UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Comparison of Parameter reduction methods for Change Detection in Satellite Imagery

Author:
Yana MULIARSKA

Supervisor:
Petr SIMANEK

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences and Information Technologies
Faculty of Applied Sciences

Lviv 2023

Declaration of Authorship

"The Brave are always lucky."

Ivan Bahrianyi

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Comparison of Parameter reduction methods for Change Detection in Satellite Imagery

by Yana MULIARSKA

Abstract

Change Detection is a critical problem in Computer Vision with applications in various domains such as medical detection, satellite imagery, quality control, and traffic analysis. However, existing change detection models often have many parameters, making them computationally expensive and challenging to implement in real-world applications. This study focuses on reducing the parameters set for the models designed explicitly for Change Detection in Satellite Imagery. These models typically process large-scale images, which can demand significant memory resources and take considerable time to compute. As a solution, we implement three approaches, evaluate and compare their performance on a toy CNN model and an advanced SNUNet-CD model [9], designed for the Change Detection task. The highest parameter reduction rate we achieved for SNUNet-CD is 10.4% (1.25 million parameters) with only a 3.7% model accuracy drop. The experiments demonstrate that, when utilizing our methods, SNUNet-CD outperforms several SOTA models in the change detection domain. We succeeded in surpassing UNet++_MSOF [22] with respect to parameter count, while the original SNUNet-CD with 32 channels was unable to do so.

The code implementation of this work is available on GitHub: `https://github.com/muliarska/parameter-reduction-for-change-detection/`.

Acknowledgements

I express my gratitude to the Armed Forces of Ukraine, the volunteers, and all those who help Ukraine. Your support has made this achievement possible.

I want to thank my supervisor, Petr Simanek, from the Data Science Laboratory at the Czech Technical University in Prague. His guidance, assistance, and recommendations were indispensable in making this work a valuable contribution to the academic community.

I am also deeply grateful to my alma mater, the Ukrainian Catholic University, for providing me with a solid academic foundation and an exceptional social environment. I am grateful to all the students, professors, faculty members, and university staff for their hard work and dedication to the community. My time at the university not only taught me about computer science but also valuable life lessons. Here I found lifelong friends who are very close and important people to me now.

Additionally, I wish to acknowledge the University of Notre Dame, where I participated in an exchange program during my final semester of undergraduate studies. Almost all parts of my thesis were written while I was at Notre Dame. I appreciate the university's support to all Ukrainian and Ukrainian Catholic University students.

Finally, I express my heartfelt gratitude to my family and friends for their support through both joyful and challenging times. I am grateful to my sister, Khrystyna, my mother, Nadiia, and my father, Oleh, for their love and encouragement despite the distance.

Contents

List of Figures

List of Tables

List of Abbreviations

CNN	**C**onvolutional **N**eural **N**etwork
CVA	**C**hange **V**ector **A**nalysis
PCA	**P**rincipal **C**omponent **A**nalysis
SOTA	**S**tate **O**f **T**he **A**rt
SVD	**S**ingular **V**alue **D**ecomposition

Dedicated to my mother, Nadiia, an extraordinary woman whose love and resilience continue to inspire me.

Chapter 1

Introduction

1.1 Motivation

Deep Learning models are notable for their vast number of parameters. However, although it helps to capture many data dependencies, it also creates several challenges in working with the networks. Training such models is computationally and memory expensive. Usually, the process is slow if the user does not own significant training capabilities. Moreover, it can lead to overfitting, which does not encourage the model to learn the generalized patterns but pushes it to simply memorize the training data.

The main challenge in solving the heavy parametrization problem is finding a trade-off between producing a smaller, more efficient model and remaining relatively accurate. In [13], the authors propose SqueezeNet, a small CNN, that handles to maintain the AlexNet metrics results and, simultaneously, reduces the number of the parameters 50 times, as well as decreases the memory usage 510 times. Changing the network architecture requires extensive experimentation and testing, which can be computationally expensive. With that reasoning in mind, we seek to speed up the model by explicitly changing the parameters set without modifying the model architecture.

Assuming which models can benefit from the reduction in parameter number, we picked the Deep learning models designed to solve a change detection problem. This task is a critical problem in Computer vision spotted in many domains, such as medical images and financial studies. We focus specifically on Change detection in satellite imagery. The reasoning is that operating with satellite imagery means processing large-scale images. Because satellite images cover extensive geographic areas, they usually require a high spatial resolution, leading to high memory consumption and performance slow-down. One such data source is the public xBD dataset [10], the most prominent building damage assessment dataset at the moment of its release, covering more than 45.000 km^2 of imagery. Moreover, the models designed to solve the Change detection in satellite imagery problem usually have a vast number of parameters.

There are multiple ways to approach the change detection task, among which one of the most popular is U-Net-based models [22, 15, 23]. Another potential answer is to employ Siamese Networks [3, 5] as they are designed to take two images as input and detect the differences between them. To understand the dependence between the number of parameters and quantitative performance metrics of the recent change detection models, see Table 1.1. Those results lead us to the assumption that change detection models are required to grow in their parameter set size in order to increase and maintain good accuracy. However, this process brings several disadvantages, such as performance slow-down or demand for high computational and memory powers. It introduces the main focus of this research which is to reduce the number of parameters while maintaining relatively accurate.

Model / Channels #	Params (M)	F1 score
IFN [33]	35.72	0.903
DASNet [3]	16.25	0.919
UNet++_MSOF [22] / 32	11.00	0.876
FC-Siam-diff [5] / 32	5.39	0.692
FC-Siam-conc [5] / 16	1.54	0.637
FC-Siam-diff [5] / 16	1.35	0.652

TABLE 1.1: Comparison of the number of parameters and F1 score for change detection models. The scores are obtained by the authors of [9].

1.2 Hypotheses

The leading job of this thesis is to speed up the model designed for the Change Detection in Satellite Imagery task by applying the parameters reduction techniques. This section summarizes and outlines the hypotheses we aim to test to achieve this goal.

1) The models designed for Change Detection in Satellite Imagery benefit from parameter reduction methods due to model complexity and operation with large-scale imagery.

2) The Fractional Filter, Pruning, and Low-Rank Approximation approaches successfully reduce the number of parameters for the change detection models as they showed the potential in the related papers.

3) We are able to integrate those approaches into the models' convolutional layers.

4) The accuracy of the model may decrease while reducing the number of parameters.

5) Fractional Filter has better scalability compared to other parameter reduction methods. It reduces the number of parameters N from $k \times k$ (where k is the kernel size) to only 6 regardless of the value of k.

1.3 Thesis Structure

This work consists of six chapters, starting with the **chapter 1** that introduces the domain and our hypotheses. Next, we briefly discuss the main ideas of each next chapter for easier navigation through the paper.

Chapter 2 contains the related literature overview: methods that could be used to speed up a Neural Network and the existing methodology for solving a Change detection in satellite imagery problem. It contains both traditional approaches and SOTA models. In **chapter 3**, there is a mathematical background on fractional calculus, which is a ground for convolutional filter approximation with fractional derivatives. **Chapter 4** introduces the proposed solution: the algorithms for the parameter reduction methods. In **chapter 5**, there is a description of experiment settings, including dataset structure and architectures of the models. Moreover, we discuss the results of the experiments and derive conclusions from them. Finally, **chapter 6** summarises the work done and proposes several ideas for future work.

Chapter 2

Related Work

2.1 Methods for Improving Model Efficiency

This section considers several techniques to improve the Deep Learning model's performance. Note that most of the methods described in this section do not conflict with each other. For example, one can apply Pruning and Fractional Filter on a single convolutional layer if the project goal requires a higher parameter reduction rate.

2.1.1 Switching the Kernel Size

The usage of smaller filters is the most direct method to approach a heavy parameterization problem. For example, replace a 5x5 filter with a 3x3 filter. [27] examines the impact of kernel size on mobile networks by testing out different options. However, this logic does not work well for more extensive and complex models. Smaller filters cannot capture intricate data patterns, leading to significant performance reduction. Therefore, we consider more complex approaches while seeking a solution to this problem.

2.1.2 Modifying Model's Architecture

Simplifying the model's architecture can reduce the memory and computational resource requirements to train the model efficiently. One can question if all Neural Network layers are crucial for its well-performance, as the authors of [25] have done. They simplify the model by using convolutional layers only, without big performance sacrifice.

The Inception method described in [26] is the traditional approach that enforces the architecture change. The authors suggest applying an additional 1x1 convolutional filter before applying more extensive 3x3 or 5x5 convolutions. This approach is also known as a bottleneck application. It leads to dimensionality reduction of the convolutional filter input, meaning the reduction in parameters number. A more complex example of modifying the bottleneck structure is described in [4]. The authors create a new bottleneck structure that shows about 25% reduction in parameters for a mobile network. Traditional bottleneck uses 1x1 convolution before and after 3x3 convolution to reduce the number of parameters, while this approach employs an inverted residual structure as a solution.

2.1.3 Weight Matrix Approximation

A common way to approximate the weights matrix is to set some subsets to zero. Usually, we set to zero small weight values as this transformation does not lead to a radical change in the output. One of the standard techniques in this domain is pruning. The magnitude-based pruning selects the weights to be pruned at the beginning by considering their absolute value. At the same time, iterative pruning redefines pruned weights on each iteration. In [11], the authors suggest a more "aggressive" approach based on those two pruning types. They achieved a weights matrix compression rate of up to 13 times.

Another possibility is not to apply the changes on top of the existing weight matrix but to factorize it into smaller matrices and operate with them. Knowledge of the SVD allows one to transform a complex matrix M into three smaller matrices U, E, and V, where $M = USV^T$. Then, we can obtain the approximation of matrix M by keeping only larger than some threshold values in the diagonal matrix E. Those simple mathematical calculations present the base for the low-rank approximation methods, which modern researchers constantly improve. The approach introduced in [1] achieved a storage compression ratio of up to 9.71. At the same time, in the [14], we witness the storage compression ratio equal to 22.47 on LeNet5 for MNIST. In practice, it is difficult to reach such promising results, as the higher the compression ratio we obtain, the bigger the accuracy reduction arises. Therefore, users need to find a trade-off between compression and good metrics maintenance.

Consider the case when the weight matrix is a sparse matrix, meaning most of the matrix elements are zero. [29] introduces an approach based on low-rank approximation, except it uses Product-of-Sums (PoS) decomposition instead of traditional SVD or PCA. The authors reached the compression in parameters number N for MNIST CNN from $N = 3 * 10^6$ to $N = 3554$ trainable parameters without significant accuracy sacrifice.

2.1.4 Modifying the Filter Setup

Note that the methods described above approximate or modify the weights matrix directly. Another way to approach the heavy parameterization problem is to approximate the weights matrix and the filter by creating a new parameters setup. The [31] suggests defining a convolutional filter using fractional calculus. The paper achieves the parameters set compression 5 times for kernel size $k = 5$.

2.2 Change Detection Approaches

2.2.1 Traditional Methods

Starting with traditional approaches, we have to mention CVA [30] and PCA [6]. Those fundamental decomposition methods consider the difference between images as the difference among a pair of pixels and then make a change/non-change decision using some threshold. As an extension of this approach, one can perform a PCA and apply k-means on top to group similar pixels as described in [2]. The traditional methods cannot sufficiently capture small noisy changes as the SOTA methods. Moreover, the images in change detection datasets are usually captured in various lighting and during different seasons, which is had to catch for the traditional methods.

2.2.2 U-Net and Siamese Networks

Alternatively to the traditional methods, one can employ a network based on the U-Net architecture [22, 15, 23] to approach the Change detection in remote imagery problem. The authors of [15] report that the highest accuracy value they achieved equals 89.2%.

Another possibility is using the Siamese Networks to solve the Change detection in remote imagery problem. The Building Damage Assessment Model developed by Microsoft [19] is one of the SOTA approaches in this domain. It has around 0.69 million parameters requiring significant computational and memory resources to train a model. This method uses the public xBD dataset [10] that consists of large-scale satellite imagery, building polygons, and labels that denote damage levels, particularly for the classification task. However, to train the Building Damage Assessment Model on the xBD dataset, one has to first calculate the change masks for each image pair. The change mask generation process is computationally expensive due to the significant size of the dataset. It causes a requirement for a significant amount of storage and computational resources to use the current model.

The SNUNet-CD model [9] can be applied to solve the change detection task. It involves the usage of both Siamese Networks and U-Net (NestedUNet in this case) architectures, as well as the previous Building Damage Assessment Model. The DASNet model [3] is another recent model in the change detection domain that achieved relatively competitive results. It has around 16.25 million parameters demanding significant computational and memory resources to train a model. While evaluating the performance of the DASNet model on the CDD dataset [17], the authors reached F1 score equal to 0.919. However, for comparison, consider the FC-Siam-diff [5] model, which has much fewer parameters (around 5.39 million) and achieves an F1 score lower than 0.7. Another well-performing approach in the change detection domain is IFN [33], with the number of parameters equal to 35.72 million. Previously introduced Table 1.1 summarizes the metrics described above by comparing the parameters number and F1 score for the models in the change detection domain.

Chapter 3

Fractional Calculus

3.1 Motivation for Applying Fractional Calculus

Fractional calculus is the part of mathematics that operates with fractional-order derivatives and integrals. Its history started with the question L'Hospital asked Leibnitz, which raised the possibility of expanding the derivative theory to the idea of derivative order n to be any number (irrational, fractional, or complex). "What if n be 1/2?" he asked. To which Leibnitz replied: "From this apparent paradox, one-day useful consequences will be drawn" [24].

Some standard fractional derivatives definitions are Riemann-Liouville and Caputo [20]. Fractional-order derivatives are more complex than integer-order ones, requiring calculating a sum over a range of fractional powers. It requires more computational resources, as well as higher execution time. Nevertheless, in recent years its usage has been found beneficial in many fields, for example, in quantum mechanics [18].

We can discover several successful cases by considering the fractional calculus application in Deep Learning. In [32], the authors introduce a fractional calculus methodology that enhances the activation function selection. They outperform the ResNet100 metrics by training a ResNet18 with fractional derivative order as an additional hyperparameter. Another example of fractional calculus application in Deep Learning is convolutional filter approximation, described in [31]. The authors report the parameter compression five times per kernel without significant accuracy sacrifice.

3.2 Theoretical Background

This section introduces the fractional derivative concept and its theoretical background.

For the beginning, consider the integer-order derivative of the power function $f(x) = x^n$. We can calculate the first derivative of y: $f'(x) = nx^{n-1}$, the second derivative: $f''(x) = n(n-1)x^{n-2}$, and others by the same logic. Equation 3.1 generalizes the calculation of the derivative of order $m \in \mathbf{N}$ for $f(x) = x^n$ function.

$$D^m f(x) = \frac{d^m f(x)}{dx^m} = \frac{n!}{(n-m)!}x^{n-m}. \tag{3.1}$$

Afterward, knowing that $a!$ works only for $a \in \mathbf{N}$, we use the generalized Gamma function Γ, defined in Equation 3.2, to replace $a!$ in Equation 3.1.

$$\Gamma(a) = (a-1)!; a \in \mathbf{N} \tag{3.2}$$

As a result, we get the final fractional derivative formula (see Equation 3.3) valid for $x \geq 0$.

$$D^m f(x) = \frac{d^m f(x)}{dx^m} = \frac{\Gamma(n+1)}{\Gamma(n-m+1)} x^{n-m}. \tag{3.3}$$

For example, consider the fractional derivative with the order $m = 0.5$ for the function $y = x^3$, meaning $n = 3$. We use Equation 3.3 to calculate it and receive Equation 3.4 as a result.

$$D^{\frac{1}{2}} f(x) = \frac{d^{\frac{1}{2}} f(x)}{dx^{\frac{1}{2}}} = \frac{\Gamma(4)}{\Gamma\left(\frac{7}{2}\right)} x^{\frac{5}{2}}. \tag{3.4}$$

The Γ of an integer is calculated as described in Equation 3.2: $\Gamma(4) = 3! = 6$. To find Γ for the fractional value, we use the property $\Gamma(a+1) = a \cdot \Gamma(a)$ and the knowledge that $\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$. Then, $\Gamma\left(\frac{7}{2}\right) = \frac{5}{2} \cdot \frac{3}{2} \cdot \frac{1}{2} \cdot \sqrt{\pi}$.

To see how the derivatives behave graphically, we visualized the derivatives of different orders, both integer and fractional, for the function $f(x) = x^3$ (see Figure 3.1). We pointed out the function $f(x) = x^3$ by the dashed line. Then we included includes integer order derivatives and fractional order ones that cover the following cases for the derivative order m:

1) $m < 1$.
2) $m > 1$.
3) $m < 0$.

Those plots prove that if the derivative order is positive, the function decreases at that point. Likewise, if the order is negative, then the function is increasing. Moreover, the lower the derivative order we provide, the lower reduction in the function we can witness.

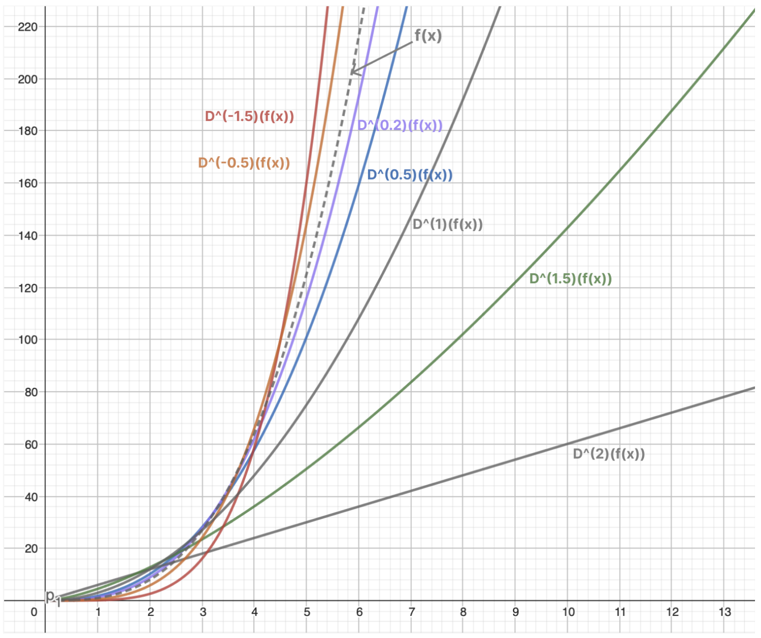


FIGURE 3.1: Visualization of the derivatives of different orders for the function $f(x) = x^3$. Plots are generated by definition, described in Equation 3.3.

Chapter 4

Proposed Solution

4.1 Motivation for Using Parameter Reduction Methods

This study aims to speed up the models designed for Change Detection in Satellite Imagery tasks. To achieve this, we decided to modify model parameters rather than model architecture. Changing the architecture of a network requires extensive experimentation and testing, which can be computationally expensive. In contrast, modifying the parameter set is a simpler and quicker approach, requiring fewer resources and time. Specifically, we utilize various parameter reduction approaches, including Fractional Filters, Pruning, and Low-Rank Approximation. Each method modifies the parameters differently (Fractional Filter: redefining the parameter set; Pruning: sets some values of the weights matrix to zero; Low-Rank: approximates the matrix with the reduced-rank one), allowing us to compare their effectiveness. Moreover, those methods can be integrated into the convolutional layer implementation, which creates a potential for its successful performance on the change detection model that is based on convolutions. In this chapter, we dive into each method algorithm we implement within this work.

4.2 Fractional Filters

Consider some traditional filters used in the convolutional layer to extract features from the data: Gaussian, Sobel, and Laplacian filters. The Gaussian filter is defined by Equation 4.1. Then, the Sobel filter approximates the first derivative of the Gaussian when the Laplacian filter does the second derivative of the Gaussian filter.

$$G(x) = e^{-\frac{(x-x_0)^2}{\sigma^2}}. \tag{4.1}$$

This derivatives-based logic leads the authors of [31] to the introduction of the Fractional filter, which operates with fractional derivatives instead of the traditional integer ones.

To compute the Fractional filter, we need to calculate the values of the weight matrix W of the shape $k \times k$, where k is a kernel size. Equation 4.2 calculates the $W(i,j)$ component of W as the 2-dimensional derivative of the Gaussian function, defined in Equation 4.1.

$$W(i,j) = D^a D^b G(i,j), \tag{4.2}$$

where $1 <= i <= k, 1 <= j <= k$, and $a, b \in (0,2)$ are the fractional derivatives orders.

Next, the two-dimensional fractional derivative $D^a D^b G(x,y)$ can be decomposed as the exterior product of one-dimensional derivatives as follows.

$$D^a D^b G(x,y) = D^a G(x) \times D^b G(y). \tag{4.3}$$

To calculate the fractional derivative of $D^a G(x)$ in practice, the authors of the Fractional Filter approach use the Taylor series and calculate only the first 15 terms for computational simplicity (see Equation 4.4).

$$D^a G(x) = \frac{A}{h^a} \sum_{n=0}^{15} \frac{\Gamma(a+1)G(x)}{(-1)^n \Gamma(n+1)\Gamma(1-n+a)} \tag{4.4}$$

where $A \in (-\infty, \infty)$, $\sigma \in (0, \infty)$, $x_o \in (-\infty, \infty)$ are the parameters that define the filter; $a \in (0, 2)$ is the fractional derivatives order, as mentioned before; and constant h, which is usually set to $1/k$.

By using (b, y, y_o) instead of (a, x, x_o) in Equation 4.4, we can compose the formula for $D^b G(y)$.

To summarize the procedure of computing the Fractional filter weights matrix step by step, we present the algorithm 1, which we use in our implementation.

Algorithm 1 Fractional filter weights calculation

```
 1: function COMPUTE WEIGHTS MATRIX($a$, $b$, $A$, $\sigma$, $x_o$, $y_o$, $k$)
 2:     for $x \leftarrow 0$ to $k$ do
 3:         $G(x) \leftarrow e^{-\frac{(x-x_o)^2}{\sigma^2}}$
 4:         $D^a G(x) \leftarrow \frac{A}{h^a} \sum_{n=0}^{15} \frac{\Gamma(a+1)G(x)}{(-1)^n \Gamma(n+1)\Gamma(1-n+a)}$
 5:     end for
 6:     for $y \leftarrow 0$ to $k$ do
 7:         $G(y) \leftarrow e^{-\frac{(y-y_o)^2}{\sigma^2}}$
 8:         $D^b G(y) \leftarrow \frac{A}{h^b} \sum_{n=0}^{15} \frac{\Gamma(b+1)G(y)}{(-1)^n \Gamma(n+1)\Gamma(1-n+b)}$
 9:     end for
10:     $W \leftarrow D^a G(x) \times D^b G(y)$
11:     return $W$
12: end function
```

4.3 Pruning

Another approach we employ to speed up the model is pruning. It sets some of the parameters to zero based on different conditions. Our method decides what weights to prune by calculating the L2 norm, also known as Euclidean norm expressed by Equation 4.5, of the weights matrix by the 0th axis.

$$\|W\| = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (w_{ij})^2}. \tag{4.5}$$

To perform pruning in our implementation, we use *prune.random_unstructured* method developed by PyTorch [21]. The hyper-parameter *amount* influences how many parameters the pruning method zeros out. It is an open discussion of choosing the most reasonable value of the parameter as it depends on accuracy and model speed requirements. As the default option, we use *amount* = 0.3, meaning the method prunes 30% of all weights.

4.4 Low-Rank Approximation

4.4.1 Factorization using SVD

For the primary implementation of our last approach, Low-Rank Approximation, we use SVD to factorize the weights matrix W into a combination of three smaller ones: U, S, V. Afterward, we trim by the rank k and decompose those three matrices to get the resulting approximation of the input matrix: $W' = U'S'V'^T$.

4.4.2 Factorization using CUR Decomposition

The baseline SVD approach introduces the problem of selecting the optimal rank value k. The [1] proposes to use the CUR decomposition to cope with this challenge. We employ this approach for the Low-Rank Approximation in this work.

To approximate the weights with CUR decomposition, we need to compute two sub-matrices: C and R. The first step in calculating C is to perform SVD on W_{2d}, a 2d representation of the 4d weights matrix W.

$$U, S, V^T = SVD(W_{2d}). \tag{4.6}$$

Then, we use matrix V to find the normalized statistical leverage scores with the leading r singular vectors, as described in Equation 4.7.

$$\pi_j = \frac{1}{r} \sum_{\xi=1}^{r} \left(\mathbf{v}_j^{\xi} \right)^2, \tag{4.7}$$

where r is the rank of 2d weights matrix W_{2d}. Note, $rank(W_{2d}) = rank(W)$.

Afterward, to obtain the final C matrix, we choose the columns of W_{2d} considering the probabilities expressed by Equation 4.8.

$$p_j = min(1, c\pi_j). \tag{4.8}$$

It appears to be a challenge to thoughtfully choose the c parameter, which can also be fine-tuned. In the article introducing the CUR decomposition [8], the authors state that the value of c depends on the application. We consider three options:

1) A constant;

2) A logarithmic in the size of n: $O(log(n))$;

3) $O\left(\frac{r \log r}{\varepsilon^2}\right)$, which is used in [1].

To calculate the matrix R, the Equations 4.6, 4.7, 4.8 should be applied to the transpose of W_{2d} matrix instead of regular W_{2d}.

One of the last steps would be to calculate the matrix U by $U = C^+WR^+$, where C^+ and R^+ are the Moore–Penrose generalized inverse of C and R, respectively. Then, to get the final approximation of the weights matrix, we compute $\Theta = CUR$ and reshape Θ back to the 4d representation.

We summarize the process of approximating the weights matrix using CUR decomposition in the algorithm 2.

Algorithm 2 Low-Rank Weights Approximation using CUR Decomposition

```
1: function COMPUTE SUB MATRIX($M$)
2:     $U, S, V^T \leftarrow SVD(M)$
3:     $\pi_j \leftarrow \frac{1}{r} \sum_{\xi=1}^{r} \left(\mathbf{V}_j^{\xi}\right)^2$
4:     Compute $S$ by selecting the columns using probabilities $p_j = min(1, cpi_j)$
5:     return $S$
6: end function
7:
8: function APPROXIMATE WEIGHTS($W, k$)
9:     Reshape W into 2d matrix: $W_{2d} \leftarrow W$
10:    $C \leftarrow$ Compute Sub Matrix($W_{2d}$)
11:    $R \leftarrow$ Compute Sub Matrix($W_{2d}^T$)
12:    $U \leftarrow C^+WR^+$
13:    $\Theta \leftarrow CUR$
14:    return $\Theta$
15: end function
```

Chapter 5

Experiments and Results

5.1 Experimental Setup

In this section, we aim to measure the performance of the parameter reduction methods on two models: CNN trained on the MNIST dataset and SNUNet-CD.

5.1.1 CNN on MNIST Dataset

Firstly, we developed a simple CNN consisting of two convolutional layers combined with some max pooling and ReLU layers. Figure 5.1 shows its backbone's visualization. We performed the experiments with this model on the MNIST dataset [7].

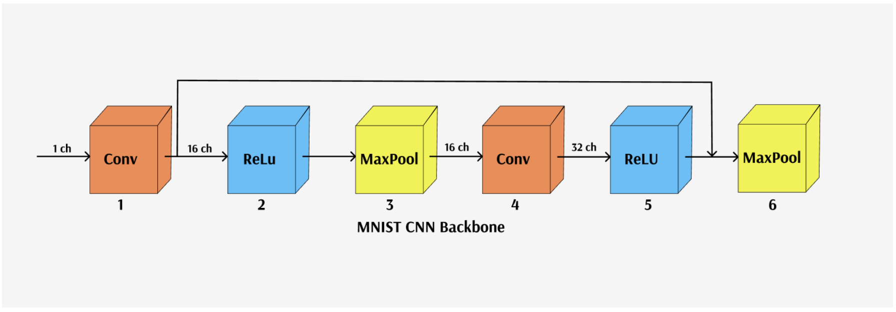


FIGURE 5.1: The CNN Architecture used to perform experiments on MNIST dataset.

5.1.2 SNUNet-CD Change Detection Model

It is worth mentioning that we first tried Building Damage Assessment Model developed by Microsoft [19]. The experiment did not go successfully due to space and computational resources limitation. However, we researched several other Siamese networks in the Change Detection domain. Figure 1.1 compares the parameters number and F1 score among those models. As a result, we stopped on the SNUNet-CD proposed in the [9] as a central model for our experiments. The reasoning behind our decision is the computational load available for us, but, at the same time, a significantly considerable number of parameters, which allows us to improve this model by involving the parameters reduction methods. Moreover, this model shows promising metrics and holds the architecture we can easily modify to apply our algorithms.

Model's Architecture

SNUNet-CD has an encoder-decoder architecture and uses the Siamese network as the encoder. Figure 5.2, initially provided in the [9], illustrates the backbone of the SNUNet-CD model. The model takes two images as input, performs the down-sampling, and restores them to the original size before outputting using a sub-decoder. By using skip connections throughout the model architecture, SNUNet-CD transfers fine localization features from the encoder to the sub-decoders and allows the maintenance of fine-grained information.

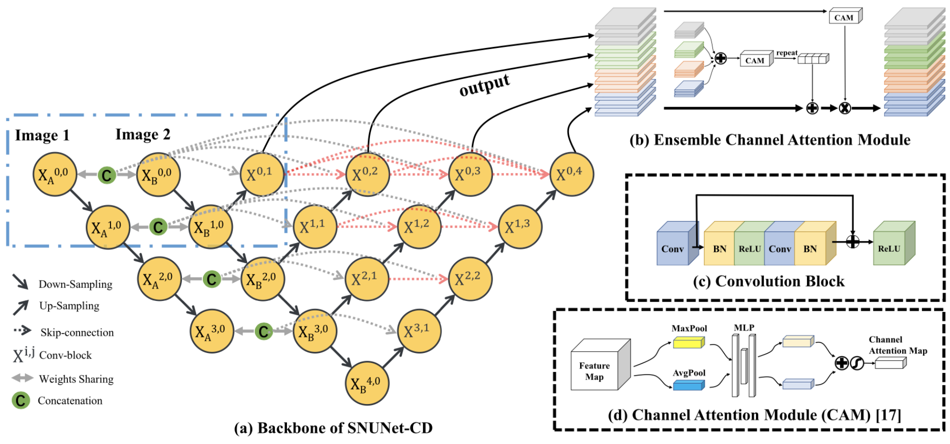


FIGURE 5.2: The illustration, provided in the SNUNet-CD article, of SNUNet-CD model architecture. (a) The backbone of the model. (b) The model's Ensemble Channel Attention Module. (c) A convolution block represented as $X^{i,j}$ on the backbone scheme. (d) Channel Attention Module from [28].: [9] © 2022 IEEE

Dataset

As the dataset for the SNUNet-CD experiments, we use the CDD dataset [17], on which the SNUNet-CD was trained originally. It is designed to solve an automatic change detection problem and consists of season-varying remote sensing images. The dataset introduces synthetic and real remote-sensing images. For our task, we operate with the second class, precisely 16,000 triples, each containing three objects: image1, image2, and the ground truth change mask. Figure 5.3 shows two different examples of the dataset triples.

5.2 Evaluation Metrics

The leading goal of this study is to reduce the number of parameters for a model to make it more efficient in terms of computing and memory usage. Based on this, we utilize the number of parameters as the primary metric for our experiments. The aforementioned metric is a primary indicator in the [31] due to its direct correlation with decreased memory and computational resource utilization. Pruning is a particular case, as this metric is relevant if the parameters are not only pruned (set to zero) but also removed from the parameters set. Therefore, for this approach, we estimate the number of non-zero parameters based on the assumption that pruned parameters do not add value to the calculation during the forward or backward steps.

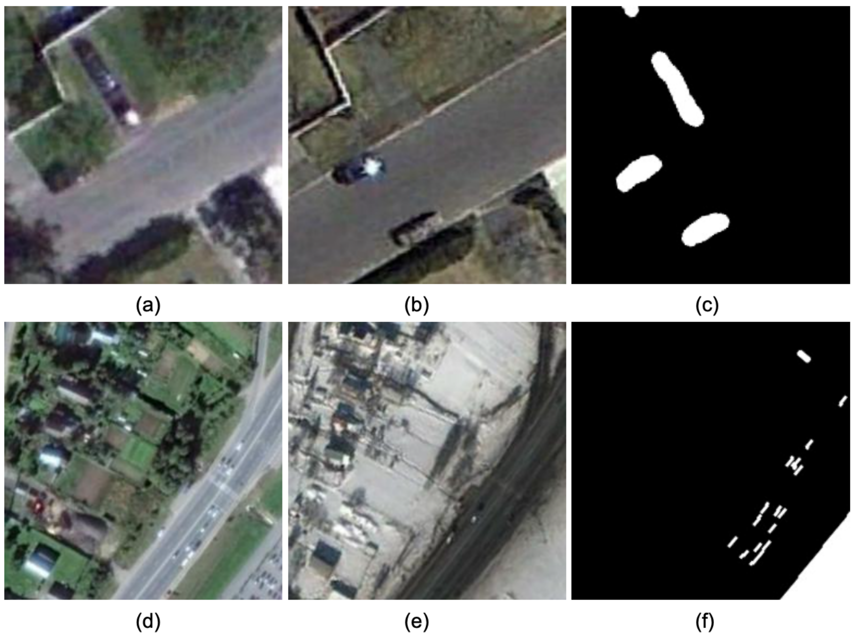


FIGURE 5.3: Two triples examples from CDD dataset. Top images (a), (b), and (c) are examples of image 1, image 2, and a ground truth mask, respectively. The same logic is applied for bottom images (d), (e), and (f). The top images represent urban development and agricultural change, while the bottom images show the change with noticeable seasonal variation.

It is important to note that Low-Rank approximation improves the model efficiency by reducing the rank of the weights matrix. However, it does not modify the number of parameters directly. Nevertheless, we compare this approach with other methods to understand its relative strengths and weaknesses. Specifically, we are interested in evaluating the potential impact of Low-Rank Approximation on the training time and quantitative metrics.

Another metric is the training and inference time. We run all of the experiments on the Google Colab GPU. Therefore, the running time can successfully represent an improvement in model efficiency. Moreover, some of the methods could increase the training time of the model. The reason is that we add additional computational load to the weights calculation process by applying our algorithms.

All of the methods can reduce the model performance while improving its efficiency in terms of computational resource usage. We measure the quantitative metrics to allow the user to find a suitable accurateness-simplicity trade-off. After looking at other Siamese Networks performance evaluations, such as [5, 33, 3], we decided to proceed with Precision, Recall, and F1 score.

Finally, while evaluating the SNUNet-CD, we, despite measuring the performance of different methods within this model, also compare it to other SOTA Siamese Networks in this domain. We use the number of parameters and F1 score as our success indicators in this case.

5.3 Implementation Details

For the entire implementation, we used the programming language Python3 and the framework PyTorch [21]. We created a custom class for a convolutional layer to integrate the parameter reduction techniques into the weights calculation of the model convolutional layer. Then, during the initialization step, calculated the weights with the algorithms discussed in the "Proposed solution" section. Afterward, we replaced the traditional PyTorch convolutional layer with our custom class.

While performing experiments on SNUNet-CD, we specifically worked with the convolutional block, visualized in Figure 5.4. It employs the architecture of the residual unit proposed in [12].

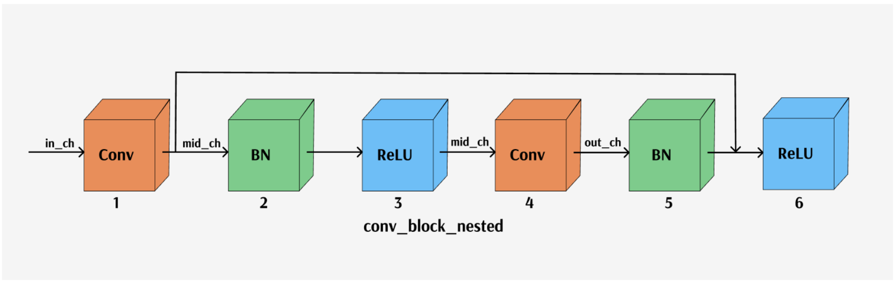


FIGURE 5.4: The structure of convolutional block used in SNUNet-CD model architecture.

It is an open discussion of how many convolutional layers we should replace within a model to reach a significant improvement. As a default version of our implementation, we update only layer 4 of the convolutional block, represented in Figure 5.4). The reasoning is that estimated number of parameters N equals to $N = (k \times k \times in\_ch + 1) \times mid\_ch$ for layer 1, and $N = (k \times k \times mid\_ch + 1) \times out\_ch$ for layer 4. Knowing that $in\_ch < mid\_ch < out\_ch$, we can assume that modifying an element with more parameters would considerably influence the model performance. Regarding running experiments on the toy dataset MNIST, we substitute our custom class for layer 4, displayed in Figure 5.1, with the same logic.

Considering our training process, we developed a classical training for MNIST CNN using Cross Entropy Loss and Adam optimizer [16]. As for the SNUNet-CD, we used a batch size of 16 as in the original implementation. In addition, we modified the training procedure by reducing the number of workers from 8 to 2 and conducting 10 epochs instead of the 100 epochs used by the authors due to our computational limitations.

5.4 Results for CNN on MNIST Dataset

5.4.1 Key Metrics Results

Table 5.1 shows the metrics results for three techniques: Fractional Filter, Pruning, and Low-Rank Approximation, compared with the original MNIST CNN performance. We witness that the Fractional Filter shows the most elevated reduction in parameter number and training time. However, we notice the most prominent performance drop for this approach. Meanwhile, Pruning and Low-Rank Approximation slightly reduce the training time compared to the original model with no accuracy sacrifice.

Regarding the Low-Rank Approximation method, we evaluated different settings: (i) an algorithm based on SVD, (ii) an algorithm based on CUR decomposition

Approach	Params (k)	Training time (sec)	Precision	Recall	F1 score
Original	28.9	372	0.982	0.982	0.982
Low-Rank	28.9	365	0.985	0.985	0.985
Pruning	25.1	365	0.983	0.982	0.982
Fractional	19.2	198	0.968	0.967	0.967

TABLE 5.1: Metrics results for MNIST CNN.

with different values of the parameter c, discussed in more detail in the Proposed Solution section results, we did not observe any significant after analyzing the results changes. Consequently, we decided to set the default implementation of this method to the CUR decomposition with the constant value of the parameter c.

5.4.2 Applying Methods to Various Convolutional Layers

We assume that the efficiency improvements of the model depend on how many convolutional layers we modify with the parameters reduction logic. To more convolutional layers we apply our algorithms, the better results in terms of the number of parameters we can get. We ran such experiments for each approach and noticed a significant shift in metrics results only for the Fractional Filter method. Table 5.2 shows the metrics for the Original model and the model with Fractional Filter applied, depending on which layer is modified: layer 1, layer 4, or both. Refer to Figure 5.1 for the order of the layers.

Approach / layer #	Params (k)	Train time (sec)	Precision	Recall	F1 score
Original / -	28.9	372	0.982	0.982	0.982
Fractional / 1	28.6	108	0.979	0.979	0.979
Fractional / 4 (default)	19.2	198	0.968	0.967	0.967
Fractional / 1&4	18.9	105	0.791	0.790	0.788

TABLE 5.2: Metrics results for MNIST CNN while applying the methods to different convolutional layers in the model.

The more parametrized layers we switch to the Fractional Filter setting, the fewer parameters we get for the whole model, meaning the number of the parameters for layers 1&4 < layer 4 < layer 1.

In addition, we noticed that the training time is significantly lower for layer 1 than layer 4, while the number of parameters results in the opposite way. The reason is that the Fractional Filter method takes more time to compute for more parametrized layers. Another cause lies in the activation functions behavior. Activations have a more considerable impact on the model in the earlier layers than during the last ones. However, when we apply the Fractional Filter for both layers, we get the lowest training time since later activations are already influenced by the activations in the previous layers. Consequently, suppose the project goal is to speed up the training, not to reduce the number of parameters directly. In that case, it is better to modify the first convolutional layer of the network or some of the leading layers.

5.4.3 Increasing the Kernel Size

In this part of the experiments, we aim to test the scalability assumption by comparing the performance of MNIST CNN for kernel sizes $k = 5$ (original value) and $k = 7$. Table 5.3 provides metrics results for Pruning and Fractional Filter, as those methods showed the most considerable improvement during previous experiments.

Approach / k	Params (k)	Train time (sec)	Precision	Recall	F1 score
Original / 7	33.9	395	0.983	0.983	0.983
Original / 5 (default)	28.9	372	0.982	0.982	0.982
Pruning / 7	26.4	410	0.983	0.983	0.983
Pruning / 5 (default)	25.1	365	0.983	0.982	0.982
Fractional / 5 (default)	19.2	198	0.968	0.967	0.967
Fractional / 7	11.9	233	0.834	0.834	0.833

TABLE 5.3: Metrics results for MNIST CNN, considering different kernel sizes k of the convolutional layer to which we apply the methods.

For $k = 5$, Fractional Filter reduces the number of parameters by 33.56%, while for $k = 7$, by 64.90%. Pruning goes from a reduction of 13.15% to 22.12%. Both methods ensure a scalable reduction rate, but Fractional Filter introduces a more significant improvement in terms of parameters number.

An interesting observation of the Fractional Filter method performance is that the number of parameters reduces with the increase of the kernel size. For $k = 7$, the number of the parameters equals 11.9, while for $k = 5$, 19.2. It differs from the trivial behavior we observe for the original architecture and the Pruning method.

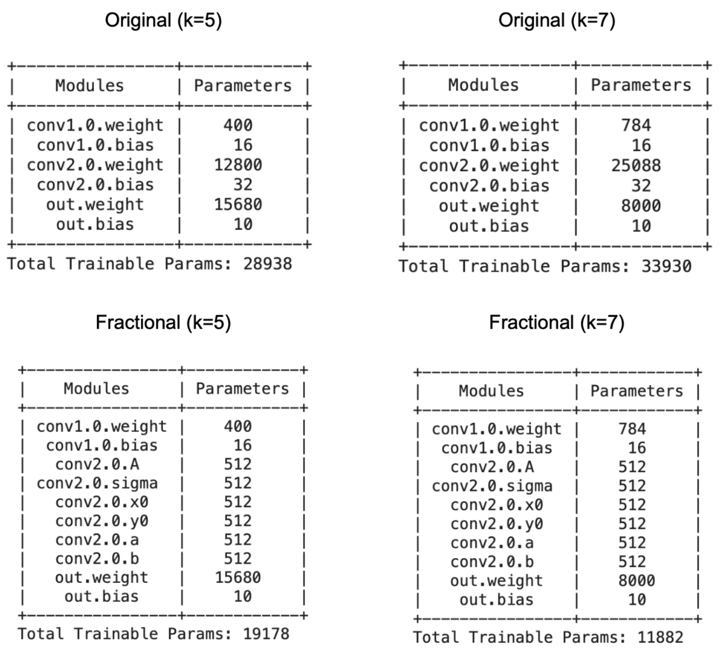
Figure 5.5 shows the number of parameters for the original model and the model with Fractional Filter for different kernel sizes. It shows the distribution of parameters number among all layers in the model. On the graphs, the number of parameters in the second convolution in the Fractional Filter model is fixed for both kernel cases. The reason is that a 2-dimensional Fractional Filter gives a compression from $k \times k$, regardless of kernel size value, to only six trainable parameters (A, σ, x_o, y_o, a, b) per kernel.

5.5 Results for SNUNet-CD

5.5.1 Key Metrics Results

The next significant milestone of our work was to perform the experiments on the SNUNet-CD model designed for Change Detection in Satellite imagery. We discuss the model architecture in more detail in the Experimental Setup section. Note that we performed the training using ten epochs to accommodate limited memory and computational resources. This decision results in relatively worse quantitative metrics (F1 score, Recall, and Precision) than the initial results provided by the authors of the SNUNet-CD model.

According to the metrics results for SNUNet-CD on the training set (see Table 5.4), the highest reduction in parameters number is marked for the Pruning method. It outperforms the Fractional Filter approach, which showed the most potential during the CNN on MNIST dataset experiments. Pruning reduces the number of weights by 10.39% (from 12.03 million parameters to 10.78 million), while Fractional Filter by 4.99% (from 12.03 million to 11.43 million). In addition, Pruning introduces


Original (k=5)

```
+-----------------+------------+
|     Modules     | Parameters |
+-----------------+------------+
| conv1.0.weight  |    400     |
|   conv1.0.bias  |     16     |
| conv2.0.weight  |   12800    |
|   conv2.0.bias  |     32     |
|    out.weight   |   15680    |
|     out.bias    |     10     |
+-----------------+------------+
Total Trainable Params: 28938
```

Original (k=7)

```
+-----------------+------------+
|     Modules     | Parameters |
+-----------------+------------+
| conv1.0.weight  |    784     |
|   conv1.0.bias  |     16     |
| conv2.0.weight  |   25088    |
|   conv2.0.bias  |     32     |
|    out.weight   |    8000    |
|     out.bias    |     10     |
+-----------------+------------+
Total Trainable Params: 33930
```

Fractional (k=5)

```
+-----------------+------------+
|     Modules     | Parameters |
+-----------------+------------+
| conv1.0.weight  |    400     |
|   conv1.0.bias  |     16     |
|    conv2.0.A    |    512     |
|  conv2.0.sigma  |    512     |
|    conv2.0.x0   |    512     |
|    conv2.0.y0   |    512     |
|    conv2.0.a    |    512     |
|    conv2.0.b    |    512     |
|    out.weight   |   15680    |
|     out.bias    |     10     |
+-----------------+------------+
Total Trainable Params: 19178
```

Fractional (k=7)

```
+-----------------+------------+
|     Modules     | Parameters |
+-----------------+------------+
| conv1.0.weight  |    784     |
|   conv1.0.bias  |     16     |
|    conv2.0.A    |    512     |
|  conv2.0.sigma  |    512     |
|    conv2.0.x0   |    512     |
|    conv2.0.y0   |    512     |
|    conv2.0.a    |    512     |
|    conv2.0.b    |    512     |
|    out.weight   |    8000    |
|     out.bias    |     10     |
+-----------------+------------+
Total Trainable Params: 11882
```

FIGURE 5.5: The number of parameters in the MNIST CNN model, distributed by each layer. The top tables represent the results for the Original model, while kernel size equals $k = 5$ and $k = 7$. The bottom tables show the number of parameters of the MNIST CNN with Fractional Filter applied as a second convolutional layer.

slightly higher performance metrics than Fractional Filter, as it works with existing weights and does not modify the parameters set directly.

Approach	Params (M)	Training time (sec)	Precision	Recall	F1 score
Low-Rank	12.03	9828	0.848	0.746	0.791
Original	12.03	6318	0.848	0.746	0.791
Fractional	11.43	6824	0.802	0.692	0.738
Pruning	10.78	7539	0.809	0.727	0.762

TABLE 5.4: Metrics results for SNUNet-CD on the training set.

However, as mentioned during the MNIST CNN experiments, Fractional Filter introduces better scalability. We can achieve a higher parameter reduction with Fractional Filter by employing a larger kernel size (for original SNUNet-CD kernel size equals $k = 3$).

Regarding the Low-Rank Approximation method, we observed that the quantitative metrics and number of parameters do not change compared to the original model performance. Additionally, this method takes the most time to compute. Nevertheless, upon looking at the weights matrix, we noticed that the rank of the new matrix is reduced, implying that the resulting matrix contains fewer independent rows and requires less memory usage. It also lowers the risk of overfitting as the approximation method derives a more generalized pattern than the original weights matrix has. Although this method improves the model's efficiency in some way, we decided to exclude it from the further results discussions as our proposed metrics do not accommodate this method.

Approach	Params (M)	Validation time (sec)	Precision	Recall	F1 score
Original	12.03	1254	0.785	0.782	0.781
Fractional	11.43	1291	0.870	0.567	0.679
Pruning	10.78	1691	0.803	0.713	0.753

TABLE 5.5: Metrics results for SNUNet-CD on the validation set.

Further, we evaluated the performance of SNUNet-CD on the validation set (see Table 5.5). The central metrics patterns are consistent with the training set results. However, in the validation step, it is more apparent that the Fractional Filter method has significantly lower recall than the other methods. It results in the model's failure to identify some of the actual changes in the image, resulting in a higher rate of false negatives. Figure 5.6 presents examples of the predicted changes by the Fractional Filter method with high and low losses. During the initial training steps, many images show poor recall. However, as training continues, the predictions tend to improve significantly.

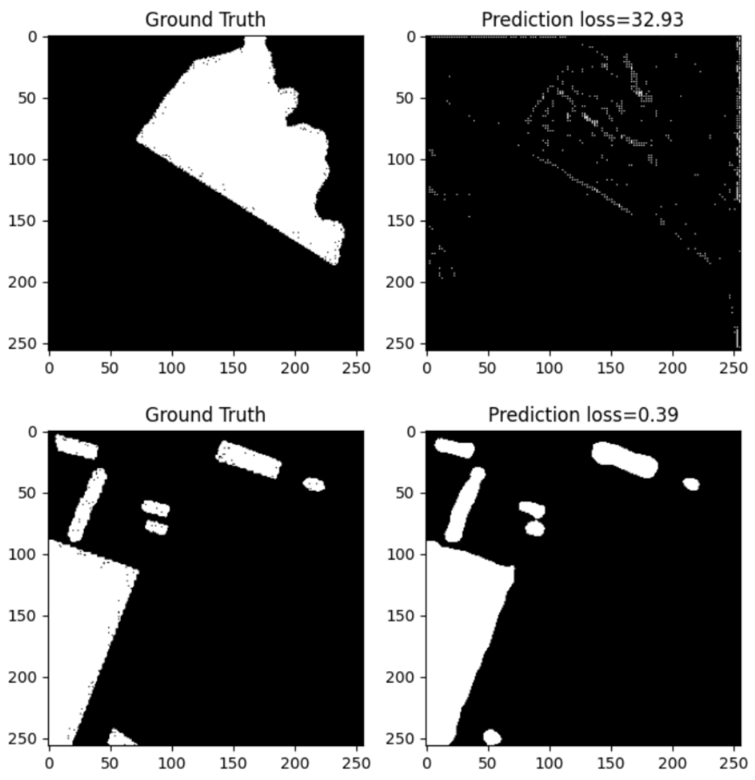


FIGURE 5.6: Ground truth images and predicted change maps (with high/low loss) for the Fractional Filter approach.

5.5.2 Metrics Distributed For Each Epoch

Furthermore, we calculated the F1 score for each epoch individually (see Figure 5.7) to track how quickly the model converges. It allows us to make more reasonable conclusions regardless of the number of epochs for training.

We can see that Fractional Filter performs more poorly than other methods regarding the F1 score, which is expected as it provides a significant parameters reduction rate for the model. Pruning also sacrifices the model performance, but less than the Fractional Filter method. The plots suggest that the F1 score goes up with each epoch, which means that the utilization of a higher number of epochs potentially introduces a more elevated F1 score than we currently get.

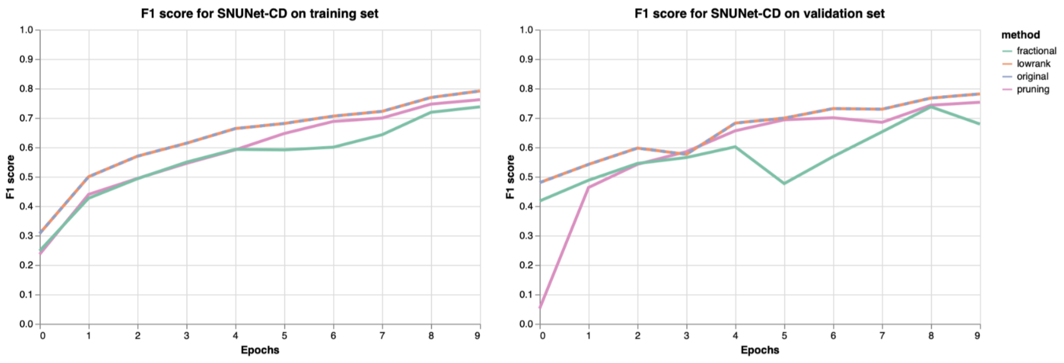


FIGURE 5.7: F1 score curve for SNUNet-CD during training on train and validation sets.

Table 5.6 compares the training time measured for each epoch individually and the inference time of the models with different parameter reduction methods applied. As a result, we see that regardless of the training time for the first epoch, after the second one, the time stabilizes and remains the same for all approaches.

Moreover, although we observe the lowest training time for the original architecture, the Fractional Filter approach outperforms all other methods regarding inference speed. It is an important observation, as some of the projects in real-life scenarios care about inference time more than the training process.

Approach	Train / 1	Train / 2	Train / 3	Train / 4	Train / 5	Inference
Pruning	2312	584	582	582	582	2445
Fractional	1713	572	570	570	568	1100
Original	1292	560	558	559	558	2433

TABLE 5.6: Training time for the first five epochs and an inference time for SNUNet-CD.

5.5.3 Benchmarking with Other Change Detection Models

Let us start by comparing the performance of the SNUNet-CD model trained by us and the results provided by the authors of [9]. The difference is in the number of epochs used for training the model. The number of parameters appears the same for both cases, which is an expected outcome as we employ the same model architecture. However, our experiments show poorer performance in terms of metrics: 0.953 F1 score reported by the authors and 0.791 presented by us. That leads us to the conclusion that we cannot expect a fair comparison of our experiments and other SOTA models in the domain. The results obtained within this work are lower than the expected performance of the SNUNet-CD model in higher computational resource settings. Nevertheless, our methods still outperformed some models in the Change Detection domain (see the results in Table 5.7).

All of our methods show a smaller number of parameters than IFN and DASNet models. Moreover, the SNUNet-CD model with Pruning applied outperforms UNet++_MSOF by the same metric. It is an important finding as the original SNUNet-CD with 12 channels was unable to surpass this method in terms of parameter count. In addition, we propose a higher F1 score than the less-parametrized models, such as FC-Siam-diff and FC-Siam-conc, even considering our limited computational resources.

Model / Channels #	Params (M)	F1 score
IFN [33]	35.72	0.903
DASNet [3]	16.25	0.919
SNUNet-CD Low-Rank	**12.03**	**0.791**
SNUNet-CD Original	**12.03**	**0.791**
SNUNet-CD [9]	12.03	0.953
SNUNet-CD Fractional	**11.43**	**0.738**
UNet++_MSOF [22] / 32	11.00	0.876
SNUNet-CD Pruning	**10.78**	**0.762**
FC-Siam-diff [5] / 32	5.39	0.692
FC-Siam-conc [5] / 16	1.54	0.637
FC-Siam-diff [5] / 16	1.35	0.652

TABLE 5.7: Comparison of the number of parameters and F1 score for SNUNet-CD and other SOTA models in the domain. The scores for all models except our experiments are obtained by the authors of [9].

Lastly, we assume that the project goals and resource availability may differ. Therefore, we leave the decision of what model and which parameter reduction method to use into consideration for the reader.

Chapter 6

Conclusions

6.1 Results Summary

Reducing the number of parameters in Change Detection models holds significant potential for optimizing model performance. It leads to the reduction of computational resources and memory usage, which makes a model more compact and efficient. Our experimental results have shown that this concept is particularly effective in the Satellite imagery field, where large-scale images are processed.

Our study involved a comprehensive analysis of existing parameter reduction methods. The central part of this work involves implementing three approaches Fractional Filter, Pruning, and Low-Rank Approximation, and applying those to two models, namely CNN for the MNIST dataset and the Siamese Network designed for Change Detection in Satellite Imagery. Moreover, one of this work's contributions is our implementation of the Fractional Filter approach, established on fractional derivatives theory. It is not generally available in open-source resources, contrasting with other methods we examined.

The preliminary experiments on the MNIST CNN have shown the highest parameter reduction rate with the Fractional Filter method applied. It equals 64.90% for kernel size $k = 7$ and 33.56% for $k = 5$.

Regarding the SNUNet-CD model designed for the change detection task, we achieved a considerable reduction using Pruning and Fractional Filter methods, 10.39% and 4.99% rates, respectively. With the highest reduction rate of 10%, the F1 score drop equals only 3.65%, which allows our implementations to remain accurate while becoming more effective in computing and space usage. Moreover, the SNUNet-CD model, with our updates applied, outperformed some of the existing models in the change detection domain.

6.2 Limitations and Future Work

While the pruning method could reach a higher reduction rate by modifying the pruned weights percentage parameter, it comes at the cost of sacrificing a significant accuracy percentage. It introduces the main limitation of this work, which does not allow us to decrease the number of parameters further and maintain good accuracy at the same time.

Moreover, the trade-off problem introduces the possibility of future work. It is worth trying to experiment with our methods on a broader range of models to identify generalized patterns, if any. Another possibility is to apply other techniques for improving model efficiency. This piece focuses specifically on operating with the parameter set. However, in the Related Work chapter, we discussed other options to make the model more efficient, such as model architecture modification. Finally, the question of an optimal trade-off remains an open challenge for researchers in Change Detection. We hope our study will inspire further study in this area.

Lastly, our study can be applied to various domains in the change detection field, including medical detection, quality control, traffic analysis, and financial studies. The methods analyzed within this work can be integrated into models beyond Change Detection. For instance, Speech Recognition and Generative Models usually operate with a high number of parameters and could potentially benefit from our research.

Bibliography

[1] Gaoyuan Cai et al. "Learning and Compressing: Low-Rank Matrix Factorization for Deep Neural Network Compression". In: *Applied Sciences* 13 (Feb. 2023), p. 2704. DOI: 10.3390/app13042704.

[2] Turgay Celik. "Unsupervised Change Detection in Satellite Images Using Principal Component Analysis and *k*-Means Clustering". In: *IEEE Geoscience and Remote Sensing Letters* 6.4 (2009), pp. 772–776. DOI: 10.1109/LGRS.2009.2025059.

[3] Jie Chen et al. "DASNet: Dual Attentive Fully Convolutional Siamese Networks for Change Detection in High-Resolution Satellite Images". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 14 (2021), pp. 1194–1206. DOI: 10.1109/jstars.2020.3037893. URL: https://doi.org/10.1109\%2Fjstars.2020.3037893.

[4] Zhou Daquan et al. *Rethinking Bottleneck Structure for Efficient Mobile Network Design*. 2020. arXiv: 2007.02269 [cs.CV].

[5] Rodrigo Caye Daudt, Bertrand Le Saux, and Alexandre Boulch. *Fully Convolutional Siamese Networks for Change Detection*. 2018. arXiv: 1810.08462 [cs.CV].

[6] Jin Deng et al. "PCA-based land-use change detection and analysis using multitemporal and multisensor satellite data". In: *International Journal of Remote Sensing - INT J REMOTE SENS* 29 (Aug. 2008), pp. 4823–4838. DOI: 10.1080/01431160801950162.

[7] Li Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477.

[8] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. *Relative-Error CUR Matrix Decompositions*. 2007. arXiv: 0708.3696 [cs.DS].

[9] Sheng Fang et al. "SNUNet-CD: A Densely Connected Siamese Network for Change Detection of VHR Images". In: *IEEE Geoscience and Remote Sensing Letters* 19 (2022), pp. 1–5. DOI: 10.1109/LGRS.2021.3056416.

[10] Ritwik Gupta et al. *xBD: A Dataset for Assessing Building Damage from Satellite Imagery*. 2019. arXiv: 1911.09296 [cs.CV].

[11] Song Han et al. *Learning both Weights and Connections for Efficient Neural Networks*. 2015. arXiv: 1506.02626 [cs.NE].

[12] Kaiming He et al. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: 1603.05027 [cs.CV].

[13] Forrest N. Iandola et al. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*. 2016. arXiv: 1602.07360 [cs.CV].

[14] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán. "Low-Rank Compression of Neural Nets: Learning the Rank of Each Layer". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8046–8056. DOI: 10.1109/CVPR42600.2020.00807.

[15] Kevin Louis de Jong and Anna Sergeevna Bosman. *Unsupervised Change Detection in Satellite Images Using Convolutional Neural Networks*. 2019. arXiv: 1812.05815 [cs.NE].

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[17] M. Lebedev et al. "CHANGE DETECTION IN REMOTE SENSING IMAGES USING CONDITIONAL ADVERSARIAL NETWORKS". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2 (May 2018), pp. 565–571. DOI: 10.5194/isprs-archives-XLII-2-565-2018.

[18] Shilong Liu et al. "Experimental realisations of the fractional Schrödinger equation in the temporal domain". In: *Nature Communications* 14.1 (2023). DOI: 10.1038/s41467-023-35892-8. URL: https://doi.org/10.1038\%2Fs41467-023-35892-8.

[19] Microsoft. *Building Damage Assessment with Siamese Networks*. https://github.com/microsoft/building-damage-assessment-cnn-siamese. 2020.

[20] Tatiana Odzijewicz, Agnieszka B. Malinowska, and Delfim F.M. Torres. "Fractional variational calculus with classical and combined Caputo derivatives". In: *Nonlinear Analysis: Theory, Methods &amp Applications* 75.3 (2012), pp. 1507–1515. DOI: 10.1016/j.na.2011.01.010. URL: https://doi.org/10.1016\%2Fj.na.2011.01.010.

[21] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[22] Daifeng Peng, Yongjun Zhang, and Haiyan Guan. "End-to-End Change Detection for High Resolution Satellite Images Using Improved UNet++". In: *Remote Sensing* 11.11 (2019). ISSN: 2072-4292. DOI: 10.3390/rs11111382. URL: https://www.mdpi.com/2072-4292/11/11/1382.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[24] Bertram Ross. "A brief history and exposition of the fundamental theory of fractional calculus". In: *Fractional Calculus and Its Applications: Proceedings of the International Conference Held at the University of New Haven, June 1974*. Ed. by Bertram Ross. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975, pp. 1–36. ISBN: 978-3-540-69975-0. DOI: 10.1007/BFb0067096. URL: https://doi.org/10.1007/BFb0067096.

[25] Jost Tobias Springenberg et al. *Striving for Simplicity: The All Convolutional Net*. 2015. arXiv: 1412.6806 [cs.LG].

[26] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[27] Mingxing Tan et al. *MnasNet: Platform-Aware Neural Architecture Search for Mobile*. 2019. arXiv: 1807.11626 [cs.CV].

[28] Sanghyun Woo et al. *CBAM: Convolutional Block Attention Module*. 2018. arXiv: 1807.06521 [cs.CV].

[29] Chai Wah Wu. *ProdSumNet: reducing model parameters in deep neural networks via product-of-sums matrix decompositions*. 2019. arXiv: 1809.02209 [cs.LG].

[30] Pengfeng Xiao et al. “Direction-dominated change vector analysis for forest change detection”. In: *International Journal of Applied Earth Observation and Geoinformation* 103 (2021), p. 102492. ISSN: 1569-8432. DOI: https://doi.org/10.1016/j.jag.2021.102492. URL: https://www.sciencedirect.com/science/article/pii/S0303243421001999.

[31] Julio Zamora et al. “Convolutional Filter Approximation Using Fractional Calculus”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*. 2021, pp. 383–392.

[32] Julio Zamora Esquivel et al. “Adaptive Activation Functions Using Fractional Calculus”. In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. 2019, pp. 2006–2013. DOI: 10.1109/ICCVW.2019.00250.

[33] Chenxiao Zhang et al. “A deeply supervised image fusion network for change detection in high resolution bi-temporal remote sensing images”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 166 (2020), pp. 183–200. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2020.06.003. URL: https://www.sciencedirect.com/science/article/pii/S0924271620301532.