{out}}$ can be applied to drop specific output channels.

This results in pruning according to the rule:

$$(23) \quad \theta_{pruned[i]}^{(l)} = \begin{cases} \theta_{[i]}^{(l)}, & if\ m_i^{(l)} = 1 \\ 0, & if\ m_i^{(l)} = 0 \end{cases}$$

What starts as a stochastic regularization step during training is later reused as a deterministic rule for pruning filters.

### 2.1.2 Dropout-based Pruning

Dropout-based pruning technique consists of three main stages:

(1) Dropout mask generation

(2) Application of the mask for structured pruning

(3) Weight rescaling

#### 2.1.2.1 Custom Dropout Implementations

We implemented two custom layers: one designed for fully connected layers and another for convolutional Layers. It helps control how dropout is operationalized in the course of training phase.

##### 2.1.2.1.1 Custom Dropout for Linear Layers

This layer randomly deactivates neurons. Unlike standard dropout, it allows setting a fixed random seed to have an ability to generate the same mask repeatedly. Moreover, It supports runtime modification of the dropout rate, keeping flexibility during training.

*Pseudocode:*

**Function: CustomDropout.forward(x, seed =None, p=None)**
**Inputs:**

| | |
|---|---|
| x | -- input tensor of shape (B – batch_size, N – number of features) |
| seed | -- optional fixed seed |
| p | -- dropout rate (default: self.default_p) |

Steps:
1. Set dropout_p = p if given, else use default_p
2. If a seed value is given:
   Set the random seed in PyTorch using torch.manual_seed(seed)
3. If model is in training mode:
   - Generate binary mask of shape (1, N) using Bernouli (1 – p)
   - Multiply input x by the mask
   - Divide the result by (1 - p) to maintain output scale
   - Return scaled result
4. Else:
Return x unchanged

2.1.2.1.2 Spatial Dropout for Convolutional Layers

This layer drops entire channels in the input by applying a binary mask across the channel dimension. It can effectively simulate spatial dropout behaviour. As with the custom dropout for linear layers, it supports setting a fixed random seed and dropout rate.

*Pseudocode:*

**Function: SpatialDropout.forward(x, seed =None, p=None)**
**Inputs:**

x -- input tensor of shape (B – batch_size, C – channels, H – Height, W – Weight)
seed -- optional fixed seed
p -- dropout rate (default: self.default_p)

Steps:
1. Set dropout_p = p if given, else use default_p
2. If a seed value is given:
   Set the random seed in PyTorch using torch.manual_seed(seed)
3. If model is in training mode:
   - Generate binary mask of shape (1, C, 1, 1) using Bernoulli (1 – p)
   - Multiply input x by the mask
   - Divide the result by (1 - p)
   - Return scaled result
4. Else:
Return x unchanged

2.1.2.2 Dropout Mask Stages

For each prunable layer in the model (such as Conv2d or Linear), we generate a binary mask by sampling from a Bernoulli distribution with the success probability $1 - p$, where $p$ is the dropout rate. The generated mask is then scaled $\frac{1}{1-p}$ due to the aim to preserve the expected output magnitude.

*Pseudocode:*

**Function: generate_dropout_mask_for_pruning(module, dropout_rate, seed = None)**

**Inputs:**

module -- A NN layer (Conv2d or Linear)
dropout_rate -- Dropout probability (float in range of [0, 1])
seed -- Seed (optional) to fix random bahaviour

**Output:**

mask -- dropout mask scaled by 1/ (1 – dropout_rate)

**Steps:**

1. If a seed value is given:
   It sets the random generator so that the same mask is created every time
2. If the module is Linear:
   - Let N be the number of output neurons
   - Generate a binary mask of shape (1, N) where each element:
     m_i = 1 with probability (1 – dropout_rate), otherwise 0
   - Scale the mask by dividing all elements by (1 – dropout_rate)
3. Else if the module is of type Conv2d
   - Let C be the number of output channels
   - Generate a binary mask of shape (1, C, 1, 1) where each element:
     m_i = 1 with probability (1 – dropout_rate), otherwise 0
   - Scale the mask by dividing all elements by (1 – dropout_rate)
4. Else:
   Raise an error indicating an unsupported layer type.
5. Return the scaled dropout mask

Once masks are generated for each layer, we can utilize them for structured pruning. The *torch-pruning* library is used to build a dependency graph. This step is quite crucial as the graph secures that pruning does not break the model architecture. Then we have to compensate the active units, since after pruning the number of them decreases. To achieve this, the remaining weights and biases are adjusted to keep the same output magnitude as in the original model.

### 2.1.3 L2-Norm Pruning

L2-norm pruning is a well-known magnitude technique that ranks convolutional filters or output neurons by their L2 norm and removes those with the lowest values. The contribution of filters with low L2 norm values to the final output is minimal, that is why they can be considered less important. L2-norm pruning is usually applied at the structured level, so rather than deleting individual weights, this method removes entire filters for convolutional layers or neurons for fully connected layers. The sparsity is matched to the dropout rate used in the dropout-based experiments. This factor supplies a fair comparison between the two strategies.

Unlike the dropout approach, the given one does not require stochastic behaviour or the use of masks, alternatively it relies directly on the trained model weights. A dependency graph is used to keep architectural consistency. After pruning the model is preferably fine-tuned to recover any performance loss.

## 2.2 System Configuration

### 2.2.1 Model Design

Convolutional, pooling and fully connected layers form the basic building blocks of CNN architectures. Typically arranged in repeating blocks, convolutional layers extract features, pooling layers reduce spatial resolution, and fully connected ones perform classification. [1]

At the early stages, convolutional filter – usually small and learnable – scan local regions of the input to extract patterns, resulting in an activation map. The ReLU function introduces non-lineority into the model, making it capable of learning beyond linear relationships.

Pooling layers, often implemented as max-pooling, downsample feature maps while preserving their depth. This reduces the number of parameters and helps the network to be invariant to small translations in the input. Finally, the extracted feature maps are flattened into a vector and passed to one or more fully connected layers. They generate predictions via softmax.

Such a structure makes CNNs effective for image classification, object detection and other machine learning tasks.

### 2.2.2 Data Description and Preprocessing

Image classification datasets consist of labeled images, where each label indicates the category of the corresponding image. These datasets are usually divided into training, validation and test sets to allow for proper evaluation of model performance.

The images are usually stored in RGB format, and their associated labels are either categorical integers or transformed into one-hot encoded vectors.

Before training image data should be preprocessed in order to standardize their format and improve the model’s ability to learn effectively. All images are resized to a fixed

resolution (for example, $64 \times 64$ or $128 \times 128$ pixels) to standardize input dimensions throughout the dataset.

To improve the robustness of the model and increase data variability, augmentation techniques are commonly utilized. The most popular one are horizontal flipping, cropping, rotation or brightness modification.

### 2.2.3 Training phase

The training process helps the model learn patterns that generalize and lead to accurate, reliable resuls. The optimization objective is to minimize the categorical *cross-entropy loss function*:

$$(24) \quad R_{\mathcal{L}}(f) = E_D[\mathcal{L}(f(x;\theta), y_x)] = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij} \log f_j(x_i;\theta) \quad [21]$$

where in formula (24):

- $R_{\mathcal{L}}(f)$ – the empirical risk of the classification function $f$
- $E_D$ – the expectation
- $f(x;\theta)$ – the model function with parameters $\theta$
- $y^{ij} \in \{0,1\}$ – the ground truth label
- $f_j(x_i;\theta)$ – the predicted softmax probability

Training is commonly performed using mini-batch optimization. So parameters are updated based on subsets of data rather than the full dataset. An adaptive *optimization method* (for instance, Adam) is suitable as it has an ability to adjust learning rates individually for each parameter.

*Learning rate scheduler* reduces the learning rate $\eta$ when the validation loss stops improving for a specified number of epochs. Mathematical representation:

$$(25) \quad \eta_{new} = \gamma * \eta_{prev}, \qquad where\ \gamma \in (0,1)$$

Another useful mechanism is *early stopping*, which checks the validation loss and stops training if no improvement is observed over a given number of epochs (usually called "patience").

# Chapter 3. Experiments and results

## 3.1 Experiment Preparation

### 3.1.1 Dataset Description and Preprocessing

The dataset used in this work is Imagenette2 [23], a simplified version of ImageNet dataset that includes only 10 selected classes. It is often chosen for benchmarking image classifiers, as it is compact and well-structured. A detailed description of the dataset format and general preprocessing step is provided in Section 2.3.

For this experimental setup, the dataset was divided into separate training and validation subsets, each representing one of the ten target categories:

| CLASS ID | CLASS NAME |
|---|---|
| **0** | Tench |
| **1** | English Springer |
| **2** | Cassette Player |
| **3** | Chain Saw |
| **4** | Church |
| **5** | French Horn |
| **6** | Garbage Truck |
| **7** | Gas Pump |
| **8** | Golf Ball |
| **9** | Parachute |

Table 2.10 Classes of Imagenette2 dataset

All images were resized to $128 \times 128$ pixels to ensure same image dimensions.

For the training set, the following data augmentation and normalization steps are applied:

- random horizontal flipping with 50% probability, which adds diversity to data and reduces the risk of overfitting
- conversion to tensor format

- normalization to the range [-1, 1], using mean and standard deviation of 0.5 for each channel

$$(26) \qquad Normalized\ pixel = \frac{pixel - 0.5}{0.5}$$

For the validation set, horizontal flipping was not applied.

### 3.1.2 Model Design

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 128, 128]           1,792
   SpatialDropout-2         [-1, 64, 128, 128]               0
            Conv2d-3          [-1, 128, 64, 64]          73,856
   SpatialDropout-4          [-1, 128, 64, 64]               0
            Conv2d-5          [-1, 256, 32, 32]         295,168
   SpatialDropout-6          [-1, 256, 32, 32]               0
            Linear-7                 [-1, 512]      33,554,944
     CustomDropout-8                 [-1, 512]               0
            Linear-9                  [-1, 10]           5,130
================================================================
Total params: 33,930,890
Trainable params: 33,930,890
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 28.01
Params size (MB): 129.44
Estimated Total Size (MB): 157.63
----------------------------------------------------------------
```

Figure 5. Model architecture

The model consists of three convolutional layers with 64, 128 and 256 filters. Each of them followed by a custom spatial dropout layer (see Section 2.1.2.1.2). Afterwards, the features are passed to a fully connected layer with 512 units and a custom dropout module (see Section 2.1.2.1.1).

The final layer maps the output to 10 classed. In general, the architecture contains around 33.9 million parameters, most of which are in the first linear layer.

## 3.2 Experiment 1: Dropout-based vs. L2-Norm Pruning

### 3.2.1 Focus and Research Questions

This experiment addresses the first two objectives outlined in the introduction:

- **Q1:** Can stochastic Dropout binary masks be utilized as a basis for structured pruning in convolutional neural networks?
- **Q2:** How does Dropout-based pruning compare to a classical L2-norm-based pruning method in terms of accuracy and sparsity level?

### 3.2.2 Experiment 1.A: Comparison of Dropout-based vs. L2-Norm Pruning (No Mask Control)

#### 3.2.2.1 Initial Training Setup

CNN model was trained on the Imagenette2 dataset with custom SpatialDropout and CustomDropout layers. The dropout rate during training was fixed to 0.5 and seed is not specified.

The model followed the given *training configuration*:

| *Parameter* | *Initial training* | *Fine-Pruning Fine-Tuning Configuration* |
|---|---|---|
| *Batch size* | 64 | 64 |
| *Number of epochs* | 20 | 25 |
| *Optimizer* | Adam | Adam |
| *Learning rate* | 0.001 | 0.0005 |
| *Early stopping* | 5 | 5 |
| *Loss function* | Categorical cross-entropy | Categorical cross-entropy |

Table 3. The training configuration for first experiment

The overall accuracy of the model is 67.69%,

### 3.2.2.2 Results for Dropout-based Pruning

After training, Dropout-based pruning was applied by generating binary masks (The detailed description of the mechanism of Dropout-based pruning is in the Section 2.1.2). We repeat the experiments with multiple dropout rates from 0.0 to 0.9 (step 0.1).

The results at each level:

| | dropout_rate | accuracy | trainable_params | sparsity_% |
|---|---|---|---|---|
| 0 | 0.0 | 67.694268 | 33930890 | 0.000000 |
| 1 | 0.1 | 67.286624 | 27519791 | 18.894580 |
| 2 | 0.2 | 65.757962 | 20708747 | 38.967864 |
| 3 | 0.3 | 62.649682 | 15048129 | 55.650650 |
| 4 | 0.4 | 56.458599 | 9751370 | 71.261084 |
| 5 | 0.5 | 53.044586 | 7125688 | 78.999407 |
| 6 | 0.6 | 47.770701 | 4998886 | 85.267448 |
| 7 | 0.7 | 40.407643 | 2362129 | 93.038411 |
| 8 | 0.8 | 19.082803 | 1016185 | 97.005133 |
| 9 | 0.9 | 16.484076 | 324981 | 99.042227 |

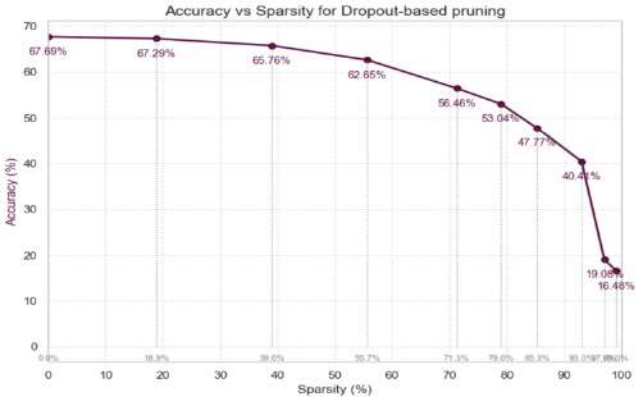


Figure 6. Results of Dropout-based Pruning with dropout rates in range of (0, 1) with step =0.1

As shown in the results (Figure 6), increasing the dropout rate leads to a steady decrease in model accuracy. For instance, when dropout rate is 0.3 (the corresponding coefficient of sparsity is approximately 56%), accuracy remains relatively similar – 62.65%. However, beyond this point the decline becomes more significant. At a sparsity rate of 0.85 accuracy fell to 47.77%, and with 0.99 the model preserved only 16.48% accuracy. This demonstrates the trade-off between model compression and predictive performance: restrained sparsity can keep acceptable accuracy, but aggressive pruning remarkably affects the model's performance.

### 3.2.2.3 Results for L2-Norm Structured Pruning

For comparison to Dropout-based pruning, we applied a self-implemented L2-norm pruning algorithm to the second model. (The detailed description of the working mechanism of L2-Norm Structured Pruning is in the Section 2.1.3).

The lowest-ranked units were removed to match the pruning ratios of the dropout-based method (for example, pruning at 10%, 20%, …. 90%):

| dropout_rate_for_pruning | accuracy | trainable_params | sparsity_% |
|---|---|---|---|
| 0.0 | 67.694268 | 33930890 | 0.000000 |
| 0.1 | 67.108280 | 27570464 | 18.745238 |
| 0.2 | 66.904459 | 21761323 | 35.865747 |
| 0.3 | 65.273885 | 16730459 | 50.692543 |
| 0.4 | 63.821656 | 12281062 | 63.805659 |
| 0.5 | 61.503185 | 8484682 | 74.994225 |
| 0.6 | 57.426752 | 5468960 | 83.882061 |
| 0.7 | 43.286624 | 3072075 | 90.946082 |
| 0.8 | 27.388535 | 1387931 | 95.909536 |
| 0.9 | 10.063694 | 350790 | 98.966163 |

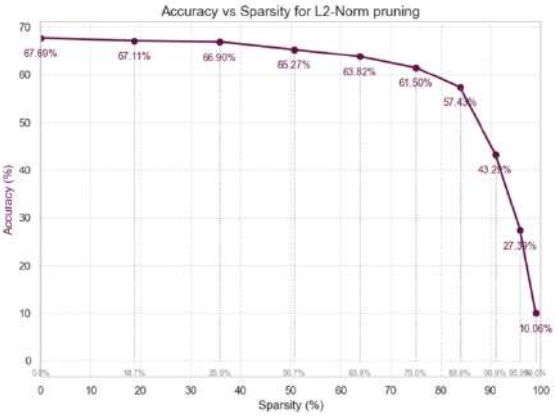


Figure 7. L2-Norm Pruning Results aligned with equivalent Dropout Sparsity Settings

According to the results presented in Figure 7, increasing the L2-based pruning ratio also gradually reduces the model's accuracy. With pruning, the network size is reduced, while keeping robust performance at moderate sparsity levels.

For example, at a dropout rate of 0.3, which corresponds to about 50.7% sparsity, the model yields good results - 65.27% accuracy. Once the dropout ratio exceeds 0.5, the accuracy begins to go down more noticeably (The accuracy drops to 61.5%). When more than 90% of parameters removed, the model's accuracy collapses to 43.29%.

### 3.2.2.4 Result Comparison

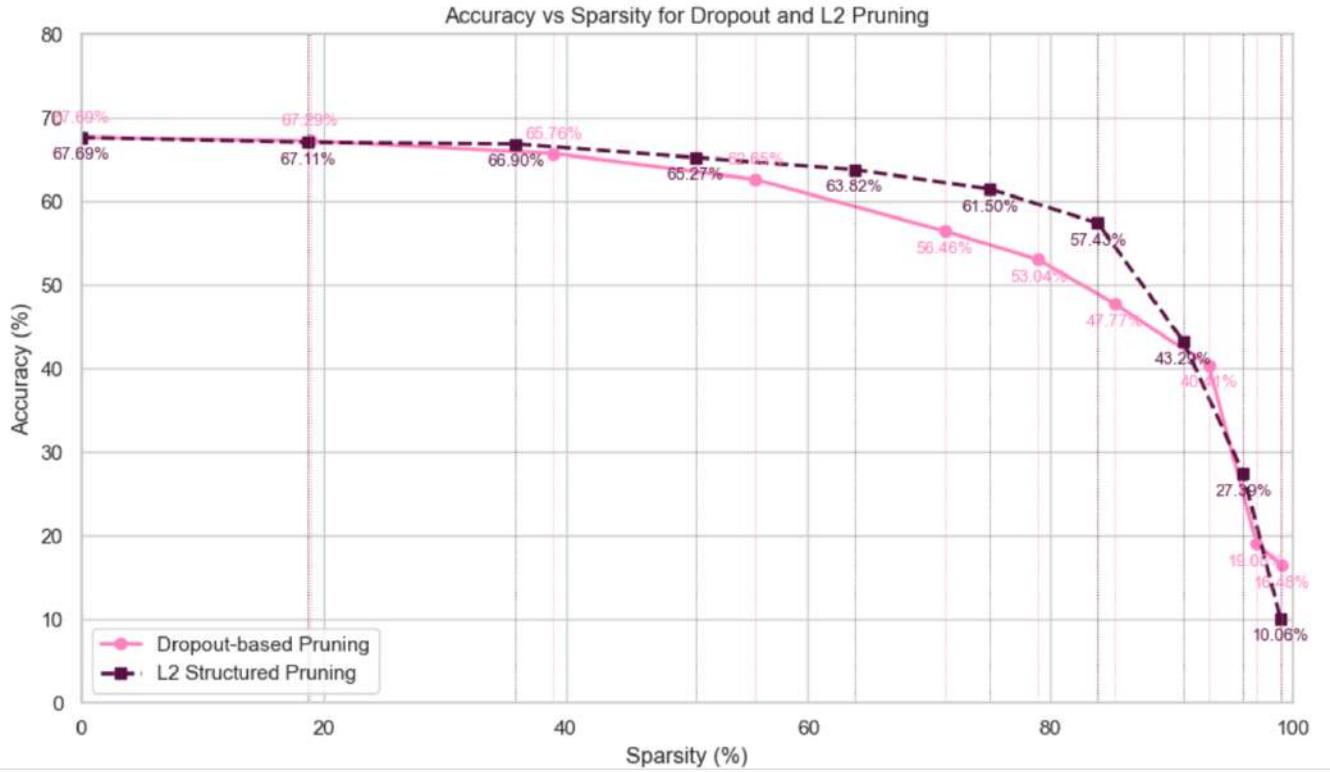


Figure 8. Dropout-based vs. L2-Norm Pruning

The results demonstrate that Dropout-based pruning can indeed be used as a pruning mechanism. The binary masks generated during Dropout training are capable of identifying redundant units and can be used for structured pruning. However, the ability to maintain accuracy depends on the degree of sparsity.

Comparing the two curves, L2-structured pruning outperforms Dropout-based pruning across most sparsity levels. The Dropout-based model experiences a sharper decrease. For example, at around 75% sparsity, the accuracy of L2 pruning is close to 61.5%, whereas Dropout pruning drops between 53 and 56 percent.

3.2.2.5 Fine-Tuning After Pruning: Enhancement of Model Performance

To further investigate the pruning bnehavior and assess whether post-pruning retraining can recover model performance, we applied fine-tuning to both Dropout-based and L2-pruned networks. In this experiment, a pruning ratio matching to a dropout rate of 0.5 was used.

The fine-tuning was fulfilled over 10 epochs using the Adam optimizer with a learning rate of $1 \times 10^{-4}$.

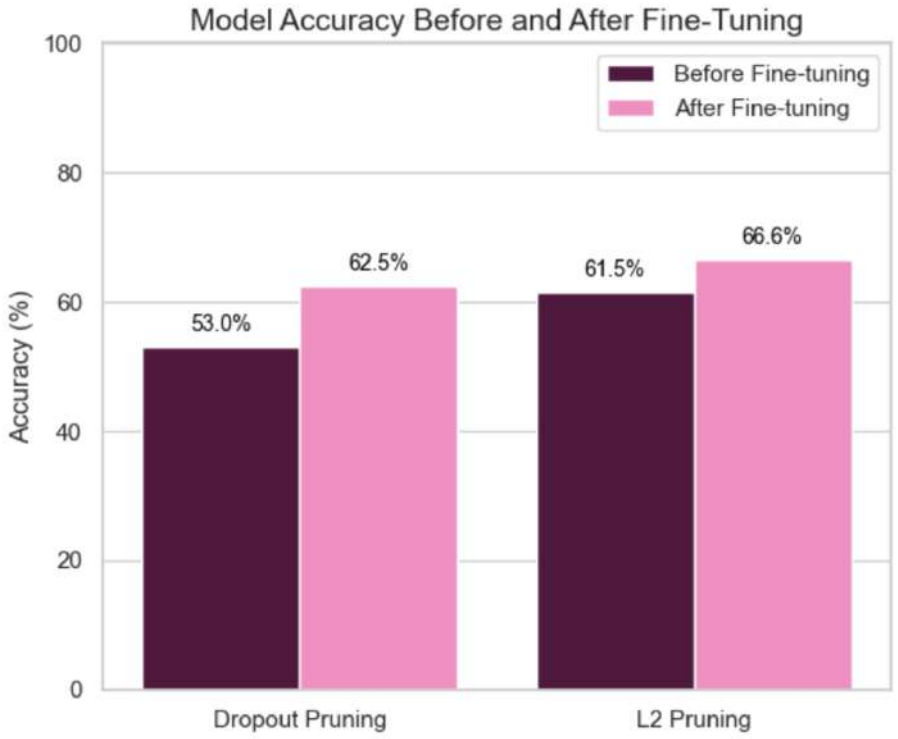


Figure 9. Fine-tuning results

For Dropout pruning the model's accuracy before fine-tuning was relatively low at 53% and after fine-tuning it improved to 62.5%, demonstrating a significant recovery of 9.5 percentage points. On the other hand, the L2-pruned model started with a higher accuracy of 61.5% and enhanced to 66.6% after retraining, yielding a smaller growth of 5.1%. This leads to the idea that while L2 pruning preserves more of the model's original performance, Dropout pruning benefits more from retraining, likely due to its more stochastic and intensive pruning approach.

### 3.2.3 Experiment 1.B: Comparison. Of Dropout-based vs. L2-Norm Pruning (With Control Mask Pool)

#### 3.2.3.1 Initial Training Setup

In this experiment, the CNN was trained using pool of 10 random seeds to simulate set of Dropout masks. The model followed the training configuration described in Table 4.

| *Parameter* | *Initial training* |
|---|---|
| *Batch size* | 64 |
| *Number of epochs* | 23 |
| *Optimizer* | Adam |
| *Learning rate* | 0.0001 |
| *Early stopping* | 5 |
| *Loss function* | Categorical cross-entropy |

Table 4. The training configuration for second experiment

During each training batch, one of these 10 seeds was randomly selected for generation of binary mask, introducing structured stochasticity while controlling the variability. This approach guarantees that the model becomes familiar to a specific subset of Dropout configurations, which is extremely crucial for later analysis that rely on consistency across mask behaviour.

The overall accuracy of the model is 63.82%,

### 3.2.3.2 Result Comparison

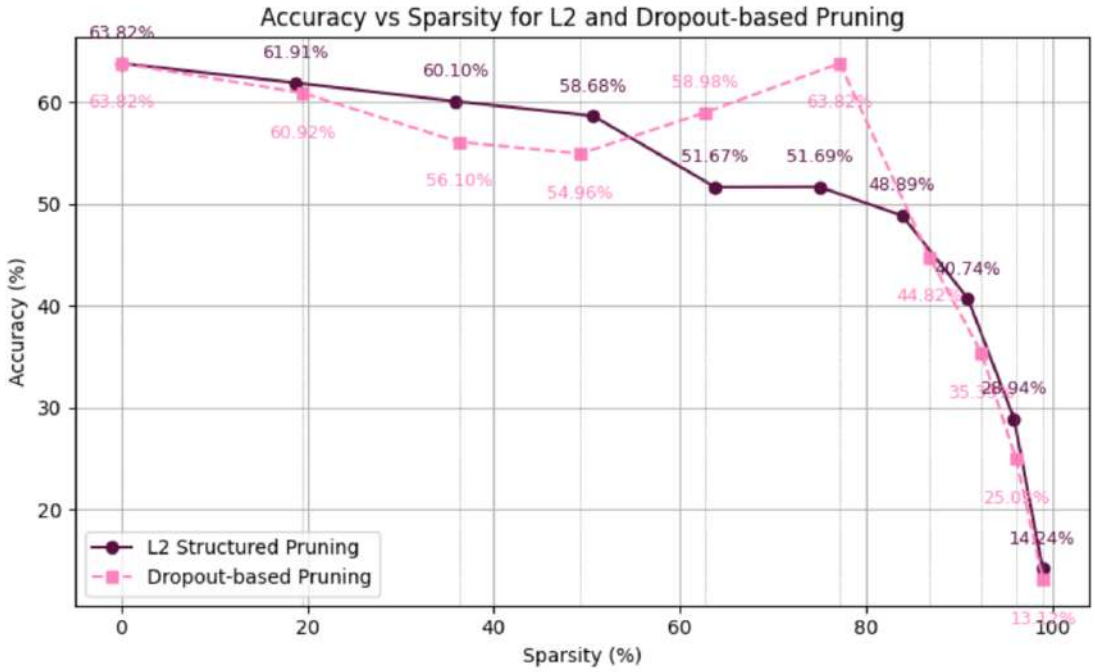


Figure 10. Results of Two Methods using a Model Trained with Fixed Dropout Seed

Observing the line chart from Figure 10, it clearly shows that the dropout-based method maintained a high accuracy above 60% even at sparsity levels exceeding 70%. This outcome shows difference in performance of dropout pruning from this experiment and the first one (see Section 3.2.3). First reason of it is that the dropout mask used for pruning was sampled from the same pool that the model was trained on. Therefore, the structure seen during training did not change during pruning. Second, using various random masks during training probably helped the model handle sparsity better.

In comparison, L2 pruning followed a more gradual decline, slightly surpassing dropout only at extreme sparsity levels – lower than 90%.

These results imply that dropout mask, which we used for pruning, are consistent with the training phase. Dropout-based pruning can complete effectively with conventional L2 Methods.

3.2.2.5 Fine-Tuning After Pruning: Enhancement of Model Performance

In this experiment, for Dropout we applied fine-tuning with two distinct settings. In the first case, the model was trained with the same dropout mask (seed = 8) that it has been used during the initial training phase. In the second case, we gave it a completely new mask (seed = 42), which the model had not utilized before. A pruning ratio matching to a dropout rate of 0.5 was used. The fine-tuning was fulfilled over 10 epochs using the Adam optimizer with a learning rate of $1 \times 10^{-4}$.

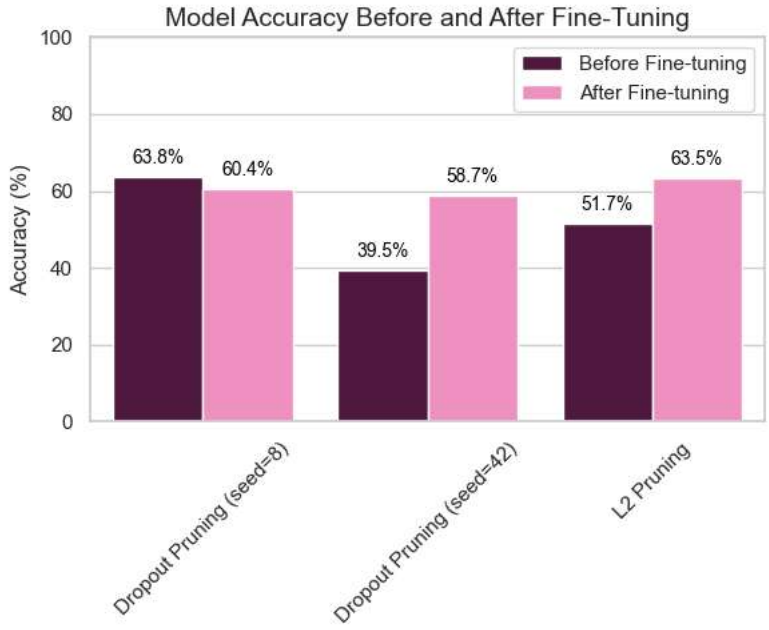


Figure 11. Validation Accuracy Before and After Fine-tuning for each pruning method

As shown in Figure 11, fine-tuning had different effects depending on the pruning setup. When the dropout mask was known to model, we can see the decrease of accuracy from 63.82% to 60.41%. It happened because of the reason that the model was already adapted to the given structure (during training) and fine-tuning turned-out to be useless in this case. In contrast, with an unfamiliar mask, accuracy improved significantly from 39.52% to 58.68%. The L2-pruned model also showed improvement, rising from 51.69% to 61.55%. This may lead us to the opinion that fine-tuning is most helpful when the model has not previously familiar with the pruning mask.

# 3.3 Experiment 2: Stability and Variability of Dropout-based Pruning

## 3.3.1 Focus and Research Question

This experiment addresses the last objective outlined in the introduction:

**Q3:** How stable and consistent are the results of Dropout-based pruning when varying the dropout rate and the random seed used for mask generation?

## 3.3.2 Full Range of Dropout Rates and Masks with Seed-controlled Training and Uncontrolled Training

The training set up for model from Figure 12 is described in Section 3.2.3.1.

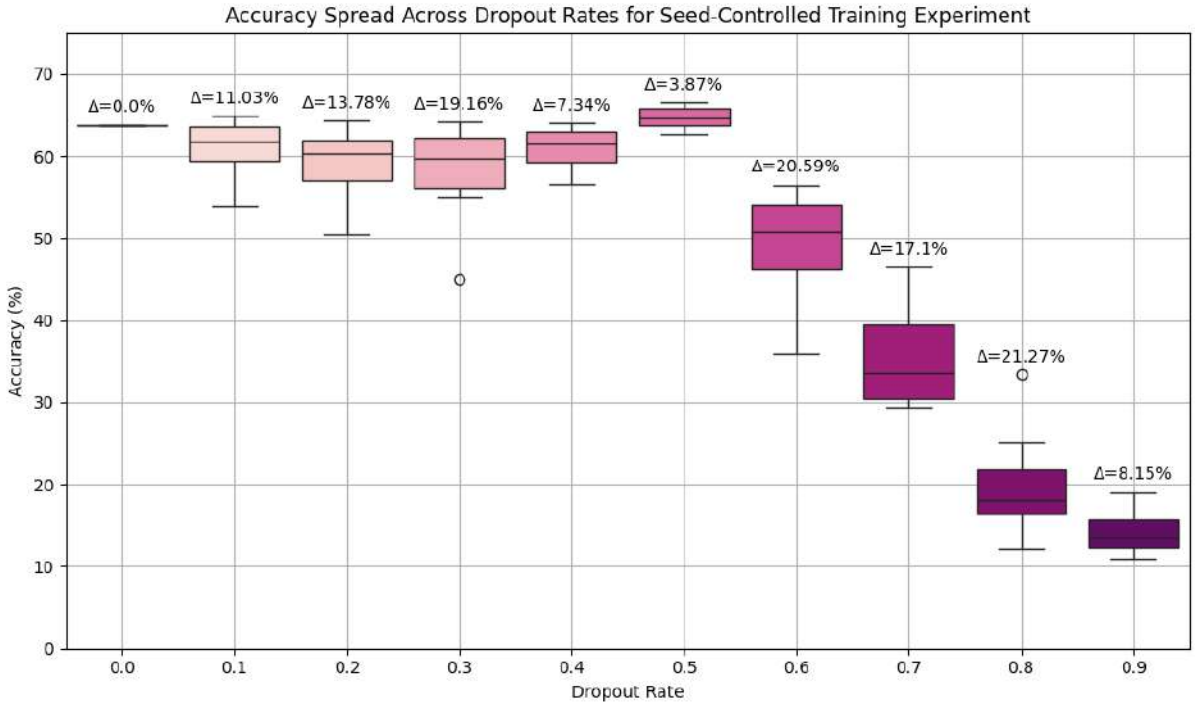


Figure 12. Box-plot for Accuracy Distribution per Dropout Rate – Model Trained with Fixed Seed Pool

The training set up for model from Figure 12 is described in Section 3.2.2.1.

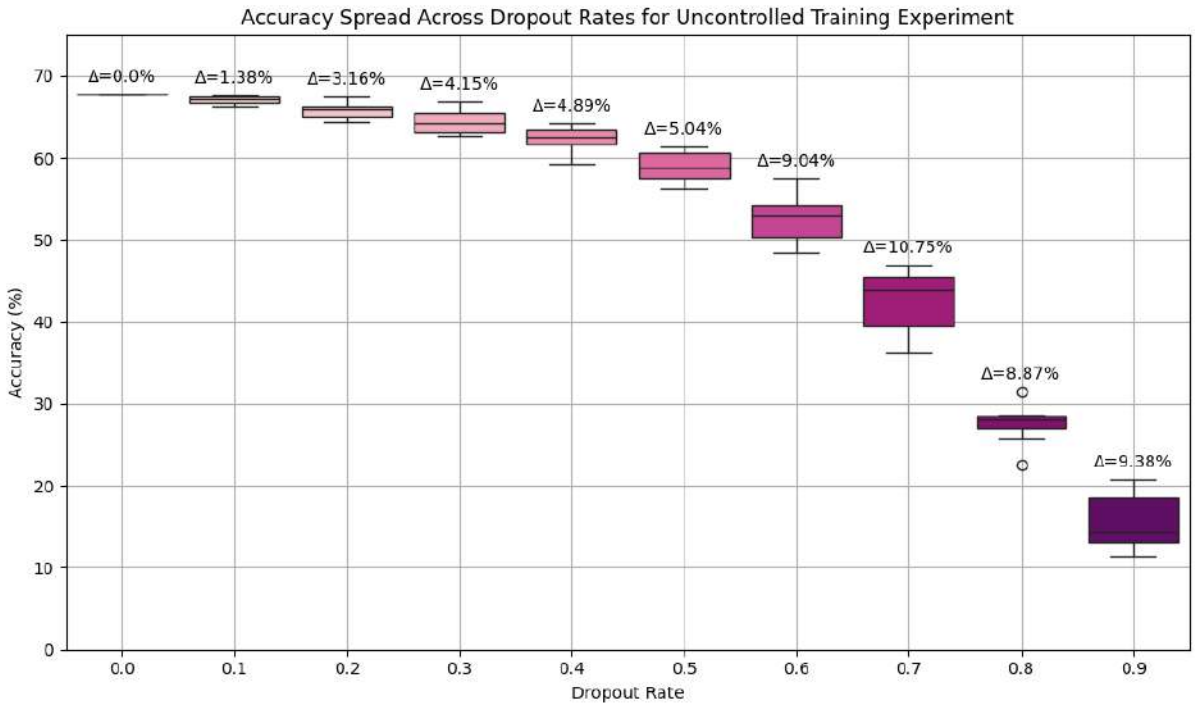


Figure 13. Box-plot for Accuracy Distribution per Dropout Rate – Model Trained without Fixed Seed

Both plots (Figure 12 and Figure 13) show how the variability and stability of the pruning results are affected by the dropout rate and training setup. In results shown in Figure 12, where we trained the model using a fixed set of seeds, are more robust to structural pruning. In particular, at dropout rate is 0.5 we observe the smallest accuracy variation (3.87%), indicating a high consistency between the structural patterns seen during training and those resulting from pruning. At the same time, for other values of dropout (even if the average accuracy rate remains high) the result variation between different seeds reaches more than 19%. This can lead to the suggestion that the model's performance under such conditions is extremely dependent on the randomly selected masks.

# Conclusion

Although the technique's effectiveness depends mostly on the level of sparsity and the mask configuration, the results of the two experiments lead us to the conclusion that Dropout-based pruning can be a functional approach to structural reduction of CNN.

The first experiment showed that L2-pruning demonstrates superior performance compared to Dropout-pruning when it comes to accuracy metrics at most sparsity levels. Through this, it still achieves suitable compression with acceptable accuracy loss. Besides, Dropout-pruning reacts more strongly to aggressive pruning, demonstrating a faster decrease in accuracy than L2. Limiting the number of Dropout masks used during training increases the efficiency of the Dropout-based pruning due to increased training time for each of the corresponding subnetworks.

Unlike L2 pruning, with Dropout multiple pruned versions of the NN can be produced. In this work, we have performed analysis of the variably of Dropout-pruning results. Further direction of work could be optimal random seed selection or Dropout-aware training.

References:

[1] Keiron O'Shea and Ryan Nash, "An Introduction to Convolutional Neural Networks", arXiv:1511.08458, arXiv, 2015

[2] James T. O'Neill, "An Survey of Neural Network Compression", arXiv:2006.03669, arXiv, 2020

[3] Quentin Fevbre, "Deep Learning Memory Usage and PyTorch Optimization Tricks", Treodo Data & AI, 2022

[4] David Patterson et al., "Carbon Emissions and Large Neural Network Training", arXiv:2104.10350, arXiv

[5] Yann Le Cun et al., "Optimal Brain Damage", NIPS papers, 1989

[6] Pierre Vilar Dantas, Waldir Sabino da Silva Jr, Lucas Carvalho Cordeiro, Celso Barbosa Calvalho, "A comprehensive review of model compression techniques in machine learning", Springer Nature, 2024

[7] Sabina Pokhrel, "4 Popular Model Compression Techniques Explained", Xailient, 2022

[8] Benoit Jacob et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference", arXiv:1712.05877, arXiv, 2017

[9] Raghuraman Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper", arXiv:1806.08342, arXiv, 2018

[10] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, "Distilling the Knowledge in a Neural Network", arXiv:1503.0253, arXiv, 2015

[11] Abanoub Ghobrial, "Evaluation Metrics for DNNs Compression", arXiv:2305.10616, arXiv, 2023

[12] Pavlo Molchanov et al., "Pruning Convolutional Neural Networks For Resource Efficient Inference", arXiv:1611.06440, arXiv, 2017

[13] Hongrong Cheng et al., "A Survey on Deep Neural Network Pruning: Taxonomy, Comparison, Analysis, and Recommendations", arXiv:2308.06767, arXiv, 2024

[14] “LSP: Low-Power Semi-Structured Pruning for Vision Transformers”, Openreview, 2024

[15] Mingjie Sun, Zhuang Liu, Anna Bair, J. Zico Kolter, “A Simple and Effective Pruning Approach for Large Language Models”, arXiv:2306.11695, arXiv, 2024

[16] Hao Li et al., “Pruning Filters for Efficient ConvNets”, arXiv:1608.08710, arXiv, 2017

[17] Giovanni Bonetta et al., “Regularization-based Pruning of Irrelevant Weights in Deep Neural Architectures”, arXiv:2204.04977, arXiv, 2022

[18] Nitish Srivastava et al., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, Journal of Machine Learning Research, 2014

[19] Jonathan Tompson et al., “Efficient Object Localization Using Convolutional Networks”, arXiv:1411.4280, arXiv, 2015

[20] Yarin Gal, Zoubin Ghahramani, “Dropout as a Bayesian Approximation: Representing Model Uncertainly in Deep Learning”, arXiv:1506.02142, arXiv, 2016

[21] Zhilu Zhang, Mert R. Sabuncu, “Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels”, NIPS papers

[22] Molchanov Dmitry et al., “Variational Dropout Sparsifies Deep Neural Networks”, arXiv:1701.05369, arXiv, 2017

[23] Imagenette2 Dataset, https://github.com/fastai/imagenette, 2022