DOI:10.15774/PPKE.ITK.2023.005

Increasing the robustness of deep neural networks against adversarial attacks and solving other prominent problems in the application of machine learning

Alafandi Jalal

PhD dissertation

supervisor:
Dr. Horvath Andras PhD

Faculty of Information Technology and Bionics
Pázmány Péter Catholic University

Budapest, 2022

DOI:10.15774/PPKE.ITK.2023.005

DOI:10.15774/PPKE.ITK.2023.005

Contents

DOI:10.15774/PPKE.ITK.2023.005

DOI:10.15774/PPKE.ITK.2023.005

DOI:10.15774/PPKE.ITK.2023.005

DOI:10.15774/PPKE.ITK.2023.005

Chapter 1

Introduction

Machine learning is an ubiquitous approach which has been successfully used in many applications to find a practical solution for a complex problem, e.g., medical image segmentation [1], self-driving cars [2], malware detection [3], data compression [4], language translation [5]. Recently, deep learning has been used intensively for safety and security-critical applications, like autonomous vehicles [6] and malware detection [3], but the security and verifiability of these approaches are unknown. It even supports other applications, e.g., facial recognition, machine translation and speech recognition which may effect our daily lives.

My thesis will cover four different topics in many machine learning applications. In the first topic, we will enhance the performance of genetic algorithm by modifying the mutation operator which is an important step in any evolutionary algorithm.

The other three topics will focus more on the application of deep learning which is a new field that got popular in 2012 due to its state of the art performance in many applications, without an adequate analysis to the theoretical limitations causing many empirical problems. One of the biggest obstacles of using neural network is the need of a differentiable objective function which is not available for every optimization problem. Traditional gradient based optimization algorithm, which is the base of most deep learning methods, can only deal with the problems which can be defined by differentiable equations. There are many algorithms which can deal with non-differentiable problems by relying on the statistic of a stochastic search e.g. genetic algorithm and random search.

DOI:10.15774/PPKE.ITK.2023.005

One variation of deep leaning which tackle the problem of non-differentiable objective function is reinforcement learning.Reinforcement learning is a framework to solve a reward-related learning problem where instead of solving the objective function explicitly, the algorithm evaluates each part/policy of a solution independently relying on the statistics tuning the parameters to achieve the highest reward. Although reinforcement learning revolutionized many applications, many optimization problems can't be defined to fit reinforcement learning framework.

Genetic algorithm is a general framework to solve most optimization problems by evaluating each solution and selecting the best solution in an iterative manner. One of the main issues of using genetic algorithm is that you can't evaluate part of a candidate solution which means any modification over the solution will be random. We will tackle this issue with our proposed solution [2] forcing the mutation to be more guided.

The second topic, which I will cover in my thesis, concerns adversarial attack which is another flaw of deep learning. This complex problem has to be solved before we perpetuate the usage of neural network in practical application.

Evasion attacks are the most common adversarial attacks where a corrupted sample is manufactured to deceive the network by modifying the input with unnoticeable perturbations changing the response of the network. While other kind of attacks try to contaminate the training dataset or collect enough information about the distribution of the dataset and the model to build a network which is able to create malicious samples. Adversarial attack, which undermines the popularization of neural network, is a genuine threat compromising the safety of many intelligent systems curbing the standardization of using neural networks in security-critical applications. Since the emergence of adversarial attacks, the research community has worked relentlessly to avert the malicious damage of these attacks.

Many recent publications in the machine learning community were working on finding a way to defend neural networks against adversarial samples [7] [8].Using external model as a detector is the most reliable choice providing the highest accuracy. Other ways, which we can find in the literature, to mitigate the impact of adversarial samples are modifying the training processor or modifying the network architecture. Even with an optimal detector, an attack can cause a halt

DOI:10.15774/PPKE.ITK.2023.005

in the system hindering the achievement of any task. I will introduce an algorithm [3] which can retrieve the original class by simply counter attacking all other classes.

The other two topics enhance the performance of deep learning algorithm in two different tasks, image classification and image segmentation, by proposing a solution to certain limitations of neural network training.

Apart from solving the limitations of neural network, another thread of investigation is involved in the continuous improvement of neural networks where immense number of papers are issued everyday using neural network as the main approach of their investigations, enhancing the performance of neural networks and creating many open questions that should be answered by our research community. Enhancing the performance of any method, which solves a particular problem, is a contentious process where a threshold has to be reached before utilizing this method in practice. Introducing a new idea, which slightly increases the performance, can lead to other beneficial improvements perpetuating the continuous development of machine learning algorithms that won't ever reach the optimum state as it always going to rely on the input distribution. Improving an existing method can be done by solving a shortcoming, increasing the size of the model, introducing a new step or relying on another method e.g. data prepossessing or ensemble learning.

My third topic concerns image segmentation which use neural networks as the backbone of their state of the art models. Algorithms designed for image segmentation require the definition of a loss function for the comparison between network outputs and desired output images. Topographic properties are not taken into account in commonly applied loss functions, they handle all pixels in a similar matter. Our brain can compare complex objects with ease and considers both pixel level and topological differences simultaneously Comparison between objects requires a properly defined metric that determines similarity between them considering changes both in shape and values e.g. snakes and cars with the same misclassified area should not have the same loss. I will introduce a new loss function, called wave loss, which can incorporate spatial information in the loss function of image segmentation training [4].

DOI:10.15774/PPKE.ITK.2023.005

The last topic which I will address in my thesis enhance the performance of deep learning over image classification and object detection tasks. I will challenge and negate the common assumption that the activation of different layers in neural networks follow Gaussian distribution. Activations do not necessarily follow Gaussian distribution in all layers where neurons in deeper layers are more and more specific which can result extremely large, out-of-distribution activations. The distribution of the activations can be transformed using normalization techniques, such as batch-normalization and group normalization, which can help with increasing the convergence speed and improving the accuracy. Normalization layers force a shifted normal distribution by calculating the mean and the variance of the activations while learning the appropriate shifting parameters. The extremely out-of-distribution activations can result inconsistent mean and variance values crippling the convergence of the network and causing an overfitting. I will introduce an algorithm 5 to filter out these activation which can help with speeding up the training and improve the performance of the models.

DOI:10.15774/PPKE.ITK.2023.005

Chapter 2

Improving genetic algorithm with locus mutation

2.1 Introduction

Neural network training, as a gradient descent method, revolutionized machine learning applications in many problem settings e.g. image classification, image segmentation, data compression, and language processing. Gradient descent algorithm gives the optimal direction inside your local search space toward the local minimum or maximum. Using gradient descent with backpropagation, which can tell you the optimal direction for a change, solves the optimization problem faster by exploiting the neighbors of the current parameters. Gradient based methods like stochastic gradient descent [9] or ADAM optimizer [10] have been the key for the recent advancements of deep learning which surpass any other optimization approach. Although gradient descent seems like the perfect approach for any optimization problem, it is not applicable and ill-defined for non-differentiable objective functions and it is vulnerable against non-convex problems [11]. Evolutionary algorithms are good alternatives that can overcome the aforementioned theoretical limitation of gradient descent approaches and sometimes can be even faster than hill-climbing optimization [12]. Evolutionary Algorithms can provide a solution; sometimes sub-optimal but still applicable in practice, without any deep knowledge of the system due to the random generation of a new population. Researchers started to integrate the two approaches striving for the optimal solution where evolutionary algorithms have been used for hyperparameter search

DOI:10.15774/PPKE.ITK.2023.005

[13] and reinforcement learning [14], which demonstrates the importance of GAs in machine learning as well.

Genetic algorithm (GA), as the most prominent evolutionary algorithm, is a probabilistic and heuristic search approach to investigate encoded solutions in an iterative manner, which was successfully applied in various practical applications, ranging from image processing [15], general optimization problems [16], biological sciences and bioinformatics [17, 18], finance, economics, and social sciences [19, 20], speech processing [21] to path planning [22].

In the case of evolutionary algorithms, a fitness function, determining the quality of each solution is used instead of a detailed formal description and analytical solution of the problem. The algorithm starts the search for the optimal solution with an initial generation encoding a set of randomly created solution candidates. Genetic algorithm in most common cases consists of three operations (selection, crossover, and mutation) which are used repeatedly to create a new generation until reaching the solution with the designated threshold or stopping after a fixed number of generations. We produce a new generation by using the aforementioned fitness function to select a percentage of the best solutions from the current generation and then recombine them to yield new offspring (solution candidates). Before evaluating the new generation, we apply mutation inducing small changes in the solution candidates to maintain population diversity. Although some papers [23] have used only mutation algorithm to create newer generations, the combination of crossover and elitism usually increases convergence speed towards the optimal solution [24].

Genetic algorithm as any other optimization algorithm could get stuck in a local optimum; a problem that can be solved by increasing the exploration rate. The exploration-exploitation dilemma is the most common trade-off problem between obtaining new knowledge and the necessity to use that knowledge to improve performance; a problem which can be found everywhere in nature [25]. This problem manifests in genetic algorithm as well, where when only applying mutation, to randomly create chromosomes [1], increases the exploration rate to the utmost resembling random search which is a time-consuming and impractical high-dimensional problem. On the other hand, selecting only the best solution and

[1]It is a term referring to a candidate solution and it will be explained in detail in section 2.3

DOI:10.15774/PPKE.ITK.2023.005

spreading it in the next population would increase the exploitation rate making the algorithm unrobust which will only lead to the first local optimum in the initial population. One would like to exploit by moving toward the best solution, but also explore to maintain a diverse population; and delve into the best current solutions finding the optimal or sub-optimal solutions, but still avoid local optimum.

The most important factors affecting the accuracy of the final solution and the convergence of the algorithm are the format of the problem representation and the fitness function, which we need to assess the validity of each solution. These two factors will determine the space of all solution candidates. These elements are usually determined by heuristic approaches and are always problem-dependent. Other factors, like different mutation and crossover methods, can be investigated more generally.

One of the most important and ubiquitous operations is mutation which has a large effect on the convergence of GAs [26]. Throughout the literature, researchers have been using static or adaptive mutation [27]. In the early implementations, static mutation probability was applied, where the mutation rate was optimized by heuristic approaches with trial and error. In these approaches, a single parameter was identified determining the rate of mutation, and was applied in the same manner for each gene in each chromosome. Later, adaptive versions appeared where mutation rate can be changed according to other state variables of the algorithm, like iteration number, quality of the selected solution candidates, or average fitness for all the solutions.

One of the major hindrances of most algorithms is parameter tuning where an appropriate parameter setting has to be chosen, but it can be a hard task due to parameters' reliance on the representation of the optimization problem. Parameter tuning relies on experimenting with many fixed parameters to reach the supposedly optimal parameter setting without taking into account the possible changes throughout the optimization process. Besides the time-consumption problem, another problem of parameter tuning is that the chosen parameters may only work well starting from a specific state of the search space. Another parameter scheme is dynamic parameter control which changes parameters adaptively, during execution. When first introduced, it relies on the state of the optimization process or on time [28]. Parameter control later adopted a self-correcting, self-adaptive,

DOI:10.15774/PPKE.ITK.2023.005

approach adjusting parameters while in progress relying on the feedback from the algorithm's recent performance. Self-adaptation can be used as a strong, effective tool for steering the parameters with the help of some performance assistance e.g the fitness of a chromosome and the fitness of a population.

Our method, called Locus mutation, extends the traditional approach of mutation where the probability of the alteration of a gene is uniformly distributed over each position of the genome in one sample. Locus mutation applies an additional probabilistic weight for each gene (i.e., location or dimension), thus, dimensions with higher probability will be selected more frequently, where an alteration could lead to a better solution with higher accuracy. Locus mutation resembles the traditional optimization approaches, back-propagation or gradient descent; where in the case of a solution that is represented as a vector, our approach would point out the location which is recommended to be changed and keep satisfying segments intact.

In Section 2.2 we will introduce genetic algorithms, in Section 2.3 we will describe our alterations, describe our method, and demonstrate results on the N-Queen problem, in Section 2.4 we will show how our approach can be used on other problems as well, like the traveling salesman problem and in the last Section we will conclude our results.

2.2 Genetic Algorithm

Genetic algorithm is a heuristic approach, exploring the search space and exploiting local optima. A gene is a singular element, encoding one dimension of the problem and representing a fragment of a solution. Problem representation is an essential prerequisite that can heavily determine the result of the algorithm. Another essential prerequisite condition to any optimization approach is its fitness function F which depends on the problem itself. Most sophisticated optimization approaches only deal with differentiable loss functions, fitness functions, which do not always exist. In case of general problems, the loss function is not differentiable, but we still have an estimation of the quality of a solution that can be used in evolutionary algorithms.

DOI:10.15774/PPKE.ITK.2023.005

Since algorithm convergence is determined heavily by the selected heuristics, we will distinguish between two different problems. In the case of many practical problems, the fitness function measures a deviation from the optimal solution where the value of the best possible solution is known (and usually is zero, like in the case of the N-Queens problem [29]), meanwhile in other sets of problems (like traveling salesman [30] or knapsack problem [31]) the fitness value of the optimal solution is unknown. Unlike the second set of problems, the first set of problems has a stopping criterion yielding a conceptual optimum. First, we will demonstrate our solution using the first set of problems, but later we will illustrate that it can even be applied in the case of the latter problem types as well.

The population will evolve exploiting the best solutions by selection and then applying crossover operations amongst them. Ordering the chromosomes using the contrived fitness function and then selecting a fixed percentage of the most fitted ones would help converging the population towards a better solution. Copying a small unchanged proportion of the fittest chromosomes into the next generation is called elitism which can steer the algorithm towards local optima. The new population consists of the selected elite chromosomes, combined chromosomes from the selected ones, and new random chromosomes. Crossover exploits the best candidate solutions by combining them taking into account problem representation, creating only valid solution candidates. In addition to the randomly generated solutions, mutation has been used to increase the exploration rate searching for the optima. Intuitively, an adaptive mutation rate has been adopted where the mutation will be mitigated over time while converging to the optimal solution.

The traditional approach of genetic algorithm is presented in Algorithm 1 as a pseudo-code in form of simple functions. Searching for the optimal solution $Optimum$, we initialize the population Pop with random values. There is a trade-off problem between the size of the population $PopSize$ and the number of populations $IterNum$ which can be investigated with parameter tuning. Starting with an enormous $PopSize$ and small $IterNum$ could increase the exploration rate but yield a small exploitation rate. On the other hand, having small $PopSize$ and large $IterNum$ would limit the exploration of the search space. Our work focused on the mutation rate $MutRate$ which drives the mutation operation $M()$. A fitness function $F()$ is used to select $S()$ the elite which is a small percentage

DOI:10.15774/PPKE.ITK.2023.005

of the fittest chromosomes. Crossover $C()$ is used after the selection process exploiting the elite. Lastly, we mutate $M()$ the current Pop with $MuteRate$ probability. In a repetitive manner as many as the $IterNum$ and in chronological order, we apply the previously mentioned steps.

Algorithm 1: Genetic algorithm main steps

Parameters: $PopSize, MutRate, IterNum$ **Result:** $Optimum$
$Pop =$ population initialization
for $i \leftarrow 0$ **to** $IterNum$ **do**
    $Values = F(Pop)$
    $Pop = S(Pop, Values)$
    $Pop = C(Pop)$
    $Pop = M(Pop, MutRate)$
end

Adaptive mutation can be divided into three categories: population level, individual level, and component level adaptation [32]. Population-level adaptation changes the mutation probability globally using feedback information from the previous population which means all the chromosomes have the same probabilistic chance for modification. At the individual level, each chromosome has its own adaptive operator which has been induced from the statistics of the previous generations. While the component-level adaptation tries to combine the two previous methods by grouping the chromosomes and setting a different adaptive operator for each group.

However, it is important to note that none of the previously mentioned approaches utilize the statistical information inside the chromosome, which can be induced from the genes. To overcome this deficiency, we come up with the novel idea of locus mutation, where every gene has its own different adaptive operator and problematic genes have a higher chance for change. Identifying the problematic genes is an important factor of our algorithm, and in our approach, problematic means those genes which are mostly responsible for the high values in the error function (e.g., the number of queens hitting each other in the Case of the N-Queens problems). The error function is something we typically minimize and the fitness function is something that we typically maximize, but with genetic algorithm usually, the objective function is called the fitness function, which in

DOI:10.15774/PPKE.ITK.2023.005

our case is an error function which we would like to minimize, thus we will always refer to the objective function of genetic algorithm in this chapter with fitness function. When the aim of the algorithm is to maximize the fitness value as opposed to our case, we can consider the same function multiplied by -1.

2.3 Locus Adaptive Genetic Algorithm

All possible solutions in the initial population (Pop) are sampled randomly from the search space. The algorithm is iterated $IterNum$ times where at each step, the population is continuously changing and better samples are selected and recombined; this helps with increasing the average fitness value of the population over time. Thus, the time dependency of the population can be noted by Pop_t which denotes the population at iteration t. At each iteration within the population, each sample is usually referred to as a chromosome. A chromosome is one possible solution; a solution candidate; and this can be referenced to as Pop_{tk} where k = 1 ... N chromosomes. A chromosome is a vector representing a solution, which can be further divided into individual elements (Like a position of a single queen on a chessboard) this is noted by a third index Pop_{tkl} where l = 1 ... M genes. In the traditional approach, the selection of a position for mutation is a random process and its major goal is the exploration of the high-dimensional search space without taking the current state of the chromosome into account. Optimal selection of the gene which will be modified requires a comprehensive knowledge of three different variables; the statistics of the inter and intra populations, the chromosomes as a function of time, and the statistic of the genes' competencies. Scrutinizing the relationship between these three variables and the fitness function will lead us to the optimal modification of every gene. Although seemingly the optimal solution can be attained from equation (2.1), it is not practical and both memory and time-consuming because you need to keep track of all generations, chromosomes, and genes throughout the algorithm.

$$PM(Pop_{tij}) = F(Pop_{qkl}^{l=1...M,k=1...N,q=1...t}, i, j) \tag{2.1}$$

PM function calculates the probability of mutation for a given gene j and F is a function calculating the mutation rate of gene j taking into account all

DOI:10.15774/PPKE.ITK.2023.005

previous generations (q), chromosomes (k) and genes (l). A lot of attempts have been made to calculate an adaptive mutation operator using one of the aforementioned variables, but the authors are not aware of any method that has used the genes statistic to form a gene-level mutation. Any mutation method can be rewritten as in equation (2.1) using constant parameters as we will see in the following paragraphs. Traditional Genetic algorithm uses static operators as defined in equation (2.2) which means that all chromosomes and genes throughout all generations would have the same mutation probability although some candidate solutions are closer to the optimal solution than others.

$$PM(Pop_{tij}) = C \tag{2.2}$$

C is a constant value for every iteration, chromosome and gene. The static mutation is good until it gets stuck in a local minimum. After we reach a local minimum, we can walk back down the hill and try another angle craving for the optimal solution or try to jump from the local minima by increasing/decreasing the mutation rate or applying crossover. Parameter tuning is a manual, time-consuming, and unpleasant road which can be superseded with parameter control [27]. Parameter control means starting from an initial value and then tuning it adaptively during execution as in the following approaches. With the presupposition of converging to the optimal solution over time, an adaptive mutation subjected to time as in equation (2.3) has been proposed.

$$PM(Pop_{tij}) = F(t) \tag{2.3}$$

F is a function of time which depends on t but does not depend on the chromosome i or the gene j. Once the population is determined the probability of mutation is the same for every chromosome and gene ($PM(Pop_{tij}) = PM(Pop_{tkl}) \forall i, j, k, l$) in that iteration. Dynamic mutation takes the number of the current iteration as an input and gives us the mutation rate as an output. The function used to calculate the mutation rate can be a linear function that relies on the fact that it is beneficial to have a high mutation rate at the beginning and lower mutation rate during convergence [33], a gaussian function [34] where the mutation rate is going to increase smoothly until reaching an apex then decrease steadily converging to zero, Lévy distribution [35] or any arbitrary function. It is not the best approach

DOI:10.15774/PPKE.ITK.2023.005

having the same probability for each chromosome which could be very close to the optimal solution or far away from it.

Another approach with the same problem is population adaptive operator [36, 37] as in equation (2.4) where each generation has a different mutation operator which is deduced from the generation statistics. In [38], more than one mutation operator are used with an equal initial probability (1/the number of operators), but after each iteration, the probabilities will increase/decrease according to the fitness values of each operator designated offspring. In general, these approaches can be defined as:

$$PM(Pop_{tij}) = F(Pop_{tkl}^{l=1...M,k=1...N}) \tag{2.4}$$

F will determine how the mutation depends on all the fitness values in the current population for every i and j (chromosome and gene). Again once the population is determined, the probability of mutation is the same for every chromosome and gene ($PM(Pop_{tij}) = PM(Pop_{tkl}) \forall i, j, k, l$). To solve this problem an individual adaptive mutation [39, 40] was proposed as in equation (2.5) where each chromosome has its own different mutation operator that can be concluded from the statistic about the search space of each chromosome through the populations which is, the statistic, implicitly maintained by the algorithm. Mutation rates can change not only in different iterations but also at the same generation where better candidates will have lower mutation rate, meanwhile worse candidates will have higher mutation rates. Each chromosome will have a different mutation rate which is proportional in comparison to the other chromosomes in the current population.

$$PM(Pop_{tij}) = F(Pop_{tkl}^{l=1...M,k=1...N}, i) \tag{2.5}$$

F is a function depending on the fitness function of the selected chromosome i, and the fitness of all the chromosomes k in the population. Even though chromosomes that are closer to the optimal solution have a smaller probability for mutation, their mutation most probably is going to diverge them from the optima. Thus, most of the genes are in a good position and any random modification is going to be mostly harmful. Hence uniformly distributed mutation over the genes

DOI:10.15774/PPKE.ITK.2023.005

is not the best option; assume we have an almost perfect solution (nine genes are perfect and one is bad), we have a 9/10 chance with uniform mutation to make this instance worse.

To tackle this problem, we have designated a different probability operator for each gene in a chromosome which can only be possible in partially solvable problems. To think about it, the mutation happens at the gene level where we choose one or two genes randomly and then we change their values. Thus, a gene which is in a good position should be less prone to mutation. The simplest model for gene level mutation is locus mutation as in equation (2.6) where all generations and chromosomes have the same mutation rate but each gene has a different mutation rate that corresponds with the other genes.

$$PM(Pop_{tij}) = F(Pop_{til}^{l=1...M}, j) \tag{2.6}$$

F is a function depending on the fitness function of the selected gene l. Once the mutation rate is set, the probability of mutation is the same for every chromosome at all times ($PM(Pop_{til}) = PM(Pop_{tkl}) \forall t, i, k$). To grasp the concept before diving into details, we can simply state that measuring the fitness of each gene in a partially solvable problem will deduce a unique customized distribution for each chromosome yielding a gene-level mutation. As an advantage of our approach, we can combine locus mutation with any of the other proposed methods. Returning to the first and most generalized equation (2.1), we can use locus mutation with an adaptive individual level where each chromosome and each gene has an individual mutation rate which may give us a better result but certainly will make the whole process slower and resource consuming. We have only focused on the simplest version of our novel idea which is locus mutation.

Algorithm 2 depicts GA with locus mutation. All parameters remain the same as in the original Algorithm 1 setting Pow parameter to one, but we do have a newly introduced gene-level mutation ($M_g()$) which depends on partial fitness ($PartialValues$). Although in our experiments we will always set Pow to one only focusing on the effect of locus mutation without taking into account Pow parameter, a detailed investigation Section 2.5.4 will be conducted illustrating the advantage which can be garnered using Pow parameter.

DOI:10.15774/PPKE.ITK.2023.005

Algorithm 2: Genetic algorithm main steps

Parameters: $PopSize, MutRate, IterNum, Pow = 1$ **Result:** $Optimum$
$Pop =$ population initialization
for $i \leftarrow 0$ **to** $IterNum$ **do**
    $Values, PartialValues = F(Pop)$
    $Pop = S(Pop, Values)$
    $Pop = C(Pop)$
    $Pop = M_g(Pop, MutRate, PartialValues, Pow)$
end

In a partially solvable problem, partial fitness can be calculated leading to mutation with probabilistic gene selection. In a problem representation, one could identify parts that are good and parts that are bad. A good gene should be changed less frequently, a worse element should be changed more often exploring further regions away from local optima. To illustrate the importance of $PartialValues$ in calculating the probabilistic mutation factor of each gene, we will discuss the partial solution of the 8 queens problem as it has been depicted in figure 2.1. In the 8 queens problem, the chromosome has eight genes which refer to the number of the row, while the index of each gene refers to the column e.g. figure 2.1 illustrates a candidate solution $[1, 3, 2, 5, 7, 4, 6, 8]$ where the index of the vector represents the column number while the value of the vector represents the row number. A chromosome with zero queens hitting each other is optimal. Intuitively, the fitness function calculates the number of hits, and our goal is to minimize the fitness function which we can also call a loss function. In the example depicted in figure 2.1, the loss is four (Calculating hitting pairs only once.) where queens 1, 2, 3, 4, 7, 8 are hitting 8, 3, and 4, 2 and 7, 2, 3, 1 respectively. On the other hand, partial fitness will give a different loss for each gene representing the number of queens hitting the current queen. In our example, partial values are [12210011] where for example queen number three is hitting two other queens (2 and 7) which means it is a bad queen and a high mutation rate should be assigned to it and likewise for queen number two. Whereas, queens five and six are not hitting any other queens, meaning that low mutation rates should be assigned to them.

Instead of using uniform distribution as in the traditional algorithm, we are using a probabilistic function conveying the information about the fitness of each

DOI:10.15774/PPKE.ITK.2023.005

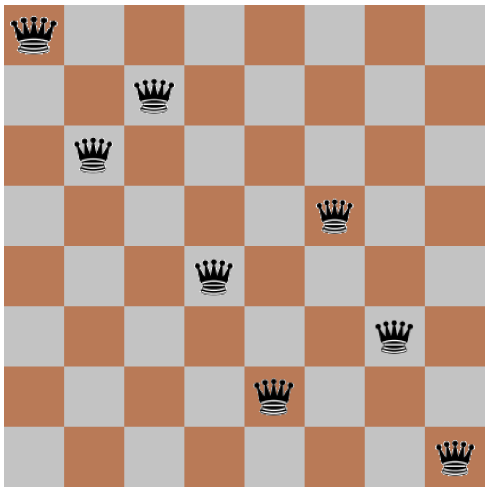

Figure 2.1: Potential chromosome for 8-Queens Problem where queen 1 is hitting queen 8, and two queens (2,3) are hitting each other and also hitting two other queens (4,7) yielding a loss of 4 where the four hitting pairs are ($[1, 8]$, $[2, 3]$, $[2, 4]$, $[3, 7]$).

gene. The mutating rate of each gene does not only depend on its partial value, but it is also proportional to other genes' partial value, also every gene has a minimal mutation rate, this ensures that even a gene with zero hitting queens, will have a non-zero mutation probability.

2.4 Heuristically Partially Solvable Problems with Unknown Optimum

As we will see in section 2.5.1, locus mutation works well with a partially solvable problem outperforming the traditional approach using gene-level information. Although Locus mutation is only applicable for partially solvable problems e.g., N-Queens problem [41], the heuristic partial solution can be sufficient which can only be inferred with a comprehensive understanding of the problem. One of the most elusive problems is the traveling salesman problem (finding the shortest route to visit a set of cities), where the optimal solution is undefined making the optimization process interminable, and a heuristic threshold has to be used. In the N-Queens problem, calculating the partial solution was a straightforward process which is the number of queens hitting the current gene taking all the other

DOI:10.15774/PPKE.ITK.2023.005

genes into consideration. Having us doing so in the traveling salesman problem (TSP) [42] requires a modification signifying the distance between the current gene (city) and the next gene with contrast to its distance with the other genes. Since traditionally the fitness of the entire chromosome, which we are trying to minimize, relies solely on the distance between each gene and its next neighbor without taking into account any other genes, we came up with a new idea which we will call **normalized comparative loss** to calculate the partial fitness of each gene taking into consideration all other genes.

An example has been depicted in figure 2.2 showing a simple example of the TSP problem with 10 cities. Each city has two indices, the first index indicates the order of the city in the candidate solution [63495110782] and the second one refers to the actual label of the city e.g., the first gene in our chromosome has the label $(1, 6)$ which means the traveler will start from this city then go to the city with the next index city number 3 and move sequentially until reaching city number 2. To explain the partial fitness value of a gene, we have used the same candidate chromosome but focused on the penultimate gene, gene $(9, 8)$, as in figure 2.3. This gene refers to the city number 8 which is located at the gene before the last gene in the candidate solution. The red line depicts the relevant connection between our gene and gene number two; the last gene in the current chromosome and the next neighboring gene. The gene which is the farthest away from our gene of interest has been linked to our gene with a green line, while the blue line portrays the closest gene. Using the three previously mentioned distances; the pertinent distance between the current gene and the next gene, the distance between the current gene and the gene which is farthest away from the current gene, and the distance between the current gene and the gene which is closest to the current gene; we can calculate a heuristic partial fitness as in equation (2.7) where i is the concerned gene while $MinDistance$ and $MaxDistance$ give us respectively the minimum and the maximum distance between gene i and the other genes.

$$PF_i = \frac{Distance(i, i+1) - MinDistance(i)}{MaxDistance(i) - MinDistance(i)} \tag{2.7}$$

Before moving to the results section demonstrating the effectiveness of locus mutation, we will substantiate the advantage of using locus mutation by applying the Wilcoxon test which is a non-parametric statistical test used to determine if

DOI:10.15774/PPKE.ITK.2023.005

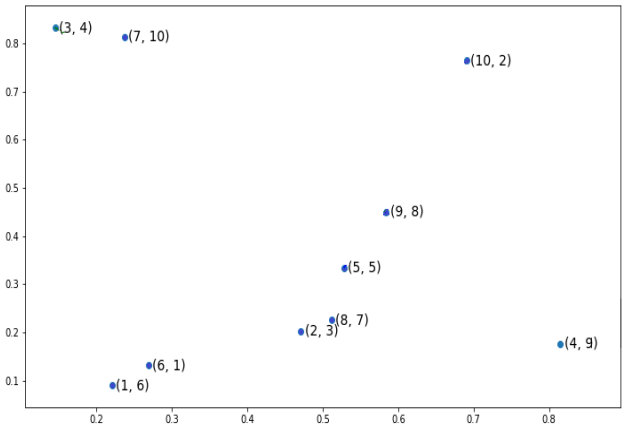

Figure 2.2: Traveling salesman problem with 10 cities, chromosomes. The vertices depict the cities where the first index refers to the position of the city inside the chromosome while the other index refers to the city label. An edge can be formed between every two sequential cities to show the path that the traveling salesman should take.

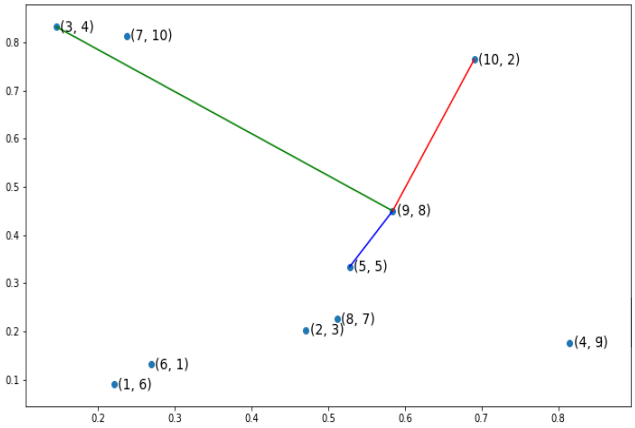

Figure 2.3: This figure depicts the pertinent distances of a specific gene (gene (9,8)) for a Traveling salesman problem with 10 cities chromosome. The red vertex depicts the pertinent path between our gene and the next gene. The green and blue vertices link the gene of interest with the farthest and closest gene in respect.

DOI:10.15774/PPKE.ITK.2023.005

two sets of distributions are different from each other in a statistically significant manner. We investigated a TSP instance with 48 cities, 1000 chromosomes, and 20 generations. We used the same initial population for baseline and locus mutation then stored the evolved two populations after the 20th generation. Figure 2.4 demonstrates the distribution of the fitness of the chromosomes for the initial population, the 20th baseline population, and the 20th locus population. Figure 2.4 illustrates visually the benefit of locus mutation where we can see that the distribution of the fitness of the 20th generation using locus mutation is closer to zero with a smaller mean value. We applied Wilcoxon matched pairs test to the fitness of the evolved baseline and locus population obtaining a very small p value, 4.25×10^{-12}.

We can conclude from the minute p value that the two distributions have different medians and reject the idea that the difference is due to chance.

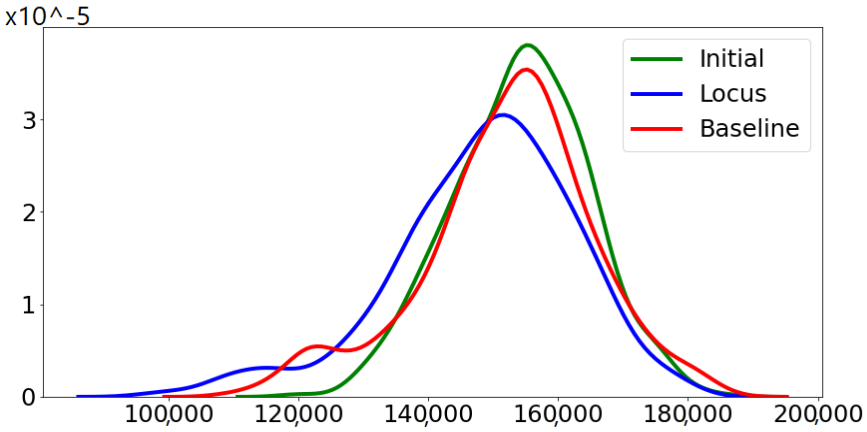

Figure 2.4: The figure shows the distribution of the fitness of the chromosomes in a generation in three different cases, initial generation, 20th generation using baseline mutation and the 20th generation using locus mutation. Locus mutation is not just attaining better chromosomes, and smaller fitness, but also moving the entire distribution closer to zero.

DOI:10.15774/PPKE.ITK.2023.005

2.5 Results

2.5.1 N-Queens Problem

To validate our hypotheses, a detailed comparison with the traditional mutation has been investigated proving a superior performance with a different set of parameters as in figure 2.5. The results have been averaged out for 50 different experiments. The best solution has been selected out of 5 unique mutation rate values $[0.01, 0.1, 0.6, 0.3, 0.9]$ which are distributed over the entire parameter space search. All experiments have been conducted using the traditionally applied crossover method as well. For the sake of reproducibility, you can find our codes online (`https://github.com/Al-Afandi/Adaptive-Gene-Level-Mutation`) alongside the chosen investigated parameters.

Figure 2.5 and Table 2.1 demonstrate a quantitative and qualitative superiority of our approach always leading to a better solution with a reasonable margin. Our approach could almost always solve the N-Queens problem with a population of 32 while the traditional approach could never reach the optimal solution. With a population of 128, our approach is two to three times better than the traditional approach in terms of the final loss. Even with a huge population of 256 queens, our approach is 1.6 times better in terms of fitness function.

To demonstrate the effectiveness of our new approach with comparison to other recent attempts working on improving mutation operator, we have compared our mutation method with traditional and individual level adaptive mutation [43] using the same set of parameters which is 64 Queens, $PopSize = 400$, $IterNum = 20$ and $Pow = 1$ as depicted in figure 2.6. For individual adaptive level mutation, we have investigated 30 different ranges of $MutRate$ while using a static mutation rate (0.5 is the middle of all ranges) for the other two methods. The ranges have been centered around 0.5 and varied from a very small range $[0.485, 0.515]$ to a very large one $[0.05, 0.95]$. Although the results of the two other methods, traditional and adaptive approaches, have been obtained by selecting the best solution and averaging out ten different experiments while only calculating the average results of our method, our approach has a superior performance and a faster convergence.

DOI:10.15774/PPKE.ITK.2023.005

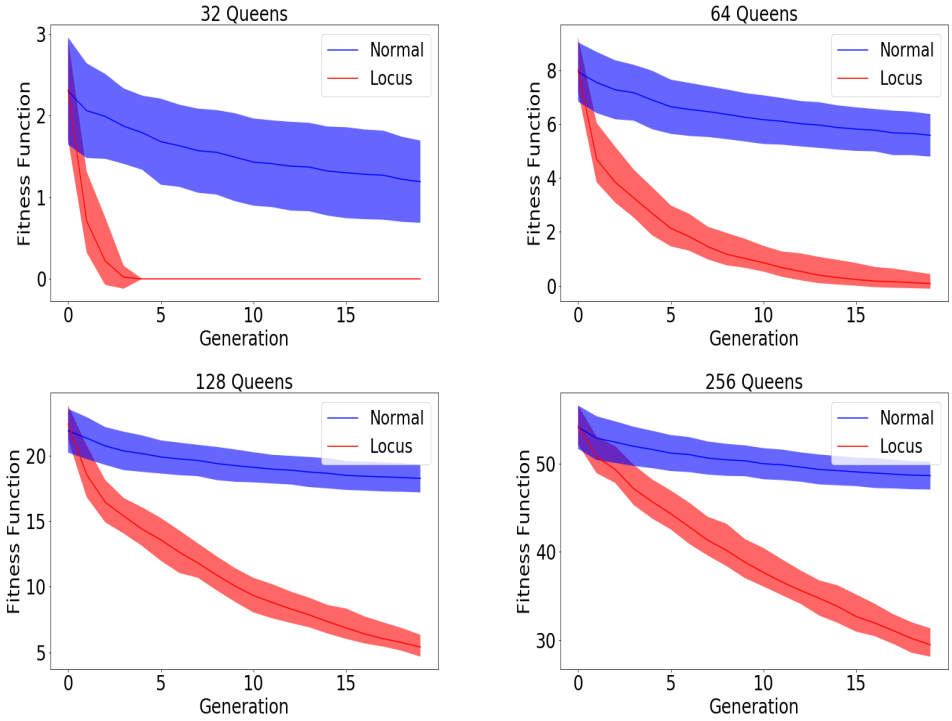

Figure 2.5: It depicts a comparison between Locus mutation and traditional mutation with different sets of parameters. The figures depict the N-Queens problem with 32, 64, 128, and 256 queens where the results have been averaged out with two different generation size $[200, 400]$, $[400, 600]$, $[600, 800]$ and $[800, 1000]$ respectively. We only selected the best solution out of these five different values of mutation rate $[0.01, 0.1, 0.6, 0.3, 0.9]$. The center of the curve is the expected value while the range visualizes the standard deviation. All the experiments have been repeated 50 times and then averaged out. We can notice that the number of hitting queens is escalating when we increase the number of queens.

DOI:10.15774/PPKE.ITK.2023.005

Table 2.1: N-Queens optimal solution, the minimum number of hitting queens, after 20 generations. Two different population sizes $PopSize$, 200 and 400, have been investigated with 50 different repetitions. All the runs have been averaged out.

IterNum	MutRate	Number of Hits	
		Baseline	Locus
32	0.01	1.78	**0.01**
32	0.1	2.06	**0.27**
32	0.3	1.7	**0**
32	0.6	1.52	**0**
32	0.9	1.64	**0**
64	0.01	6.6	**1.**
64	0.1	7.16	**2.33**
64	0.3	6.64	**0.71**
64	0.6	6.38	**0.35**
64	0.9	6.52	**0.24**
128	0.01	19.82	**7.98**
128	0.1	20.28	**11.48**
128	0.3	19.57	**7.17**
128	0.6	19.24	**6.19**
128	0.9	19.89	**5.91**
256	0.01	51.04	**33.33**
256	0.1	51.8	**38.8**
256	0.3	50.37	**32.07**
256	0.6	50.18	**30.75**
256	0.9	51.17	**31.06**

DOI:10.15774/PPKE.ITK.2023.005

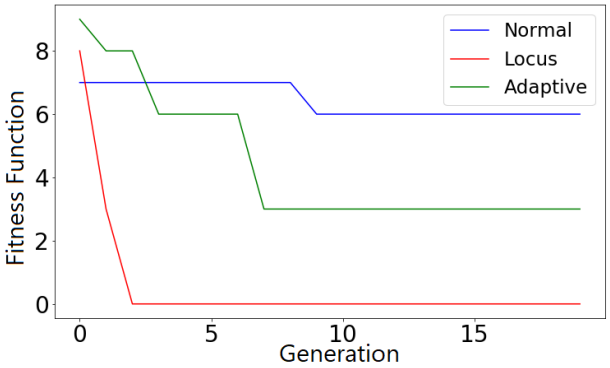

Figure 2.6: The fitness value of 64-Queens problem as a function of the number of generations comparing locus mutation with traditional and adaptive mutation. For adaptive mutation, We have averaged out 30 different ranges and used the center of each range for traditional and locus mutation. All the experiments have been repeated 10 times with the same set of parameters. Although we have used the mean solution for locus mutation, we have selected the best solution for adaptive and traditional mutation.

We have shown one comparison between locus mutation and another mutation method; figure 2.6, but the advantage of our method is that it is compatible with any other generally applied mutation e.g., population level mutation and individual-level mutation; to the extent where you can obtain statistic information from generations, populations, chromosomes and even genes (Locus mutation) as in equation (2.1).

2.5.2 Traveling salesman problems

Apart from the results on the N-Queens problem, we have also investigated another commonly examined problem, the TSP problem. A detailed comparison with the traditional mutation has been investigated manifesting a superior performance with a big set of different parameters as in Figure 2.7 and Table 2.2. The best solutions have been selected out of 5 unique mutation rate values $[0.01, 0.1, 0.6, 0.3, and 0.9]$ expanding through the parameter space. All the experiments have been repeated 100 times and then averaged out. With every repeat, TSP problem (cities location, weights, and initial population) will be created automatically by randomly placing N cities on a small grid.

DOI:10.15774/PPKE.ITK.2023.005

Figure 2.7 and Table 2.2 demonstrate a quantitative and qualitative superiority of our approach always leading to a smaller distance. Having a bigger and bigger population increases the gap between the two approaches e.g., the gap was maximum 1 at the beginning with 32 cities to reach 5 at the end with 254 cities.

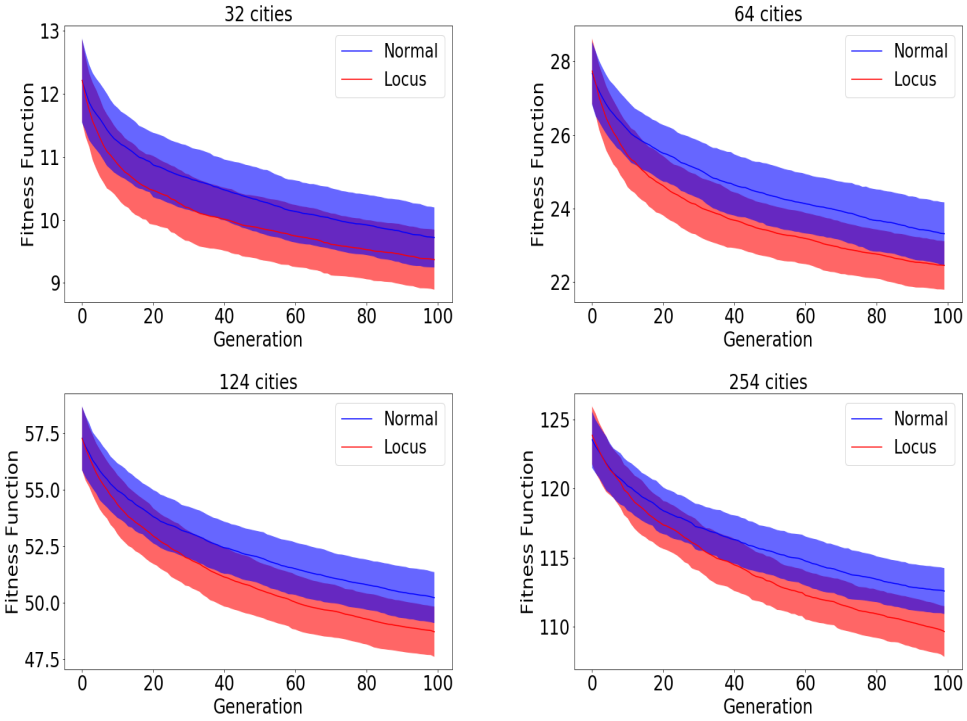

Figure 2.7: Comparison between Locus mutation and traditional mutation with different sets of parameters. The figures depict TSP problem with 32, 64, 124 and 254 cites. We averaged out the runs with two different generation size $[400, 600]$, $[400, 600]$, $[600, 1000]$ and only 1000 for 254 cities constellation. We only selected the best solution out of these five different values of mutation rate $[0.01, 0.1, 0.6, 0.3 and 0.9]$. The center of each curve is the expected value while the range visualizes the standard deviation. All the experiments have been repeated 100 times and then averaged out.

2.5.3 Using Locus Mutation with Other Heuristic Algorithms

Mutation can be a substantial operation for other heuristic algorithms as well e.g., simulated annealing [44], variable neighborhood search [45], tabu search [46] and

DOI:10.15774/PPKE.ITK.2023.005

Table 2.2: TSP optimal solution after 100 generations. Two different population sizes, 200 and 400, have been investigated with 100 different repetitions. All the runs have been averaged out. $IterNum$ is the number of cities (population number), while $MutRate$ is the mutation factor.

IterNum	MutRate	Distance traveled	
		Baseline	Locus
32	0.01	8.486	**8.287**
32	0.1	8.574	**8.341**
32	0.3	8.62	**8.30**
32	0.6	8.997	**7.93**
32	0.9	8.915	**8.36**
64	0.01	22.281	**21.470**
64	0.1	**20.293**	21.265
64	0.3	**21.038**	20.859
64	0.6	21.627	**21.002**
64	0.9	22.579	**20.714**
124	0.01	48.797	**45.380**
124	0.1	46.679	**46.256**
124	0.3	47.830	**46.338**
124	0.6	47.381	**46.320**
124	0.9	49.481	**45.774**
254	0.01	110.328	**102.464**
254	0.1	108.453	**107.808**
254	0.3	109.147	**107.704**
254	0.6	110.102	**106.701**
254	0.9	110.946	**105.642**

DOI:10.15774/PPKE.ITK.2023.005

Hill climbing [47]. Locus mutation works well with genetic algorithm solving the TSP problem giving better results than the baseline, as we illustrated in earlier paragraphs. In the previous results, TSP instances were created by randomly sampling the interval $[0, 1]$ where these samples composed the cities coordinates. To further instantiate our results, we did investigate the TSP problem with genuine data provided from the TSPLIB dataset. We only investigated the instances which have less than 101 cities, 16 instances. We investigated the efficiency of locus mutation in two other heuristic algorithms (simulated annealing (SA) and variable neighborhood search (VNS)). Table 2.3 demonstrates the conspicuous advantage of using locus mutation. The experiments for the three algorithms were repeated 10 times and with two different setups (one run with simple parameters and another one with complex parameters). In most cases, using locus mutation gives a better result, and the best solutions were obtained from VNS algorithm with locus mutation. Although in most instances, we didn't reach the optimal solution, We were only focusing on the benefit of using locus mutation with other algorithms, apart from GA, without thoroughly investigating other enhanced versions of the same algorithms.

2.5.4 Exploiting the Tuning of the Power Parameter

The importance of each loss (partial fitness) can be changed by using a power function which can raise the partial fitness vector to a power (Pow). Using a power function with $Pow = 0$, you get the uniform mutation back making our approach an extension of the original method.

For further illustration, we will go back to the previous example as in figure 2.1. Partial fitness, which is the main bulk calculating the mutation rate of each gene, equals [12210011]. Using the Pow parameter, we would have the following updated partial fitness values:

DOI:10.15774/PPKE.ITK.2023.005

Table 2.3: Comparison between baseline (Base) and locus (Loc) mutation for three different algorithms, genetic algorithm (GA), simulated annealing (SA), and Variable neighborhood search (VNS). We applied these algorithms on 16 different instances from the TSPLIB dataset. In most cases, locus mutation is enhancing the performance of the algorithms.

Instance	VNS Loc	VNS Base	SA Loc	SA Base	GA Loc	GA Base
att48	**38,074.95**	38,349.60	**66,162.13**	78,766.73	**125,478.95**	127,962.10
berlin52	**8633.02**	8718.89	**15,005.75**	16,852.92	**24,670.25**	25,154.85
burma14	**24.99**	25.56	27.11	**26.83**	**40.70**	44.78
eil51	**477.05**	487.21	**827.75**	951.21	**1375.58**	1410.37
eil76	**605.78**	626.56	**1387.55**	1570.88	**2167.07**	2209.15
kroA100	**25,923.15**	26,478.45	**90,299.24**	103,019.28	**148,699.96**	150,347.27
kroB100	**25,248.30**	25,437.91	**88,441.77**	102,262.19	**145,522.35**	148,050.07
kroC100	**24,983.77**	25,042.63	**88,651.16**	102,050.32	**146,813.87**	147,521.41
kroD100	**26,225.47**	26,276.29	**87,275.06**	100,001.28	**142,412.64**	142,597.61
kroE100	**24,608.56**	24,779.56	**90,021.99**	104,828.39	**148,325.58**	150,791.90
pr76	**129,505.66**	132,538.60	**315,049.50**	354,149.35	**491,017.72**	500,005.56
rat99	**1386.94**	1388.55	**4412.32**	5126.53	**7280.52**	7344.15
rd100	**9580.81**	9697.09	**30,877.39**	34,885.76	**49,093.79**	49,482.80
st70	**750.11**	769.31	**1902.85**	2161.98	**3091.04**	3149.35
ulysses16	**52.01**	55.33	**59.52**	61.09	**100.05**	101.36
ulysses22	**55.18**	55.66	**73.00**	75.17	**129.77**	133.62

DOI:10.15774/PPKE.ITK.2023.005

$$
\begin{aligned}
[1\,2\,2\,1\,0\,0\,1\,1]^0 &= [1\,1\,1\,1\,1\,1\,1\,1] \\
[1\,1\,1\,1\,1\,1\,1\,1] &\approx [0.12\,0.12\,0.12\,0.12\,0.12\,0.12\,0.12\,0.12]
\end{aligned}
$$

$$
\begin{aligned}
[1\,2\,2\,1\,0\,0\,1\,1]^1 &= [1\,2\,2\,1\,0\,0\,1\,1] \\
[1\,2\,2\,1\,0\,0\,1\,1] &\approx [0.12\,0.25\,0.25\,0.12\,0.00\,0.00\,0.12\,0.12]
\end{aligned}
$$

$$
\begin{aligned}
[1\,2\,2\,1\,0\,0\,1\,1]^2 &= [1\,4\,4\,1\,0\,0\,1\,1] \\
[1\,4\,4\,1\,0\,0\,1\,1] &\approx [0.08\,0.33\,0.33\,0.08\,0.00\,0.00\,0.08\,0.08]
\end{aligned}
$$

$$
\begin{aligned}
[1\,2\,2\,1\,0\,0\,1\,1]^3 &= [1\,8\,8\,1\,0\,0\,1\,1] \\
[1\,8\,8\,1\,0\,0\,1\,1] &\approx [0.05\,0.40\,0.40\,0.05\,0.00\,0.00\,0.05\,0.05]
\end{aligned}
$$

$$
\begin{aligned}
[1\,2\,2\,1\,0\,0\,1\,1]^\infty &= [0\,1\,1\,0\,0\,0\,0\,0] \\
[0\,1\,1\,0\,0\,0\,0\,0] &\approx [0.00\,0.50\,0.50\,0.00\,0.00\,0.00\,0.00\,0.00]
\end{aligned}
$$

We can notice that when $Pow = 0$ we will get back to the uniform distribution where all genes have the same probability for a mutation. When $Pow = 1$, queens 2 and 3 have a bigger chance for a mutation. Increasing the value of Pow drastically decreases the comparatively good genes' probability for a mutation. When $Pow = \infty$, All probabilities are going to be zero except the genes with the worst partial fitness. As we mentioned earlier, partial fitness will always have a non-zero mutation operator; thus in practice, we will add a low mutation operator for every gene e.g., 0.001.

The effectiveness of the Pow parameter is illustrated in figure 2.8 depicting the fitness value (the number of hits) of the best optimal candidate solution ($Yaxis$) with regards to Parameter Pow ($Xaxis$). The figure starts with uniform distribution; the traditional approach; then depicts the optimal solution using the default value $Pow = 1$ and end up with ℓ_∞ norm which will deterministically select

DOI:10.15774/PPKE.ITK.2023.005

the worst genes for mutation. The best result with the same set of parameters, which have been chosen arbitrarily, can be obtained with $Pow = 2$.

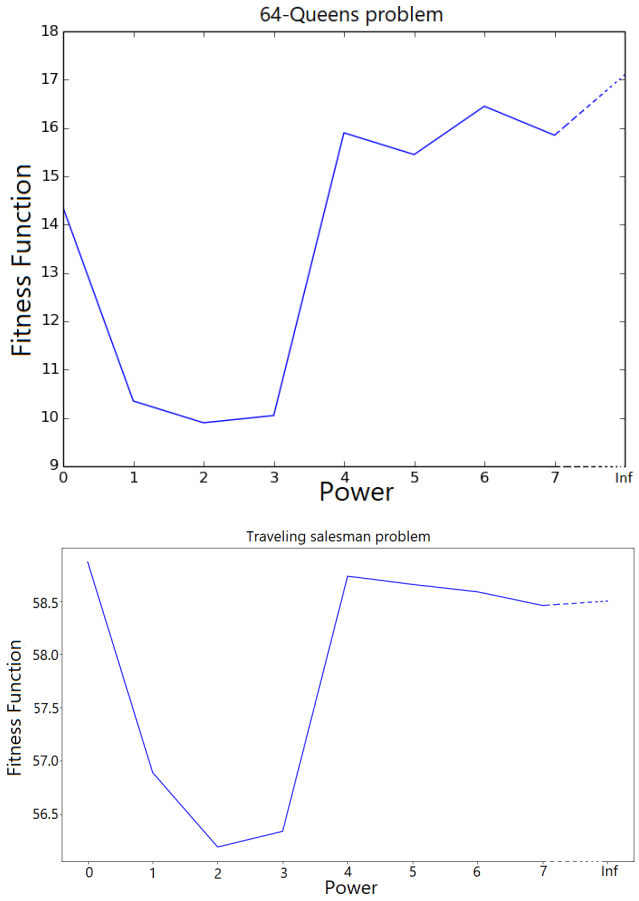

Figure 2.8: The fitness value of the best optimal solution for the 64-Queens problem and TSP problem respectively as a function of Powers Pow where we averaged out ten runs. It starts with uniform distribution then uses a logarithmic scale of Pow, and ends up with the L-infinite norm. We have achieved similar results with two separate problems.

Regarding the N-Queens problem, the experiments have been done with 64 Queens, $PopSize = 200$, $IterNum = 20$, $MutRate = 0.3$, $MinGeneMutRate = 0.1$, and a logarithmic scale of Pow values where $MinGeneMutRate$ is the lower bound of gene mutation operator. We have also obtained similar results investigating the Pow parameter on the TSP problem using normalized comparative loss, as in figure 2.8, where we have arbitrarily chosen a set of parameters; 128 cities,

DOI:10.15774/PPKE.ITK.2023.005

400 population size, and 30 generations. All experiments have been repeated ten times, and the fitness functions at the last generation have been aggregated. $Pow = 2$ can give us the best result for the two problems. Locus mutation works very well relying on the selfishness of each gene where each gene wants to mutate craving for perfection i.e., in the TSP problem, each gene is eager to be connected with the closest city. The idea of the Selfish gene is manifested in nature [48] leading to prosperity. Although this seemingly would lead to local optimum, sometimes the interest of individuals meets the interest of the population, and the interest of genes meets the interest of individuals where all genes are striving for excellence. We can even escalate the selfishness of the genes by increasing the power parameter Pow.

2.5.5 Running Time Comparison

Using this heuristic method, calculating the partial fitness and fulfilling the only prerequisite for locus mutation, requires some additional computations. $MinDistance$, $MaxDistance$, $Distance$, and even the denominator in Equation (2.7) are only vectors or matrices which can be pre-calculated once, but applying locus mutation will require $N * K$ operations (the number of genes multiplied by the size of the population). This big number of operations is needed because locus mutation gives each gene a distinct mutation rate, which corresponds with the normalized, comparative, and partial loss, taking other partial fitnesses into account. We have only investigated TSP extra time consumption due to the fact that for the N-Queens problem, beside the $N * K$ operations, no extra calculations are needed. According to our extensive experiments with a vast set of parameters, As in Table 2.4, Our approach can be in maximum two times slower than the traditional approach, but it will saturate with a better solution as we can see in figure 2.9 where our solution converged to the optimal solution but the traditional approach did saturate before approaching the optimum which we calculate using brute force method searching through each and every possible combination. To demonstrate the performance advantage of our approach having the same wall time, as in figure 2.10, we ran the traditional approach for two times more generations, 600 generations for the traditional approach and 300

DOI:10.15774/PPKE.ITK.2023.005

generations for locus mutation, proving that on the long run our approach will saturate to a better solution consuming the same time.

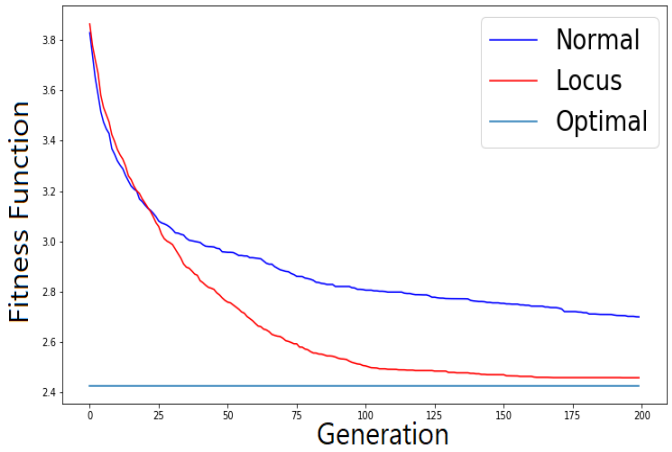

Figure 2.9: This figure depicts the TSP Problem with 10 cities manifesting the speed and the ability of our approach to nearly reach the optimal solution in comparison with the traditional approach. All the experiments have been conducted with a 200 population size, 200 generations, and a 0.5 mutation rate. The optimal solution has been obtained using a brute force algorithm. The experiments were repeated 100 times.

I have illustrated that using a gene-dependent local mutation operator where every gene has a different mutation rate induced from a heuristic and partial fitness function will speed up the convergence of the algorithm and yield a more accurate final solution. We have investigated two common problems, the traveling salesman problem (TSP) and N-Queens problem. In the case of the N-Queens problem, Locus mutation has produced better solutions in all cases, regardless of the investigated parameters. Even with a big population number of 254, locus mutation yields a 1.5 times lower error than its traditional counterpart. Similar results were obtained using locus mutation for the TSP problem where our approach has always surpassed the baseline solution.

2.5.6 Thesis Point 1

According to the results of this chapter which were published in Algorithms journal 2021 [49], I formulated my first thesis point as the following: I have introduced a

DOI:10.15774/PPKE.ITK.2023.005

Table 2.4: Time comparison between locus and baseline mutation where PN is the number of cites, PS is the population size, GN is the number of generations and MR is the mutation rate. TSP timing gives us the time consumption for each algorithm using the specified parameters. Ratio gives us the speed rate, speed advantage, of the original approach.

PN	PS	GN	MR	TSP Timing		Ratio
				Baseline	Locus	
32	200	25	0.01	0.6297	0.9822	1.5598
32	200	25	0.5	0.8174	1.243	1.5207
32	200	25	0.9	2.3387	3.3526	1.4335
32	200	50	0.01	1.2445	1.9482	1.5655
32	200	50	0.5	1.6238	2.4778	1.526
32	200	50	0.9	4.6735	6.72	1.4379
32	400	25	0.01	1.4315	2.1328	1.4899
32	400	25	0.5	1.8039	2.6621	1.4757
32	400	25	0.9	4.838	6.8533	1.4166
32	400	50	0.01	2.9027	4.3041	1.4828
32	400	50	0.5	3.6441	5.3463	1.4671
32	400	50	0.9	9.7452	13.8431	1.4205
64	200	25	0.01	1.0071	1.7144	1.7023
64	200	25	0.5	1.1947	1.9664	1.6459
64	200	25	0.9	2.7421	4.0509	1.4773
64	200	50	0.01	2.0139	3.3967	1.6866
64	200	50	0.5	2.385	3.9082	1.6387
64	200	50	0.9	5.4714	8.1485	1.4893
64	400	25	0.01	2.2131	3.5941	1.624
64	400	25	0.5	2.5717	4.1369	1.6086
64	400	25	0.9	5.7171	8.3777	1.4654
64	400	50	0.01	4.379	7.1742	1.6383
64	400	50	0.5	5.1617	8.1775	1.5843
64	400	50	0.9	11.3358	16.5914	1.4636

DOI:10.15774/PPKE.ITK.2023.005

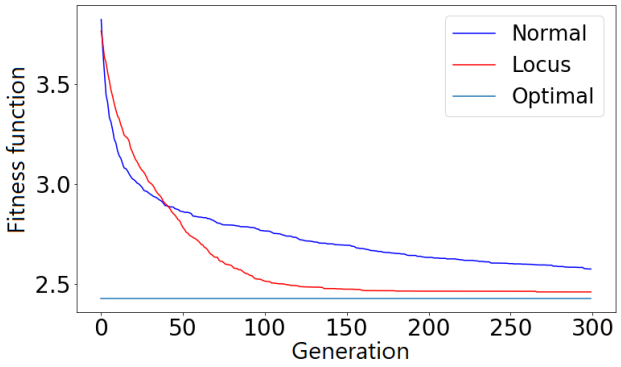

Figure 2.10: This figure depicts the TSP Problem with 10 cities manifesting the ability of our approach to surpass the traditional approach consuming the same time. We did run the traditional approach for two times more generations e.g. when axis x equals 25 generations for locus mutation (as in the figure), it equals 25 * 2 generations for traditional mutation. All the experiments have been conducted with a 200 population size and 0.5 mutation rate. We run the traditional approach for 600 generations, while we run locus mutation only for 300 generations. The optimal solution has been obtained using a brute force algorithm and the experiments were repeated 100 times.

new mutation variant in genetic algorithm with an additional parameter that can determine a mutation factor on the individual gene. I have demonstrated that with the proper tuning of the local mutation parameter the final accuracy of the investigated algorithmic setups was increased by 64% on the 256-Queens problem and 7% on the TSP problems with 254 cities.

DOI:10.15774/PPKE.ITK.2023.005

DOI:10.15774/PPKE.ITK.2023.005

Chapter 3

Adversarial attack retrieval

3.1 Introduction

Intellectual property theft and adversarial attacks are practical problems rooted in theoretical properties which were only analyzed recently after the big advancement of neural networks over many applications e.g., image segmentation [1] and self-driving cars [2]. The two problems can avert companies from selecting neural networks as a secure and safe solution to their many products.

Recently, deep learning has emerged as a cornerstone for safety and security-critical applications, like autonomous vehicles [6] and malware detection [3]. The authors in [50] present a case of a malevolent attacker who can exploit the high-dimensional inputs by slightly modifying the pixels approaching the border of the high-dimensional geometrical manifold of the original class [51]. Although the attacks can be very strong, reaching more than 90% confidence for misclassified classes, the modification which has been applied to the original image is imperceptible to the naked eye. Minor perturbation over the entire image is the first adversarial attack that has been introduced by Goodfellow [52]. Many threatening results have surfaced urging the research community to find some defense mechanisms, e.g., [53] demonstrates a universal perturbation fooling a classifier on any image, [54] showed the possibility of fooling a classifier with a 3D printed object.

The most commonly applied defenses against adversarial attacks depend on one of three approaches: adversarial training [55], modifying the network [56] or a

DOI:10.15774/PPKE.ITK.2023.005

detection-based approach [8]. Adversarial training slows down the training and it has also been demonstrated by [53] that you can form an adversarial attack against a network that has been trained with adversarial training. Most defenses which rely on modifying the network are too computationally infeasible to be applied on larger networks or they only work against specific attacks [57]. Detection-based approaches lack any notion of security in safety and time-critical applications where an immediate decision has to be taken without any delay, e.g., object detection in self-driving cars.

Detection of adversarial attacks can be a good first step, but, on its own, it is not enough to ensure safety in practical applications since detection will only render the autonomous system in complete doubt, preventing the AI from making a sound and reliable decision. The ultimate safe solution for adversarial attacks, in the case of accurate detection, is recovering the original class. We will introduce a novel algorithm restoring the original class of the attacked image. We will demonstrate the effectiveness of our algorithm on three different datasets MNIST, CIFAR10 and 10 randomly selected classes from ImageNet.

3.2 Adversarial Attacks

The term *adversarial example* was coined by [50], where attacks on neural networks trained for image classification were generated via a very low-intensity additive noise, completely unobtrusive to the human eyes as in figure 3.1. An adversarial example is a misclassified sample that has been modified with unnoticeable perturbations drastically changing the response of a network.

Figure 3.1: The figure illustrates an adversarial attack with a very low-intensity perturbation entirely unperceivable to the naked eye.

DOI:10.15774/PPKE.ITK.2023.005

The eminent high risk of adversarial attacks urged the science community to thoroughly investigate any potential attack enlarging the concept of adversarial attack. Three kind of attacks have been investigated (evasion attacks, poisoning attacks, and exploratory attacks) where the first published adversarial attack [50] belongs to the first category. Evasion attacks are the most common attack in an adversarial setting where malicious samples are created to fool the network during the testing phase, causing a misclassification [52, 58]. Poisoning attacks [59] contaminate the training dataset by injecting meticulously contrived samples compromising the training procedure. Exploratory attacks [60] are used when we only have a black-box access to the model and, thus, the attacker tries to collect as much knowledge as possible about the distribution of the training dataset and the network response which can be used to build a network that is capable of building adversarial samples. We will concentrate on the widespread white-box gradient-based attacks which we will investigate in this chapter and briefly describe in this section.

3.2.1 Adversarial Attack algorithms

The first attacks [52] were implemented by calculating the sign of the gradient of the cost function (J) with respect to the input (x) and expected output (y), multiplied by a constant to scale the intensity of the noise (formally $\epsilon \text{sign} \nabla_x J(\theta, x, y)$, where $theta$ is the model parameter) where the calculated value is the perturbation which is added to the original sample creating the adversarial image. This method, which is called the fast gradient sign method (FGSM), allows for a much faster generation of attacks than the earlier algorithms due to the fewer number of iterations which are needed for a successful attack.

An extension to FGSM by [58] is to use not only the gradient's sign of the loss but rather a scaled version of the gradient's raw value ($\epsilon \nabla_x J(\theta, x, y)$). In the same paper a targeted attack and an iterative basic method were proposed. A targeted attack can be achieved by steering the perturbation to the opposite direction of the gradient of the loss function ($J(\theta, x, y_{target})$) maximizing the probability of a specific class (y_{target}). Iterative-based method attack, which is also called a projected gradient descent attack (PGD), is a targeted attack that can be

DOI:10.15774/PPKE.ITK.2023.005

applied N number of times with a small step size, taking into account the intensity boundary where the perturbations calculated at each iteration are accumulated, leading to a misclassification.

Another extension to the iterative version of FGSM by [61] was to incorporate momentum into the equation, hypothesizing that, similarly to regular optimization, momentum helps in stabilizing the direction of the update and avoiding poor local minima or maxima, narrow valleys and other non-convex patterns in the objective function's landscape.

Ref. [7] proposed DeepFool attack to calculate minimum norm adversarial perturbations, taking the image to the edge of the decision boundaries of the classifier. It is similar to the basic iterative method in the sense that it relies on gradients, but DeepFool normalizes the gradient by linearizing the boundaries of the manifold where the image resides.

Many other variations of PGD attack have been formulated enhancing the quality and the speed of the adversarial attack [62, 63]. A general overview of adversarial attacks, containing most of the previously mentioned methods, can be found in the following survey papers [57, 64].

3.2.2 Adversarial Attack Detection

Defenses against adversarial attacks are required to prevent security threats in the real-world applications of neural networks. Most defenses rely on one of the following three main approaches:

- Modifying the training process, e.g., adding adversarial samples (adversarial training) [7] or modifying the input before testing (decision making) [65, 66].
- Modifying the network, e.g., adding an extra masking layer before the last layer [67] and changing the loss function by penalizing the degree of variety per class in the output [56].
- Using external model as a detector, e.g., SafetyNet [8] and convolutional filter statistics detector [68].

DOI:10.15774/PPKE.ITK.2023.005

Detectors can be a reliable choice providing the highest accuracy reaching $\sim$ 85% [68] and preventing most security breaches. There are a lot of other detectors [69] [70] separating the clean image from the adversarial one by finding some distinguishable features and properties e.g. convolution filter statistics [68] and manifolds [69]. Although detectors are considered strong defenses against adversarial attacks, an attacker can cause a halt in the system hindering the achievement of any task. In a time-sensitive task e.g self-driving cars, where an on-the-spot decision has to be drawn, detectors are not sufficient and a retrieving approach has to be installed to recover the original class of the input.

3.3 Class Retrieval

Most non-detection defenses are vulnerable to counter-counter attacks [71], rendering potential exposure and keeping the system in a state of being without any functioning protective shield. Detection-based defenses, on the other hand, can be continuously updated but lack the ability to steer the decision-making process obstructing the installation of any safety measure. Thus, a recovery algorithm has to be employed after the detection of adversarial attacks, providing robustness and resilience.

Ref. [72] hypothesized that adversarial attacks exploit the edge of the decision boundary between classes, pushing the adversarial sample to the targeted class. Their idea stemmed from the speculation that training data will be pushed to the edge of the decision boundary once they are classified correctly. In [73, 74], the authors assume that the reason behind the adversarial vulnerability of neural networks is the highly positively curved decision boundary where the curvature is very intricate near the classes borders. The high dimensionality of neural networks creates convoluted borders between all the classes, making a targeted adversarial attack highly possible. Taking into account the complexity of the curvature of the decision boundary, we hypothesize that the distance between the adversarial sample and the original class's manifold in the feature space of the decision boundary is smaller than the distance between the adversarial sample and any other classes' manifolds and, hence, all the adversarial samples and their counter-attacks are in the vicinity of the original class manifold. We have implemented our

DOI:10.15774/PPKE.ITK.2023.005

idea, a class retrieval algorithm, on the notion of our former hypothesis to predict the original class by counter-attacking the adversarial samples, targeting every class, and then selecting the class with the minimum loss. What we can derive from our hypothesis is that during the counter-attack, it would be the easiest to transform back the attacked image to its original class since the attacked sample still contains many features belonging to the original class, and the manifold of the attacked class is the closest to the decision boundary of the original class, as we can observe in figure 3.2. The counter-attack can return the attacked sample to its original class easily since the adversarial sample is on the edge of the original class decision boundary. Due to the high dimensionality of the decision boundary curvature, there exists an intricate border between the manifold of each of the two randomly selected classes.

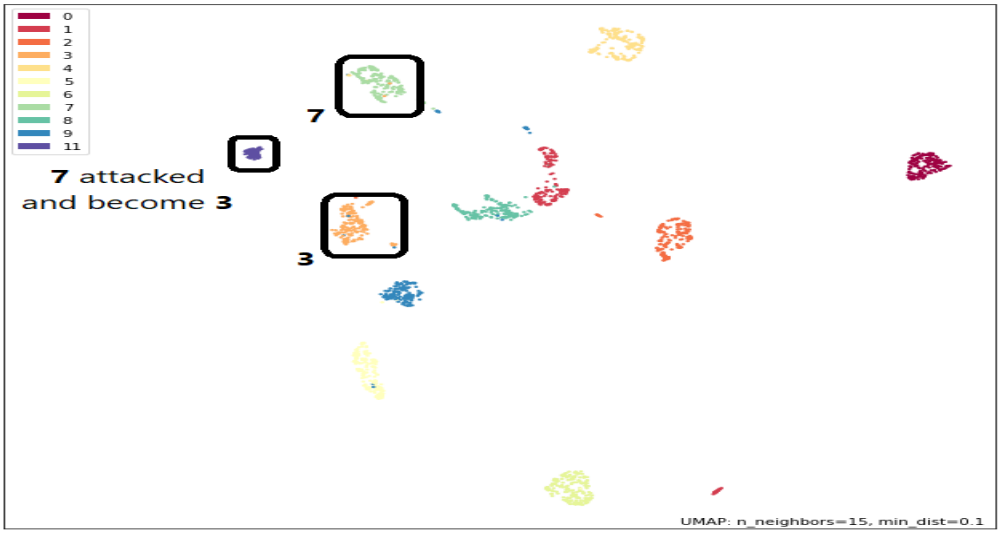

Figure 3.2: This figure displays a two-dimensional UMAP projection of the MNIST digits in the sklearn package with an additional 100 attacked samples which originally belonged to class 7 and were transformed to class 3 with the PGD algorithm. Each class is marked with a different color while the adversarially attacked samples are marked with purple. We can notice that the newly generated samples are between their original class and their new adversarial class, which led us to believe that going back to the original class would be the fastest. We generated similar figures for other classes as well, and the results were qualitatively the same in all cases.

To illustrate this hypothesis, we conducted experiments on the MNIST dataset,

DOI:10.15774/PPKE.ITK.2023.005

where 100 randomly selected samples from the same class were attacked (we considered them to be a separate class) and the two-dimensional positions of their manifolds were depicted using the UMAP algorithm. An example can be seen in figure 3.3, which confirms our assumptions that going back to the original class manifold can be achieved in fewer iterations than turning the sample to any other class, i.e., the cross entropy loss of a targeted class will be the smallest when targeting the original class.

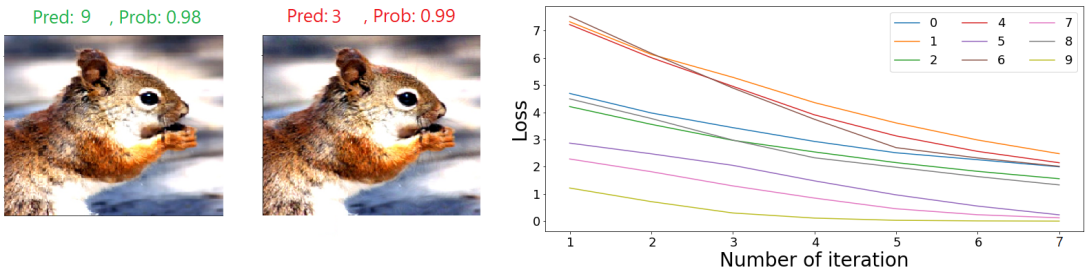

Figure 3.3: A showcase of our algorithm is illustrated in the figure using an image from the ImageNet dataset, where the first two images show the clean sample, squirrel, and its adversarial version, where *Pred* is the index of the predicted class and *Prob* is the confidence. The last image depicts the losses of the counter-adversarial attacks throughout 8 iterations, where the legends refer to the targeted classes. We can see that the index of the original label, nine, has the smallest loss since the first iteration.

Our class retrieval algorithm for a detected adversarial attack is presented in Algorithm 3 as a pseudo-code, and explained as a flowchart in figure 3.4. $AdvImg$ is the adversarial image that has been selected by an adversarial attack detector filtering any potential adversarial threat. The neural network prediction for the label of the adversarial image is $AdvLab$, which is a misclassification according to our detector. $NbClass$ is a fixed parameter representing the number of classes in our classification problem. We apply a counter-targeted attack using the $Attack()$ function where $NbIter$ is the number of iterations in the iterative adversarial attack and $Target$ is the targeted label. The loss function, $loss()$, calculates the cross entropy loss of the counter adversarial image, $ContAdvImg$, having $Target$ as a label. We exterminate the possibility of the adversarial label, $AdvLab$, being the original class by setting its loss to infinity. The original label, $OrigClass$, is

DOI:10.15774/PPKE.ITK.2023.005

the class with the minimum loss excluding the adversarial label where we used the *argmin* function to return the index of the smallest loss.

Algorithm 3: Class retrieval algorithm for a detected adversarial attack

Parameters: $NbIter, NbClass, AdvImg, AdvLab$ **Result:** $OrigClass$
$Losses = 0, Losses[AdvLab] = \infty$
for $Target : 0$ **to** $NbClass$ **do**
  if $Target! = AdvLab$ **then**
    $ContAdvImg = Attack(AdvImg, NbIter, Target)$
    $Losses[Target] = loss(ContAdvImg, Target)$
  end
end
$OrigClass = argmin(Losses)$

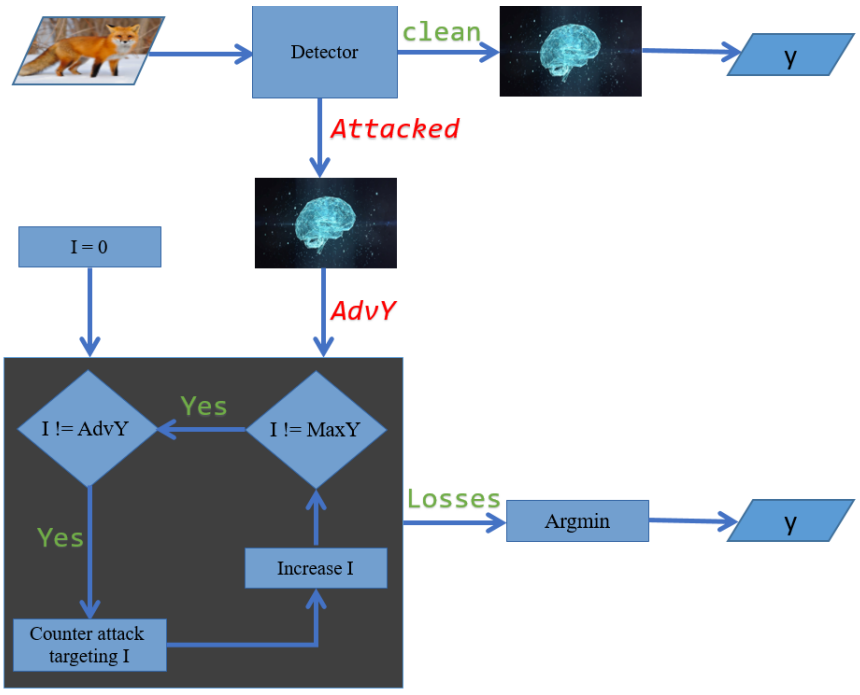

Figure 3.4: A flowchart explaining our class retrieval method starting from the input image until the output label where the *argmin* process returns the index of the minimum loss, $MaxY$ is the number of classes and $AdvY$ is the adversarial label.

To demonstrate the validity of our work, we assumed the existence of an optimal detector which can identify any adversarial attacks. We investigated

DOI:10.15774/PPKE.ITK.2023.005

four different adversarial attacks (projected gradient descent attack (PGD) [75], MPGD [61], Deepfool [7], TPGD [63], PGDDLR [62]; most of the attacks were adopted from Torchattacks library [76] while we used the codes of the original paper for the Deepfool attack) which were briefly explained in the previous section. Deepfool is not a targeted attack and the last two attacks are extended versions of the first attack; thus, we only used the first two previously mentioned attacks, PGD attack and PGD with momentum, as a counter adversarial attack. All the investigated adversarial attacks are white-box attacks, and this relies on the gradients to calculate the small perturbations fooling the classifier. For the sake of reproducibility, you can find our codes online `https://github.com/Al-Afandi/class_retrieval` (accessed on 7 June 2021), which include the chosen investigated parameters.

3.4 Results

3.4.1 MNIST

To validate our hypotheses, detailed experiments were conducted using the MNIST and other datasets, as we will see in the next paragraphs. MNIST is a commonly investigated dataset with ten classes containing 70 thousand images of handwritten digits with a 28×28 resolution. We investigated five different adversarial attacks (PGD, MPGD, Deepfool, TPGD and PGDDLR) to create a matrix of success rates with another two counter-attacks (PGD and MPGD). In each case, we attempted to retrieve the class of 1000 successfully attacked samples, which means 10,000 experiments altogether were conducted. The maximum distortion of the adversarial attack was set to 0.2. The number of iterations, $NbIter$, is 10 for the adversarial attacks insuring a successful attack, but we set it to 3 for the counter adversarial attack, illustrating the fast convergence to the original class.

AlexNet architecture was used throughout our experiments, providing a good baseline network where the average accuracy on clean samples is 96% (the original 28×28 images were rescaled to 224×224, ensuring the required input size). Figure 3.5 demonstrates the high accuracy of the class retrieval algorithm investigating the usage of two counter-attacks against the adversarial samples of five different

DOI:10.15774/PPKE.ITK.2023.005

attacks. On average, 72% of the attacked samples were correctly recovered, predicting their original class and averting misclassification.

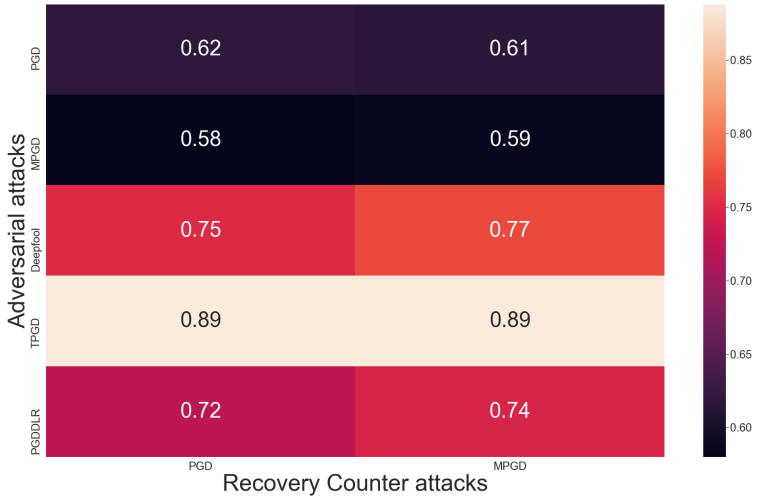

Figure 3.5: The figure illustrates the success rate of our class retrieval algorithm on the MNIST dataset, where each cell represents the accuracy of the retrieval in a specific setup, i.e., the algorithm used for the attack can be seen in the rows and the algorithm used for the counter-attack can be found in the columns.

3.4.2 CIFAR10

We investigated another simple but more intricate dataset, CIFAR10, to show the effectiveness of our novel approach. The same setup described in detail in the previous paragraph was used. The only parameters which were significantly modified were the adversarial attack maximum distortion and number of iterations $NbIter$, where we set the former to 0.05 and the latter to 5. We opted to use these smaller values in comparison to our setup with MNIST due to the faster and easier conversion to adversarial samples. Figure 3.6 shows the success rates using our class retrieval algorithm over the CIFAR10 dataset, engulfing ten different setups. We used the ResNet-18 architecture throughout our experiments, providing a good baseline with 93% accuracy in classifying clean samples. The overall average retrieval accuracy is 65%, which demonstrates the viability of our approach.

DOI:10.15774/PPKE.ITK.2023.005

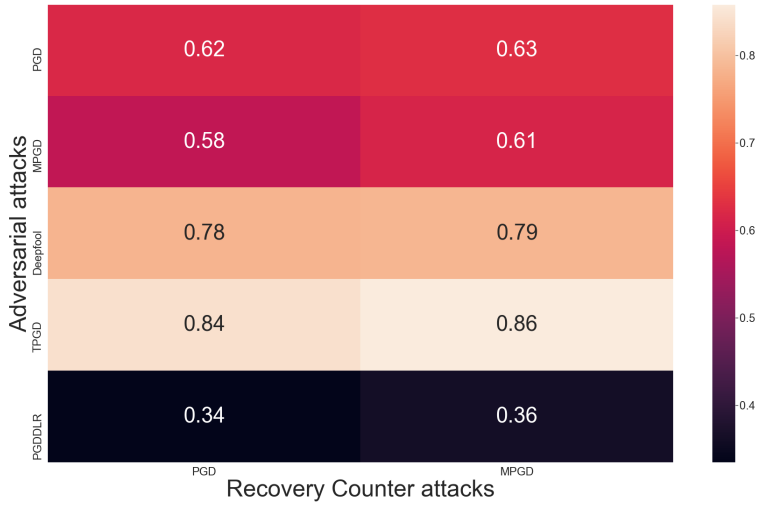

Figure 3.6: The figure illustrates class retrieval success rates on the CIFAR10 dataset, where each cell represents the accuracy of the retrieval in a specific setup, i.e., the algorithm used for the attack and the algorithm used for the counter-attack.

3.4.3 ImageNet

Our algorithm can be applied, in practice, with datasets that contain a limited number of classes N, because of the nature of the algorithm ($N-1$ number of counter attacks have to be made). To investigate complex and more practical datasets with high-resolution images, we have randomly selected 10 classes from ImageNet to execute similar experiments as in case of MNIST and CIFAR10. We did not use Deepfool as an adversarial attack because only targeted adversarial attacks can be used due to the fact that we can only target one of the ten selected classes. Altogether, 8000 attacks and retrievals were made and, to balance the effect of random class selection, we selected ten different classes for every 100 attacks. Throughout our investigation, we used the pretrained version of $Inception_{v3}$ architecture from the torchvision models library. The inception model has one thousand possible output classes, but our adversarial and counter-attacks were only targeting the randomly selected ten classes. We can see the success rate on ImageNet in figure 3.7, where each cell represents the accuracy of a specific attack and counter-attack investigating a thousand cases with 10 different random classes for every hundred trials. There was a 65% average accuracy of recovering

DOI:10.15774/PPKE.ITK.2023.005

eight thousand attacked images from ImageNet using our class retrieval.

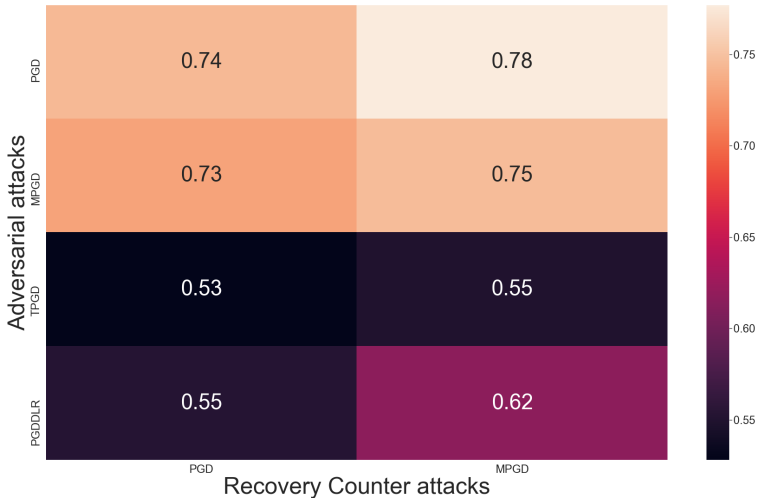

Figure 3.7: The figure depicts the success rate of our adversarial retrieval on the ImageNet dataset, where each cell represents the accuracy of the retrieval in a specific setup investigating 1000 cases selecting 10 random classes for every hundred trials.

3.4.4 Time burden analysis

During our experiments, we evaluated our algorithm over three datasets MNIST, CIFAR10, and ImageNet. MNIST and CIFAR datasets contain 10 classes and we investigated ImageNet with different classes always setting the number of classes to ten. In the case of a small number of classes, the time burden will be limited. You can find the average time consumption of the algorithm over 50 runs in the following table 3.1. The number of counter-attacks which is needed for the algorithm is the number of classes minus one and in each attack, we will only need two iterations to calculate the loss and conclude the original class. When using a device with a GPU the algorithm won't take more than a second.

3.4.5 Parameters Investigating

Our class retrieval algorithm has two important parameters, the number of iterations ($NbIter$) and the number of classes ($NbClass$). $NbIter$ parameter does not have much effect on the algorithm according to our measurements, and it gave

DOI:10.15774/PPKE.ITK.2023.005

Table 3.1: The time burden in seconds of the algorithm over three datasets MNIST, CIFAR10 and ImageNet. The experiments are conducted over two counter attacks and using a simple PC and a workstation.

Device	Counter attack	MNIST	CIFAR10	ImageNet
PC	PGD	0.049	23.9	30.32
	MPGD	0.05	23.8	30.21
Workstation	PGD	0.0287	0.1358	0.920
	MPGD	0.0282	0.1384	0.921

the same results after the first iteration, as we can observe in figure 3.3. $NbClass$ parameter is essential to the algorithm as we have to conduct $(NbClass - 1)$ counter attacks. Statistically, the accuracy will drop when having a large number of classes preventing a correct retrieval. We conducted an extensive investigation of the CIFAR10 dataset, evaluating the success rate of adversarial attacks having a different number of classes, from four to ten classes. We used AlexNet architecture to train a classifier for the randomly selected classes (4 to 10 classes), and we then calculated the average of the success rate of our class retrieval for each classifier (evaluating the retrieval of 10k adversarial attacks) to create a plot, figure 3.8, illustrating the relationship between the accuracy of our algorithm and $NbClass$ parameter. We can see that the algorithm is functioning effectively with a relatively high number of classes, but the accuracy will eventually decrease when processing a huge number of classes. Although a high number of classes is a limitation of our method due to time consumption and statistical probability, in most practical applications, only a few output classes are used.

Another parameter that can effect our method is the size of the input space. While low-resolution images, e.g., MNIST, are difficult to attack with small bounded unnoticeable perturbation, high-resolution images, e.g., ImageNet, are easy to attack because of their high dimensionality, rendering a complicated but vulnerable decision boundary curvature which can be easily compromised by slightly modifying the high number of pixels.

As we mentioned before, we assumed the existence of an optimal detector that can identify any adversarial attacks, and we implemented our experiments in light of this assumption. Unfortunately, practical detectors may result in false positive

DOI:10.15774/PPKE.ITK.2023.005

Figure 3.8: The figure depicts the relationship between the class retrieval algorithm and the number of classes parameter. It is clear that the algorithm is still working well with an expected drop in the accuracy.

and false negative samples. False negative samples will not be processed by our method but false positive samples will, yielding bad classification and reducing the success rate of our method. In practice, the success rate of our method will drop slightly due to false positive detected samples, but we hope that with the rapid development of this field, a near-optimal detector will be available soon.

We presented a novel problem, class retrieval, and the recovery from adversarial attacks along with a proposed solution, which can be used as a baseline approach in further experiments. Our retriever is a self-evident addition to adversarial attack detectors and the combination of these two methods can enable the practical applicability of deep neural networks even in case of attacks. We investigated four different adversarial attacks (PGD, MPGD, Deepfool, TPGD and PGDDLR) on three different datasets (MNIST, CIFAR10 and ImageNet). The results are promising and consistent across all attacks and datasets, where the average accuracy is 72%, 65%, and 65%, respectively. Our retrieval algorithm was not able to recover the original class in all cases but, as a preliminary concept, it clearly shows that it is possible to build an algorithm where the original class can be retrieved. We hope this can open the way for further development and fine-tuning of class retrievals of adversarial attacks, which can increase the robustness of deep neural networks in real-world applications.

DOI:10.15774/PPKE.ITK.2023.005

3.4.6 Thesis Point 2

According to the results of this chapter which were published in the journal of Applied Sciences 2021 [77], I formulated my second thesis point as the following: I created a baseline for adversarial attack recovery and showed that it is a necessary extension of adversarial attack detection in practical problems. I demonstrated that using an algorithm based on counter-attacks can retrieve the original input classes with high confidence reaching 68% accuracy over MNIST, CIFAR-10, and a subset of ImageNet.

DOI:10.15774/PPKE.ITK.2023.005

DOI:10.15774/PPKE.ITK.2023.005

Chapter 4

Incorporating spatial information in image segmentation

4.1 Introduction

The previous chapters raised some concerns about many issues surrounding the usage of neural networks and proposed a few feasible solutions. In the coming chapters, I will investigate many novel ideas with the aim to improve the performance of neural networks where much room for improvement is possible in each step of training e.g. activation function, batch training, batch normalization, loss function, neural network architecture and etc. Mathematically defined problems have a perspicuous solution, but in applied science, the definition of a solution is not clear where a small improvement can be considered as a solution e.g., it is not clear which solution is acceptable in the task of self-driving cars; is it when exceeding a predefined confidence, performing better than a human being or even having less than a threshold of false positive samples. Although I used the term improvements, it can be debatable that an improvement is just a solution for a problem/shortcoming e.g. the importance of the border of a shape for a loss function; if a new loss function takes the object's border into consideration, is it an improvement, or just a solution for the shortcoming of previous loss functions? In this chapter, I will focus on improving the loss function of neural networks for the task of image segmentation.

The application of neural networks and modern machine learning techniques opened up various applications for image segmentation, where instead of, or

DOI:10.15774/PPKE.ITK.2023.005

additionally to bounding box detection a pixel-level segmentation of input images can be created. In the past years, segmentation networks became ubiquitous in computer vision applications, since they usually provide a better understanding of scenes then classification or detection with bounding boxes.

These methods are applied in various tasks from medical imaging [1] to self-driving cars[2]. These methods may vary depending on the selected architectures (U-Net[1], SegNet[78], Mask-RCNN[79], RetinaNet[80]) or even on the exact specification of the segmentation problem (semantic segmentation[81], instance segmentation[82] or amodal segmentation[83]), but all of these approaches require a metric which will compare the actual network output to the expected, ideal outcome or ground truth.

These distances are indispensable for classification, data clusterization, or in the application of any modern supervised learning method for artificial intelligence. From an engineering point of view, a metric is inherently a simplification of the problem representation, which condenses similarity or difference between two high-dimensional data points into a scalar value, and if significant and important data is lost during this projection the algorithm can not provide correct results.

Apart from information compression one may have another important expectation about a proper metric, which works against the generality of information compression. On one hand, the metric has to be sensitive enough to allow comparison in an abstract space, meanwhile on the other hand it has to be robust to filter out noise.

In current applications in almost all cases, a pixel-based distance is applied, where two images are compared to each other according to a given metric (like L1, L2, or Smooth-L1[84] distances). Similarly, the outcome image and the ground truth can be considered as probability distributions and cross-entropy can be applied to determine the distance between them, but none of these metrics take into account the position of the differences.

It is not our aim to speak against intensity-based distances and loss functions, but we would like to present that a metric involving topological information about the shape of the object and the relative position of the differences can have additional value in network training.

DOI:10.15774/PPKE.ITK.2023.005

The representation of topological information in loss calculation has appeared in the past year in various papers, such as in [85], [86] or [87], but all these approaches in their core calculate the pixel-wise differences and approximate topology using persistence barcode calculation[86] or skeletonization[87].

One can easily see, that in the case of a perfect solution the loss will indeed be zero for every metric and higher losses will encode either a larger area of altered pixels, larger intensity differences, or both. But in case of errors with similar values the position and shape of the misclassified pixels will also matter. Simply putting a disc with an area of hundred pixels should be considered differently than hundred individual points distributed all over the image.

Additionally, loss functions should also identify those regions which are responsible for the error. In the case of segmentation, falsely detected pixels around the real object and not segmented pixels inside a homogeneous region are usually caused by the false detection of the boundary and not because of the exact pixels at that position. This weighting is implicitly present in the network via the downscaling operations in different layers, but it is advantageous to explicitly find the source of the error at the calculation of the loss function. These aspects are present in boundary losses, such as in [88], where the boundary regions of the ground truth masks are handled with increased importance, but the topology of other regions is not represented at all.

In this chapter, we would like to show that these differences matter, and topological information incorporated in the loss function can be used to increase the accuracy of segmentation networks.

In Section 4.2 we will demonstrate how the lack of topological information could tamper binary object comparison, describe binary wave metric, and how it can be used for shape comparison. In Section 4.3 we will introduce the extension of the wave metric to three dimensions, to make it applicable on gray-scale images and two-dimensional probability distributions. In Section 5.4 we will compare our approach to other commonly applied metrics via commonly applied datasets and architectures. We have implemented and compared our topographic loss with the traditional L1 loss and Smooth-L1 loss on two different datasets, our new and simple dataset as described in section 4.4 and COCO dataset[89], with two different architectures U-Net[1] and Mask-RCNN[79].

DOI:10.15774/PPKE.ITK.2023.005

4.2 Comparison of Shapes and the Binary Wave Metric

In every application currently, non-topographic metrics are applied to calculate pixel-wise differences between images, which completely neglects topographic information. In this section, we will focus on binary (black and white images) to illustrate the flaw of pixel-based metrics and reveal how wave metric can enhance these similarity functions.

In a particular problem, a metric has to be chosen depending on the nature of the problem. All metrics are problem dependent and since a metric condenses high-dimensional similarities into a scalar value, no metric can be general and well-performing for every practical problem. This motivates us to use topographic metrics for topographic problems, like segmentation.

The most commonly applied metrics for binary objects are Hamming[90] and Hausdorff[91] distances.

Hamming distance computes the number of differing pixels between two images:

$$D_{Hm} = \sum (A \bigcup B) \setminus (A \bigcap B) \tag{4.1}$$

Where A and B are the input images both containing only values of zeros and ones. This metric is fast and easy to calculate and although it is commonly applied to compare shape of various objects, in many tasks it performs poorly, because of a complete lack of topological information. This metric is also commonly referred to as an area-based metric since only the area of the different regions will determine the metric and take into account the number of different pixels regardless of their neighbors or their relative positions. Almost all popularly used metrics such as cross-entropy, Dice[92], Lovasz[93] or Tversky[94] losses are area-based metrics, where the area of the different regions matter, their topologies are not considered.

In the case of grayscale images, an extension of this metric can be applied as the pixel-wise difference between the two images, these metrics are usually referred to as L1: $(\sum_i |a_i - b_i|)$ or L2: $(\sqrt{\sum_i (a_i - b_i)^2})$ distances.

Figure 4.1 demonstrates a simple illustration of the main deficiency of the Hamming metric: the complete lack of topological information.

DOI:10.15774/PPKE.ITK.2023.005

Figure 4.1: This image demonstrates the contradiction between the Hamming distance and subjective human judgments. However every human observer would judge the middle-right pair as more similar, the Hamming distances between the middle and left and the distance between the middle and right images are exactly the same. Our perception is based not only on the area of the differing parts but also on shape-related information. This information can not be ignored if we want to create a trustworthy metric. One can see that the Hamming distance (and any other area-based metric) can provide misleading decisions. Thus, shape-related descriptors are also required.

The other often used metric is the Hausdorff distance[91], which is determined solely by topological differences between the objects:

$$D_{Hs} = max(h(A,B), h(B,A)) \tag{4.2}$$

Where $h(F,G) = \max_{f \in F} \min_{g \in G} d(f,g)$. The sensitivity of this metric to noise prevents its utilization in practical applications. If the largest distances between two different pixels are the same on two image pairs, the metric will return the same result and hides all other information about shapes as well.

The illustration of the Hausdorff distance and its poor applicability can be seen on figure 4.2.

4.2.1 Binary Wave Metric

Both the Hamming and Hausdorff metrics reveal important properties about similarity, but to create a multipurpose, efficient metric, their advantageous properties should be combined, eliminating their flaws. The application of different metrics in a parallel manner might be beneficial, but in the case of simultaneously computed metrics one will increase processing time, and algorithmic complexity and we may not be able to solve the problem, since the weighting of these metrics during the combination is always problematic. One has eventually combined all the metrics into a single function that results a scalar to ease classification and comparison,

DOI:10.15774/PPKE.ITK.2023.005

Figure 4.2: This image illustrates the contradiction between the Hausdorff distance and subjective judgments. Hausdorff distances between the image in the middle and the one to the left and between the image in the middle and the one to the right are exactly the same. A human observer would most probably select the middle-left pair as more similar images. Natural vision and perception are not only based on the topology of the differing parts, but also on area-based information. This information also has to be considered during computation to produce a reliable and useful metric.

The idea of the binary wave metric was first introduced by Istvan Szatmari in 1999 [95]. His work covers the metric calculation for convex, binary objects only and in this work, we will extend it to non-convex, grayscale images, which makes it applicable as a loss function in image segmentation algorithms. This metric can be defined as the volume of an ascending wave starting from the intersection of the objects and filling out the area defined by the union of the two binary objects. On a suitable hardware architecture, the non-linear wave metric can measure both the shape and the area difference between two objects in a single operation (e.g.: on a multi-layer cellular neural network)[96].

Based on this, the equation of the metric calculation for convex two-dimensional, binary objects can be given as the following :

$$W_M(A,B) = \int_{x,y \in D_{Hm}(A,B)} D_{Hs}(x,y) \tag{4.3}$$

Which is the point-wise integration of local Hausdorff distances over every point in the disjunctive union of the two objects. In the case of non-convex objects, the non-linear wave metric is the integration of the local Hausdorff distances along the shortest path in the union of the two sets.

It can be easily seen that the wave metric contains and compresses both previously introduced metrics. The maximal height of the ascending wave (the propagation time of the wave) is proportional to the value of the Hausdorff metric

DOI:10.15774/PPKE.ITK.2023.005

for connected objects and the area of the wave propagation is proportional to the Hamming distance.

The slope, the increase of the wave for each step, determines the connection between the topological (Hausdorff) and the area-based (Hamming) information. For example, if this increase is set to zero and propagation starts with a constant, non-zero magnitude, the wave metric will yield the Hamming distance multiplied by the initial constant. In case of a larger slope, the metric will shift more towards the Hausdorff metric and the distances between the further and further differing points will determine the result more and more.

It is easy to see that this similarity function fulfills almost all the required properties of a metric. It can be defined as a function on a given set $d : X \times X \mapsto \mathbb{R}$ and it fulfills the following properties ($\forall a, b, c \in X$): non-negativity or separation axiom: $d(a, b) \geq 0$, identity of indiscernibles, or coincidence axiom: $d(a, b) = 0 \Leftrightarrow a = b$ and symmetry:$d(a, b) = d(a, b)$.

Unfortunately the triangle inequality or subadditivity axiom:

$$d(a, c) \leq d(a, b) + d(b, c)$$

does not hold this way, since three objects, from which two are completely disjoint (a and c) and the third which has a common part with both (b), would result in a zero value for $d(a, c)$ and a non-zero value for both $d(a, b)$ and $d(b, c)$.

To come around this problem we have applied an extra penalty for points that can not be reached in the union during wave propagation. This penalty has to be larger than the maximum penalty in the reachable region. With this addition, the wave metric fulfills all the axioms and forms a proper metric for non-connected objects as well. The illustration of the wave propagation and the metric can be seen in figure 4.4 and figure **??**.

This metric can be used to compare images and it is used for accuracy calculations, but unfortunately can not be used during network training, as it was shown in [97], since it can only be calculated between two binary images.

DOI:10.15774/PPKE.ITK.2023.005

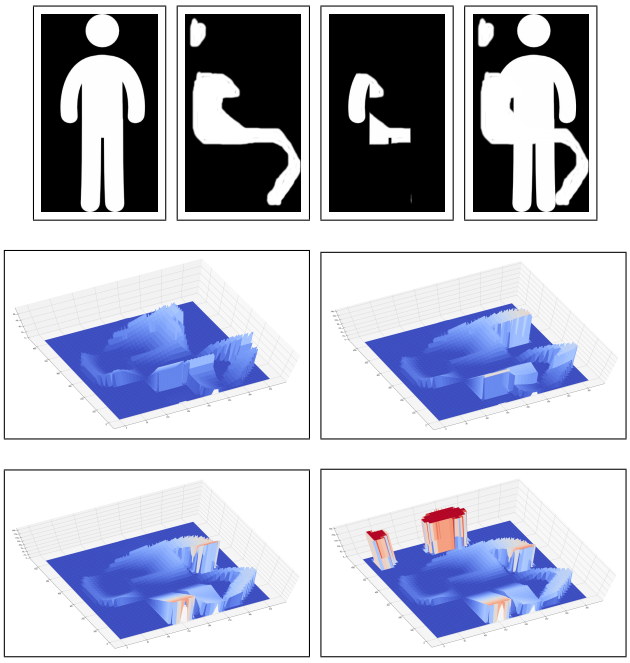

Figure 4.3: Illustration of the wave propagation. The first row depicts two possible binary input images (first and second images from the left) and their intersection and union (third and fourth images from the left). The last two rows depict four 3D versions of the wave metric in an increasing manner until reaching the union at iteration 100, 150, 300, and the last iteration including not reached regions. During propagation further and further pixels will be incorporated in the loss function with higher and higher values. In the last step, a high penalty will be assigned to all pixels that were not reached during propagation.

DOI:10.15774/PPKE.ITK.2023.005

Figure 4.4: Illustration of the steps of the wave propagation. We start from the intersection (top left corner) until reaching the union (bottom right corner). The figure depicts 12 steps, 3 steps in each row. Each step comprises of two stages which are shown in sequential order. In the first stage, the wave propagates spatially covering more space until reaching the union while in the second stage the intensity of the wave increase with a constant step until reaching the union. In each step, the first stage will propagate the wave for 14 pixels and the second stage will increase the intensity by 0.09. During propagation further and further pixels will be incorporated in the loss function with higher and higher values as you can see in the progression of the wave.

DOI:10.15774/PPKE.ITK.2023.005

4.3 Wave Loss: Extension of the Wave Metric to Three-dimensions

In the previous section, the binary wave metric was described and, as it was demonstrated, it creates a connection between topological and area-based metrics. To extend it to gray-scale images and two-dimensional probability distributions we have to consider intensity-based differences as well.

The metric should depend on three not-independent measures: The area of the differences, the topology of the differences, and the intensities, the values of the differences. In this case, the output and ground-truth images can be imagined as two two-dimensional surfaces in three dimensions. From this, we can calculate the intersection and the union (which will also be two-dimensional surfaces) then the metric can be imagined as a three-dimensional wave propagating and filling out the space between these two surfaces. A weight will be associated with every new voxel at each time step of the propagation and this four-dimensional volume (the weighted sum of the three-dimensional changes) will be called wave loss.

Our goal was to differentiate between value and topology-based differences and because of this the propagation speed of the wave is different in the z (intensity) and x, y (topological directions)[1].

Compared to the binary wave metric, where only topological distances were covered, an upper bound for the number of required steps till convergence can easily be identified. An upper bound for spatial propagation can also be found (identifying the object containing the longest possible path with the given image size), but this bound is fairly high compared to the number of steps required to cover differences in intensity. In practice, this means that a small number of iterations will be enough to calculate the metric.

Our algorithm, as described in pseudo-code, calculates wave loss for two gray-scale images. The input values are: Img1 and Img2 and the output of the algorithm is a scalar variable WaveLoss. The parameters of the algorithm are the following:

[1] the wave could propagate differently along the x and y dimensions as well, but in image processing applications these dimensions are usually handled in the same manner

DOI:10.15774/PPKE.ITK.2023.005

Algorithm 4: Calculation of Wave Loss

Data: Img1, Img2
Parameters : ValInc, SpaInc, SpaW, ValW
Result: WaveLoss

$Union \leftarrow max(Img1, Img2)$
$CurrentWave \leftarrow min(Img1, Img2)$
$WaveLoss = 0$
for $i \leftarrow 0$ **to** $1/ValInc$ **do**
    /* Loss for intensity differences */
    NewWave+=ValInc;
    NewWave=min(NewWave,Union);
    ValueChange=sum(NewWave-CurrentWave);
    WaveLoss+=ValW[i]*ValueChange;
    CurrentWave = NewWave;
    /* Loss for spatial differences */
    NewWave= maxpool(CurrentWave,[SpaInc,SpaInc], [1,1]);
    NewWave=min(NewWave,Union);
    SpatialChange=sum(NewWave-CurrentWave);
    WaveLoss+=SpaW[i]*SpatialChange;
    CurrentWave = NewWave;
end

DOI:10.15774/PPKE.ITK.2023.005

- ValInc will determine how fast the wave propagates along the intensity differences, the intensity of every pixel will increase by this amount in every iteration. This Parameter will also determine the maximum number of required iterations and by this, it will also determine the largest distance from the intersection where topological differences are considered. Having a larger distance than the maximal receptive field of a neuron in the network is illogical because this way the error could be derived back to the neuron, which had no vote in the classification of that input pixel. In our Experiments this value was between 0.05 and 0.1, meaning that the wave from a selected point could propagate for 20 and 10 pixels.
- SpaInc will determine the spatial propagation speed of the wave. Spatial propagation is implemented by a max pooling operation with window size: SpaInc and a stride of one. In our simulations, this value was always set to 3.
- ValW is a vector of penalties for the intensity differences. If this value is constant, the weight differences will be linearly proportional to the penalties in the loss. If this is increasing, it means larger differences (where more iterations are required to reach the desired value) will have larger and larger penalties. In our simulations, we were using constant values in ValW.
- SpaW is a vector containing the penalties for topographical differences. SpaW[0] will weight those points which can be reached in one spatial propagation and which are in the direct neighborhood of the intersection. SpaW[k] will have a penalty for those values which will be reached at the k-th iteration. In our simulations, we have applied linearly increasing values which were all lower than the values of ValW. In most networks, we want to have good results on average, but minor mistakes about the shape of the object can be tolerated. Applying lower values than the intensity weights (ValW) means, that the importance of the shape of the segmented object will become less important. Monotonically increasing SpaW means that the further we are from the object, the higher impact a misclassification will result. Applying higher weights than ValW, which are monotonically decreasing would mean that the boundaries are really

DOI:10.15774/PPKE.ITK.2023.005

important, and classifying a pixel around a boundary is a larger problem than misclassifying a pixel somewhere far from the object.

One can easily see that the wave metric is an extension of the normal L1 metric, if there is no spatial propagation ($SpaInc = 0$) and ValW values are all the same, we will get L1 loss as a result. Similarly, if the values in ValW are increasing exponentially with no spatial propagation, this metric will calculate the traditional cross-entropy, between the two images, representing two-dimensional distributions.

During the calculations, we first increase $CurrentWave$ according to the intensities. This is a global change and it happens everywhere in the image where the values have not reached the union and after this step, we apply spatial propagation. One could change this order, but we considered intensity-based differences more important. One could also execute both propagations separately and sum their penalties, but we did not observe any measurable effect applying this modification. In this setup compared to the binary implementation, all points are reached during propagation since the intensity differences are limited, therefore there is no need for an extra penalty for unreached regions.

Since it is difficult to plot wave loss using two-dimensional images, we have opted to display it using two one-dimensional gray-scale 'images'. This can be seen in figure 4.5

From an implementation point of view, value increase is just an addition and spatial propagating is a grayscale dilation, which is essentially a max pooling operation that can be found in every modern machine learning environment. For a training step, the number of additional pooling operations is fairly small compared to the pooling operations already contained by a typical convolutional neural network. Therefore the calculation of the wave loss will not increase training time significantly and has no effect on inference time.

4.4 Simple dataset for segmentation

Since we were not able to find a simple segmentation dataset (like MNIST[98] or CIFAR for classification), we have created a simple dataset based on CLEVR[99].

DOI:10.15774/PPKE.ITK.2023.005

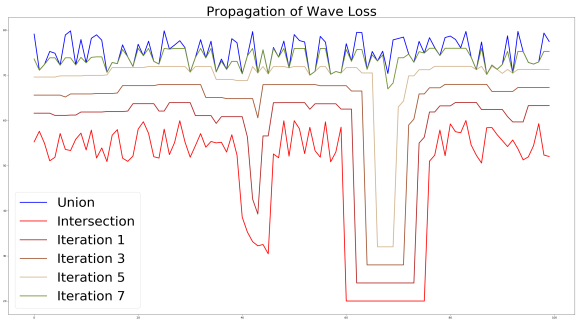

Figure 4.5: The propagation of the wave during the calculation of wave loss is depicted on this figure using one-dimensional inputs. as it can be seen the wave fills out the region between the intersection and the union of the two surfaces. At each propagation, the newly reached pixels will be weighted and added to the loss function. This metric incorporates intensity, area, and shape-related information.

The dataset contains 25200 colored images (of size 320×240) of simple objects along with their instance masks, amodal masks, pairwise occlusions, and three-dimensional coordinates for each object. This produce a simple dataset for various tasks, like three-dimensional reconstruction, instance segmentation, and amodal segmentation.

For the sake of reproducibility and a detailed description of parameter setting, our code for network training and evaluation on both datasets along with the data generation script for the CLEVR dataset can be found at `https://github.com/horan85/waveloss`. The dataset contains objects of simple shapes, but also contains shadows, reflections, and different illuminations, which make it relevant for the evaluation of segmentation algorithms. An example image of the dataset along with a few generated masks can be seen in figure 4.6.

4.5 Comparison and Results

4.5.1 simple simulated dataset: CLEVR

We have selected the U-net architecture to compare Wave Loss and L1 loss on our simple CLEVR-inspired dataset. We have used a U-NET like structure containing

DOI:10.15774/PPKE.ITK.2023.005

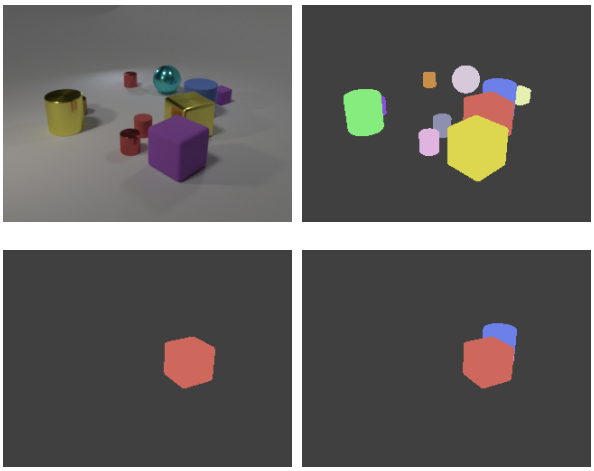

Figure 4.6: An example image from the generated dataset. An input image (top left), the instance segmentation mask (top right), an example amodal mask which was generated for each object individually (bottom left) and the pairwise occlusion mask (bottom right) are displayed on this figure. The pairwise occlusion images with the amodal mask can be used to determine the front-back relation between objects. Apart from the mask the exact object coordinates and sizes are also stored in JSON format.

8,16,32,64 convolution blocks (each 3×3). Downscaling was done by strided convolutions, while upscaling was implemented by transposed convolutions.

We have trained the network 20 times independently on our dataset for semantic segmentation, using 23400 images for training and 1800 images for validation (all the validation scenes were generated independently from the training scenes). In one setup we trained the network to minimize the L1 Loss on the training set, in the other setup Wave loss was defined as the error function. In both cases, we have measured both L1 and Wave loss during training and validation. The Losses can be seen in figure 4.7. Some qualitative examples from the validation set during different train iterations can be seen in Fig. 4.8

As one can see from these measurements wave loss results faster convergence and better accuracy both on the training and the validation sets. These simple examples show the applicability of wave loss in segmentation tasks, but to demonstrate the advantage of this topological loss function, we have to investigate it on more complex and practical tasks.

DOI:10.15774/PPKE.ITK.2023.005

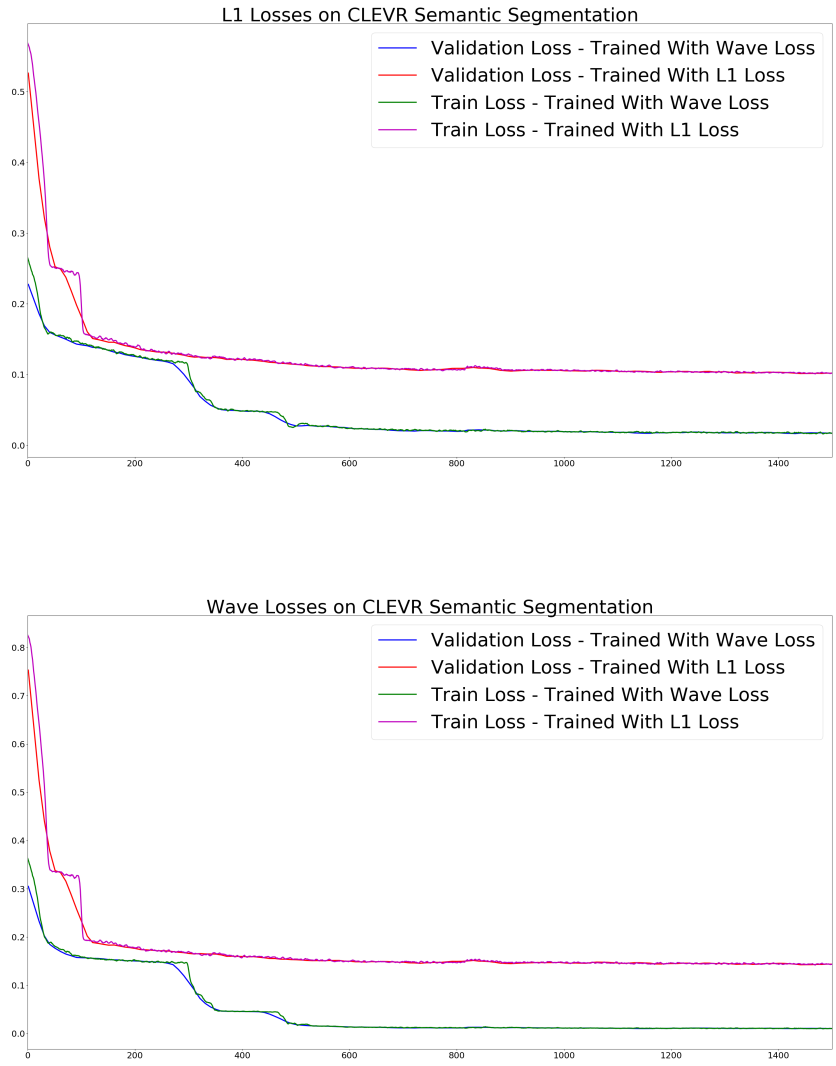

Figure 4.7: Losses on the CLEVR dataset, averaged out on 20 independent runs. We can see L1 loss plot at the top row and wave loss plot at the bottom.

DOI:10.15774/PPKE.ITK.2023.005

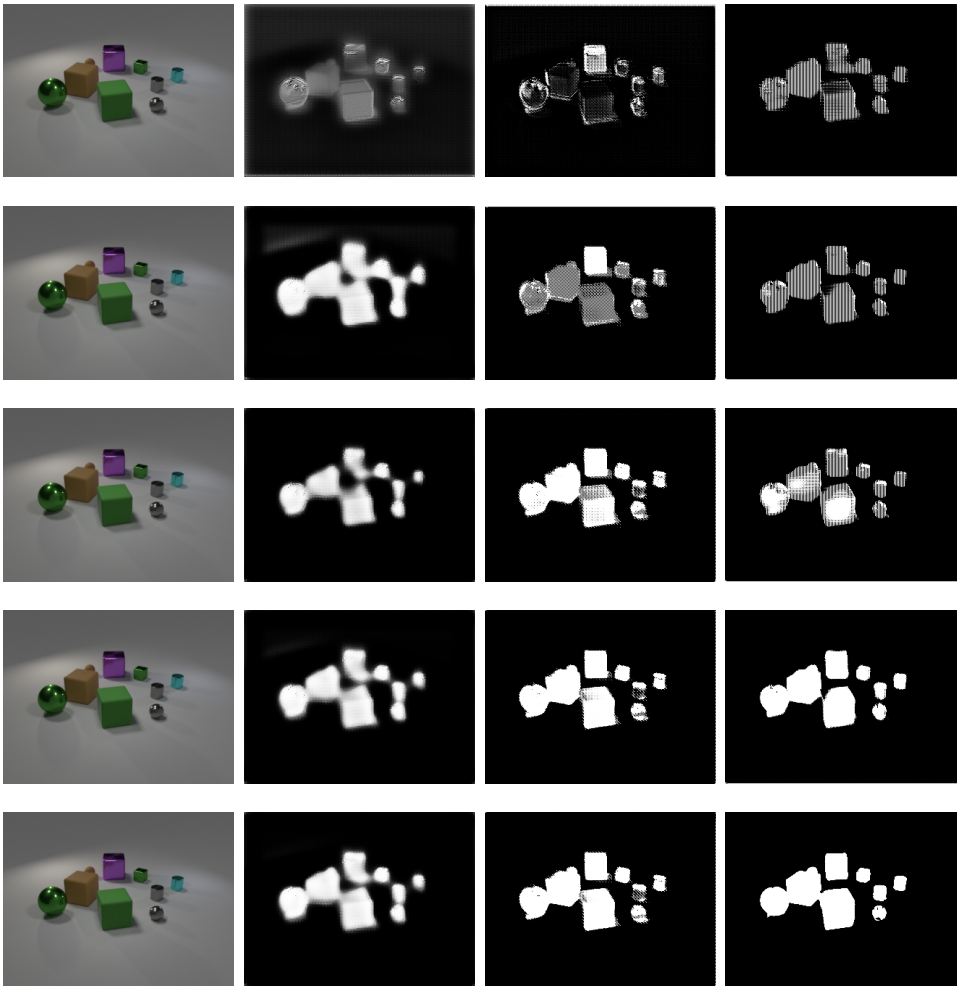

Figure 4.8: Qualitative examples for a randomly selected test sample at 200, 400, 600, 800, 1000 iterations (rows). The first column depicts the input image, the second column depicts network outputs which is trained by L1 loss, the third column by cross-entropy, and the fourth column by wave loss. As can be seen in the first row, wave Loss creates a checkered pattern on the objects, since having one large value in each region decreases the loss function, but later gives better segmentation than any of the other investigated losses.

DOI:10.15774/PPKE.ITK.2023.005

4.5.2 Semantic segmentation on Cityscapes

We investigated the Cityscapes dataset [100] with the following architectures: SegNet [78], HRNET [101], DeepLab [102] and DeepLabv3 [103]. We trained these networks with three different loss functions (ℓ_1, cross-entropy, and wave loss) and the accuracy results can be found in Table 4.1. During training, we initialized the weights randomly and executed five independent trainings with every configuration and trained them for 400,000 iterations.

The parameters of our loss function were the following: SpaInc was set to three; this means propagation happened using 3×3 kernels. Topology weights (SpaW) exponentially increased from 0.01 to 1 and intensity weights (ValW) were all set to a constant value of one. ValInc was set to 0.05; this means that the largest value gap of one will be filled in twenty iterations. Using this value, neighborhoods of maximum of twenty pixels are affected by wave propagation. Since the input resolution of the networks was 513×513, we considered these 20×20 neighborhoods sufficiently large.

As can be seen from the results, the application of wave loss increased the network performance compared to traditionally used cross-entropy loss with an approximated 3% in the case of all network architectures and provided better segmentation accuracy than any of the investigated loss functions in all cases.

Table 4.1: This table contains the average accuracy results of five independent runs on the Cityscapes dataset using four different network architectures (rows) and six different loss functions for semantic segmentation.

Model	**L1 Loss**	**CrossEnt**	**Dice**	**Boundary**	**ShapeAware**	**Wave**
SegNet	54.2%	57.0%	57.3%	57.7	58.6%	59.5%
DeepLab	59.7%	63.1%	64.1%	64.3%	65.4%	66.7%
DeepLabv3	77.6%	81.3%	81.4%	81.5%	81.7%	82.2%
HRNET	77.4%	81.6%	81.8%	81.8%	82.1%	83.4%

4.5.3 Instance segmentation on MS-COCO

We also investigated the problem of instance segmentation on COCO 2017 [89]. We investigated MASK-RCNN with different backbone architectures using the

DOI:10.15774/PPKE.ITK.2023.005

Detectron 2 framework where we kept the architecture and all the other parameters unchanged in the configuration files used for instance segmentation on this dataset. These configurations contained data augmentation in the input samples containing random flip, random crop, brightness change, and random additive noise. The original training script used cross entropy and we added our implementation of the wave loss to the framework and compared its performance.

The parameters of our loss function were the same as in the case of the Cityscapes dataset. We would like to emphasize that the images used for segmentation differ significantly in size from the images used in Cityscapes since in the case of Mask R-CNN the segmentation head is executed on the 28×28 outputs of the RoiAlign layer. Even though the object sizes may differ, the same parametrization worked well for this architecture and dataset as well, which demonstrates that our loss function is not heavily dependent on the exact parameter values.

We measured mean average precision values using the evaluation script of COCO. We have to note that IOU is more related to wave loss than to L1 metric or cross-entropy since wave loss uses the intersection and union to determine wave propagation, but we think this does not bring an unfair bias to the evaluation. The results can be seen in Table 4.2. As can be seen from the results, the application of the wave loss increased the precision of the network with an overall 3% on our validation set and the network performs especially better in the case of small objects, where an improvement of 5% was achieved compared to our reference network trained by cross-entropy loss. We also have to emphasize that segmentation improved in the case of every architecture and for all object sizes. Qualitative results about the generated masks and bounding boxes can be seen in Figure 4.9. The segmentation masks in the first column were generated by a network trained with cross-entropy loss; the masks in the second column are the results of a network trained with wave loss. As can be seen, wave loss does not cover boundaries as sharply, but altogether gives better coverage of the objects. (Although we have to note that this judgment might be subjective and could also depend on the exact parametrization of wave loss.)

DOI:10.15774/PPKE.ITK.2023.005

Figure 4.9: Example results on the COCO dataset segmented with Mask-RCNN. Images in the first column are taken from the validation set and the masks were generated by a network trained with cross-entropy loss, the images in the second column are the results of a network trained with Wave loss. As can be seen Wave loss does not cover boundaries as sharply, but altogether gives better coverage of the objects. (Although we have to note that this judgment might be subjective and could also depend on the exact parametrization of wave loss.)

DOI:10.15774/PPKE.ITK.2023.005

Table 4.2: Average precision results on COCO 2017 validation set using the same network architectures with three different loss functions in different columns (ℓ_1, cross-entropy, Dice loss, active boundary loss, shape aware loss, and wave loss). Two different architectures (ResNet-50 and ResNet-101) can be found in the rows, with feature pyramid networks (FPNs) or when the activation of the fourth convolution layer (C4) was used for region proposals. The results display the mean average precision for all objects, except the last three rows, where the accuracy results for the best-performing network are detailed for small-, medium- and large-sized objects as well.

Model	**L1**	**CrossEnt**	**Dice**	**Boundary**	**Shape**	**Wave**
R50-C4 mAP all	28.75%	32.2%	32.83%	32.9%	34.721%	35.93%
R50-FPN mAP all	29.43%	35.2%	36.14%	36.12%	37.53%	38.11%
R101-C4 mAP all	30.17%	36.7%	37.2%	37.4%	38.86%	38.23%
R101-FPN mAP all	31.67%	38.6%	38.8%	39.3%	40.25%	41.7%
R101-FPN mAP s	14.25%	17.37%	18.18%	18.35%	19.33%	22.24%
R101-FPN mAP m	37.53%	39.23%	39.74%	40.52%	41.27%	43.26%
R101-FPN mAP l	50.14%	51.64%	51.83%	52.17%	52.22%	53.27%

4.5.4 Discussions

My results presented in the previous section clearly demonstrate that incorporating topological information in the loss function can improve the segmentation accuracy of various network architectures. The results in Table 4.1 and 4.2 demonstrate that our approach not only improved the accuracy by 3% on average but also performs better than any of the other loss functions selected for comparison. I have compared our method to a recent reformulation of Dice loss [92], which calculates the ratio between the intersection and the union between the two binary objects. Unfortunately, Dice loss is a metric applied over binary images and they are based solely on area-based differences. Until the number of pixels in the intersection and union remains the same, the regions can change arbitrarily. Namely, the different pixels can move anywhere in the image space; only the number of different pixels matters.

Another recent improvement over the area-based metric is the active boundary loss [104] where, as an additional loss value, the boundary pixels are calculated with a larger weight, this way representing the shape of the object in the loss

DOI:10.15774/PPKE.ITK.2023.005

function. This is more similar to our approach, but it considers only the boundary pixels and no other pixels in the differing area. The third selected loss is the shape aware loss function [105] which considers all pixels in a differing region, but with a pre-computed weight which is the pixels' Euclidean distance from the intersection. Unfortunately, this distance is not the same as the shortest path of differing pixels since in the case of non-convex regions this distance can be significantly larger.

One can easily see that incorporating more topographic information in the loss functions improved segmentation accuracy and the best results could be achieved with wave loss. We also have to note that our method is not another completely different loss function, but a combination and generalization of the area and distance-based metrics. For example, setting the $SpaW$ parameter to an all-zero vector except the first value will result in the spatial information being incorporated only at one pixel from the intersection, which is exactly the same region as the boundary of the object. This way, one can calculate boundary loss using our method. Similarly, if all $SpaW$ values are zero our metric will compute an area-based metric, similar to the Hamming distance or Dice score.

On the other hand, if $ValW$ parameters are all set to zero, only the distance of the differing pixels will be a determining factor similar to Hausdorff distance. These results show that our approach defines a more general metric that can mimic most of the previously applied loss functions and with proper parameterization it can also perform better in practical applications.

4.5.5 Thesis Point 3

According to the results of this chapter which were published in the Mathematics journal 2022 [106], I formulated my third thesis point as the following: I demonstrated that a three-dimensional extension of wave loss can be employed as a loss function in the training of deep neural network in case of segmentation problems. I have shown that, with proper parameters setting, wave loss can increase the accuracy of segmentation with 2% on the MS-COCO dataset and cityscapes dataset as well.

DOI:10.15774/PPKE.ITK.2023.005

Chapter 5

Filtered batch normalization

5.1 Introduction

So far we investigated two approaches to improve the performance of neural networks by explicitly/implicitly guiding the gradient of the loss function. We will introduce another potential improvement by modifying another commonly used training step, activation normalization.

The application of normalization methods is ubiquitous in signal processing and has a long history in machine learning. Various techniques were introduced in the past decade, such as local response normalization, which was one of the cornerstone components of AlexNet [107], instance normalization [108], which is commonly applied in style transfer networks or layer normalization [109], which is mostly applied in recurrent neural networks. The most generally applied and in most cases best performing normalization technique still remained batch normalization (BN), which was introduced in [110].

Layer Normalization (LN) [109] computes the normalization statistics from the entire layer i.e. using all the activation channels. In contrast, like BN, Instance Normalization (IN) [108] computes the normalization statistics for each channel independently, but only from the sample being normalized, as opposed to the entire batch, as BN does. It was shown to be useful for style transfer applications but was not successfully applied for recognition. Group Normalization (GN) [111] fills the middle ground between the two. It computes the normalization statistics over groups of channels.

DOI:10.15774/PPKE.ITK.2023.005

Batch normalization became an important building block of neural networks in the past five years. It was demonstrated in various tasks that this method can accelerate network training and results in higher test accuracy in practice if the mini-batch size is sufficiently high.

The beneficial effect of batch normalization was introduced in [110] hypothesizing that the reduction of internal covariate shift -which is the imposed change in the input distribution of layers triggered by the updates of the preceding layers - can result in faster convergence.

Although [112] demonstrated that mitigating internal covariate shift plays only a minor role in the effectiveness of BN, it is most beneficial as a regularizer which parametrizes the activations during training to make the optimization problem more stable by creating a smoother loss landscape and increasing the predictability of gradients rendering a robustness against exploding or vanishing gradients, hyperparameters, and initialization sensitivity.

BN is useful in training but can be difficult to be applied during inference since usually a single input is presented instead of mini-batches. Batch re-normalization [113] was introduced to mitigate this problem, where instead of the calculated first and second-order moments, smoothed moving averaged mean and variance values are used. The same paper also suggested that mean and variance values can be inconsistent in the case of smaller mini-batches, but smoothing them with moving averages can help to produce more consistent values for normalization.

We will demonstrate that BN with or without running averages can still induce inconsistent mean and variance values even with fairly large mini-batches (such as 128 or 256) and this is caused by unlikely large activations in the network (assuming that they follow Gaussian distribution). We will demonstrate that the specificity of network activations for certain features works against the common assumption that activations can be described well by Gaussian distributions. Exploiting these facts, we will demonstrate a method that filters out these out-of-distribution samples in batch normalization using robust statistics, resulting more consistent moments and faster convergence.

Our method adds an additional mean and variance calculation step in training, which does not generate a significant increase in the number of operations in the case of complex networks. Also, our method has no effect on the number of

DOI:10.15774/PPKE.ITK.2023.005

additional operations in inference, hence the trained networks can be executed with the same computational performance as in the case of BN.

It was also recently demonstrated that although batch normalization is still the best-performing approach in the case of sufficiently large mini-batches, the performance can deteriorate in the case of smaller ones. Unfortunately, in the case of complex models like Resnet-101 [114] or Mask-RCNN [79] only small mini-batches fit in the GPU memory (in the case of Mask-RCNN with ResNext-101[115] backbone, only mini-batches of two images can fit the memory limit of an NVIDIA RTX 2080 TI), resulting inferior performance. This can be improved using group normalization [111] where the moments for normalization are calculated over multiple channels and spatial positions, but not over instances of the mini-batch. We will demonstrate that our method can improve group normalization as well. In theory, our approach -the robust calculation of mean and variance values- could be used in other methods as well (such as layer normalization and filter response normalization [116]), but the investigation of these possibilities is out of our scope.

5.2 Batch Normalization and The Distribution of Neural Network Activations

5.2.1 Batch normalization

The aim of batch normalization is to produce coherent output distributions in every iteration at each layer. Assuming that network activations follow a Gaussian distribution, which can be fully described by the mean and variance values, we can transform the output distribution of a network to zero mean and variance of one. After calculating the first two moments (mean and variance), batch normalization scales and shifts the activations using trainable parameters maintaining landscape flexibility. We also have to note that because of this scaling the exact mean and variance which are selected for normalization do not matter from an algorithmic point of view, the moments just have to be constant to ensure the same output distributions.

Based on this, batch normalization can be described by the following formula:

DOI:10.15774/PPKE.ITK.2023.005

$$y_i = \gamma \frac{(x_i - \mu_i)}{\sigma_i} + \beta \tag{5.1}$$

Where x_i are the activations of layer or kernel i, y_i are the transformed activations, γ and β are trainable parameters and μ_i and σ_i are the mean and the standard deviation, which are calculated by the following equations:

$$\mu_i = \frac{1}{m} \sum_{k \in S_i} x_k \tag{5.2}$$

$$\sigma_i = \sqrt{\frac{1}{m} \sum_{k \in S_i} (x_k - \mu_i)^2 + \epsilon} \tag{5.3}$$

Where m is the number of neurons in the selected layer or kernel, ϵ is a small numerical value to avoid division by zero and S_i is the set of activations which are selected for normalization (this set of activations are determined differently in case of group, instance or layer normalization [111]).

5.2.2 Distribution of Neural Network Activations

It is a common assumption that activations follow Gaussian distribution in neural networks before the application of the non-linear transfer functions.

This is true for untrained networks with randomly initialized weights following Gaussian distribution, but is also commonly hypothesized to be true for networks after or during training as well.

Although this assumption is helpful and led to many normalization techniques, it contradicts the assumption that neurons or kernels in convolutional neural networks (CNNs) are feature specific responding and emitting high activations only to certain features. This specificity is obvious at the logit layer: in case of classification problems (but similarly in detection and segmentation as well) one expects a single neuron with large activation, which belongs to the specific class, meanwhile all other neurons should have minimal activations. This more Bernoulli-like distribution is compelled on the neurons during training.

The specificity of deeper kernels was demonstrated in visualization of network responses [117] and attribute maps [118] which showed that kernels in deeper

DOI:10.15774/PPKE.ITK.2023.005

layers of CNNs are feature specific and output high activations only if the proper output class is presented.

To demonstrate this we have measured the normalized activations in each layer in pretrained networks. We have selected VGG-16 [119] with batch normalization as a reference to investigate the activations in each layer after batch normalization on the validation set of ImageNet 2012 [120]. The weights of the pretrained network were taken from the torchivision.models module [1] to ensure reproducability.

We have used this network in inference mode, where previously learned constant values are used for normalization instead of calculating the current mean and variance. We investigated the magnitude of the activations in the batch normalization layer before parameters γ and β are applied, so all activations should have zero mean and variance of one. The activations according to the layers can be seen on figure 5.1. From this plot one can see that although 68% of the activations are in a narrow band ($\pm\sigma$), which results a variance of one, there are some activations outside $\pm 7\sigma$ in every layer. The three fully connected layers at the end of the network have especially extreme activations (-140σ), which should not happen in case of a Gaussian distribution.

The presence of values outside the 7σ range should have a probability of 1/390,682,215,445. This means that considering a layer with 512 kernels and 16x16 positions, a value should appear out of every 2,980,668 input images, so even in the whole validation set of ImageNet the probability of the appearance of such extreme activations is as low as 0.0167. We have also investigated VGG-19 and ResNet-50 architectures and have observed similar activations.

To demonstrate that apart from these outliers the activations are indeed Gaussian we have investigated the distribution of a randomly selected (453th) kernel of the 13th layer (last convolutional layer) of VGG-16-BN. We have plotted the distribution of the activations of this single convolution kernel after normalization on the whole validation set which can be seen in figure 5.2. This distribution can be considered Gaussian with rare outliers, but these outliers can have extremely large values. To demonstrate the specificity of this randomly chosen kernel, we have also selected all samples from the validation set of ImageNet which results

[1] The weights can be downloaded from: `https://download.pytorch.org/models/vgg16_bn-6c64b313.pth`

DOI:10.15774/PPKE.ITK.2023.005

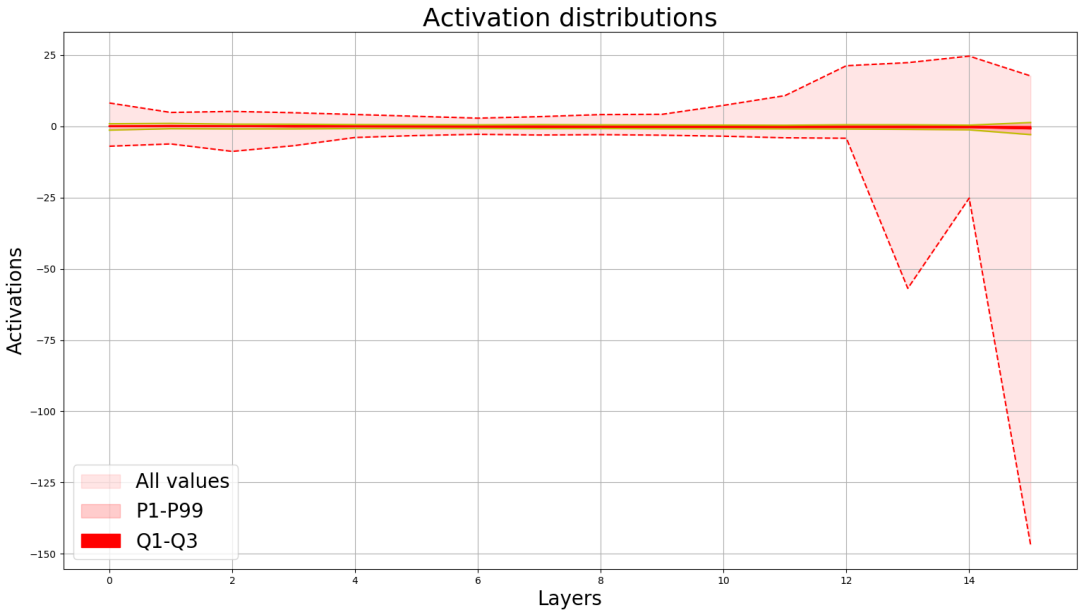

Figure 5.1: This figure depicts the distribution of the activations in a pretrained version of the VGG-16-BN architecture. The dashed lines display the maximum and minimum values in each layer, the golden lines contain 98% of the activations and 50% of them is in the solid red region. This demonstrates that although the data has zero mean and variance of one, it contains outliers especially in layers closer to the logit layer.

DOI:10.15774/PPKE.ITK.2023.005

an activation larger than 14σ. Altogether there were 51 such images and all of them contained people with masks. Some randomly selected samples from these 51 images are also displayed on Figure 5.2. 51 images out of the 50,000 samples of the validation set means that in case of batches of 16, the probability that such image will be present in the mini-batch is 0.01632. Out of every 61 mini-batch there will be one in average, where extremely large activations (above 14σ) will be present, therefore distorted mean and variance values will be calculated for the activations forming the Gaussian like activations.

We have also investigated network activations during training in the AlexNet architecture on the CIFAR-10 dataset. The activations after batch normalization (again without applying the γ and β parameters) in the third convolutional layer are displayed on figure 5.3. As we can easily see, larger and larger activations appear in the network during training.

From the previously presented examples, we can easily see that if we disregard these outliers activations indeed form a Gaussian distribution, but including these samples can heavily change the calculated mean and variance values during normalization. This problem is further exacerbated by the fact that training in case of complex networks and datasets happens in mini-batches. In most mini-batches, activations will have Gaussian distribution, but once a specific extreme activation appears in a channel, it can drastically alter the expected value and the variance which are used for normalization.

To overcome this problem we will introduce filtered batch normalization which removes these outliers before the mean and variance calculation, therefore resulting more consistent distributions over training.

5.3 Filtered Batch Normalization

At the creation of filtered batch normalization our aim was to design an algorithm that would filter out outliers from a distribution which can appear with low probability in mini-batches, but do not modify the mean and variance values if the input is a perfect Gaussian distribution without outliers.

We were investigating commonly applied methods for robust mean and variance calculation [121], throwing out the highest and lowest k-percent samples from the

DOI:10.15774/PPKE.ITK.2023.005

Figure 5.2: The bottom plot displays the density plot of the activations in the 453th kernel of the 13th (last convolutional) layer of VGG-16-BN. As one can see, activations follow Gaussian distribution apart from the extreme outliers, meanwhile the top two rows contain images which resulted activations above 14σ.

DOI:10.15774/PPKE.ITK.2023.005

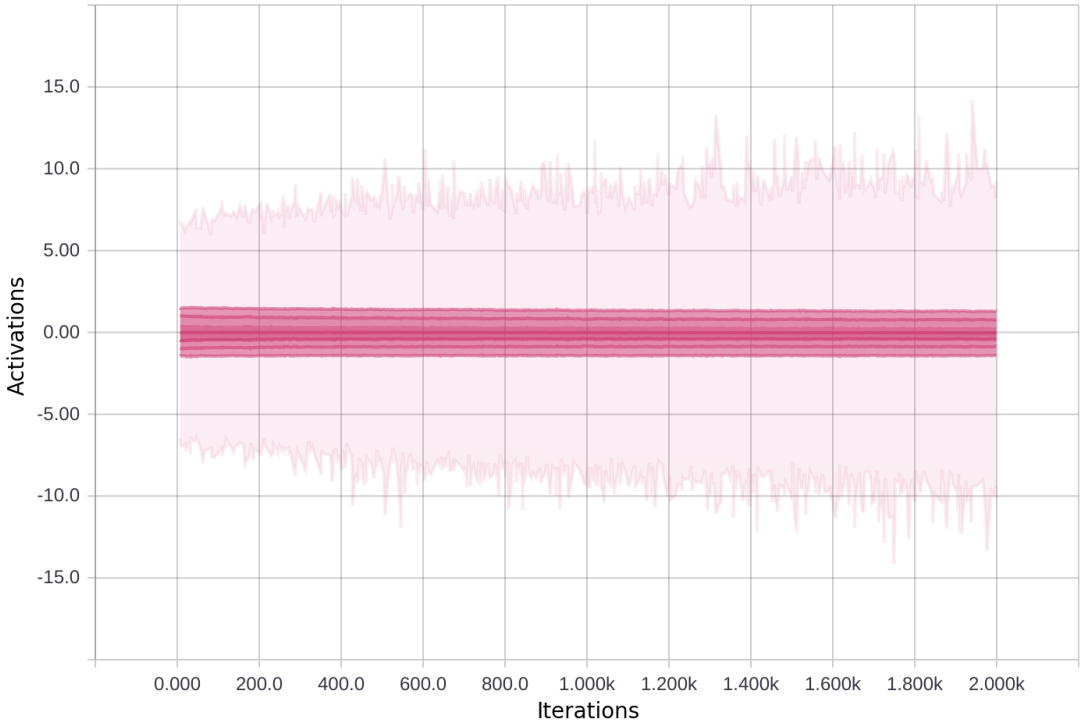

Figure 5.3: This figure depicts the distribution of activations of the third convolutional layer in AlexNet on the CIFAR-10 dataset after batch-normalization during training in the first 2000 iterations (with batches of 64). The X axes contain the training iterations; the Y axes contain the value of the activations. The bold lines represent $25\%, 50\%, 75\%$ of the values, and the fourth lines represent the minimum and maximum values. As one can see activations above 10σ appear in this layer.

DOI:10.15774/PPKE.ITK.2023.005

data or applying winsorization [122], substituting these samples with other values. Unfortunately, getting rid of samples or replacing them will change the variance in those cases where the activations do not contain outliers, determining the optimal value of k is difficult and the additional sorting operation of the samples requires $\mathcal{O}(n \log n)$ operations.

In the first step of the algorithm we calculate μ_i and σ_i values similarly as in equations 5.2 and 5.3, but we do not use these values directly for normalization. We create a Gaussian candidate distribution $\hat{x'}_i$ which might contains outliers, but has zero mean and variance of one:

$$\hat{x'}_i = \frac{1}{\sigma_i}(x_i - \mu_i) \tag{5.4}$$

Based on this Gaussian candidate, we create a mask ($f(x_k)$) to select those values which are only less than T_σ distance from the mean value:

$$f(x_k) = \begin{cases} 1 & \text{if } -T_\sigma \leq \hat{x'}_k \leq T_\sigma \\ 0 & \text{if } \hat{x'}_k < -T_\sigma \vee T_\sigma < \hat{x'}_k \end{cases} \tag{5.5}$$

T_σ is a hyperparameter of the algorithm and the performance of the algorithm does not depend heavily on its value. As it can be seen from the previous example, $T_\sigma = 7$ can filter out most outliers in the data.

We use this mask to calculate the mean and the variance only including those which are not considered outliers (which are inside the $\pm T_\sigma$ band of the mean value).

$$\mu'_i = \frac{1}{\sum\limits_{k \in S_i} f(x_k)} \sum_{k \in S_i} f(x_k) x_k \tag{5.6}$$

$$\sigma'_i = \sqrt{\frac{1}{\sum\limits_{k \in S_i} f(x_k)} \sum_{k \in S_i} f(x_k)(x_k - \mu'_i)^2 + \epsilon} \tag{5.7}$$

μ'_i and σ'_i values are used to transform the activations which can be calculated similarly to equation 5.1:

$$y'_i = \gamma \frac{(x_i - \mu'_i)}{\sigma'_i} + \beta \tag{5.8}$$

DOI:10.15774/PPKE.ITK.2023.005

If our loss at the end of the filtered batch normalization layer is defined as ℓ, backpropagation of the filtered batch normalization layer can be defined by the following equations:

$$\frac{\partial \ell}{\partial {\sigma'}_i^2} = \sum_{k \in S_i} f(x_k) \frac{\partial \ell}{\partial y'_k} \gamma (x_k - \mu'_i) \frac{-1}{2} ({\sigma'}_i^2 + \epsilon)^{\frac{-3}{2}} \tag{5.9}$$

$$\frac{\partial \ell}{\partial \mu'_i} = \sum_{k \in S_i} f(x_k) \frac{\partial \ell}{\partial y'_k} \gamma \frac{-1}{\sqrt{{\sigma'}_i^2 + \epsilon}} + \frac{\partial \ell}{\partial {\sigma'}_i^2} \frac{\sum_{k \in S_i} -2 f(x_k)(x_k - \mu'_i)}{\sum_{k \in S_i} f(x_k)} \tag{5.10}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial y'_i} \gamma \frac{1}{\sqrt{{\sigma'}_i^2 + \epsilon}} + \frac{\partial \ell}{\partial {\sigma'}_i^2} \frac{2 f(x_i)(x_i - \mu'_i)}{\sum_{k \in S_i} f(x_k)} + \frac{\partial \ell}{\partial \mu'_i} \frac{1}{\sum_{k \in S_i} f(x_k)} \tag{5.11}$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{k \in S_i} f(x_k) \frac{\partial \ell}{\partial y'_k} \tag{5.12}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{k \in S_i} f(x_k) \frac{\partial \ell}{\partial y'_k} \frac{(x_k - \mu'_k)}{\sigma'_k} \tag{5.13}$$

Computation wise, the calculation of this method requires only an additional mean and variance calculation in training, which is less expensive ($\mathcal{O}(2n)$) than sorting the samples. Furthermore, the mean and variance, which can be used directly in inference can be learned and smoothed during training with moving averages through iterations as it is done in batch re-normalization. This means that in inference the application of filtered batch normalization does not have any additional computational overhead.

In training, only a minor time increase could be observed: one iteration of training with VGG-16 without batch normalization took in average $148ms$ on an NVIDIA GTX 2080 TI using Pytroch, which increased to $164ms$ with batch normalization and finally resulted $172ms$ using filtered batch normalization.

The additional hyperparameter of the algorithm T_σ can be tuned fairly easily. We were typically using values between two and seven and have not observed major changes in accuracy.

We also would like to emphasize that this approach can be used together with other methods as well, the important part is the filtering step and it can be similarly applied in batch re-normalization [113] or group normalization [111].

DOI:10.15774/PPKE.ITK.2023.005

5.4 Results

Here we will introduce our main findings on commonly applied network architectures and datasets. We will only list the most important hyperparameters of our training algorithms, but we would like to emphasize that all of our training scripts and codes are shared as supplementary material and will be shared publicly in the final version. Also in all comparisons only the normalization method was changed, all other hyperparameters from batch size to optimization algorithms remained the same.

5.4.1 MNIST

We have selected the LeNet-5 architecture and the MNIST dataset as a proof of concept to validate our method. This simple dataset allows for detailed investigations of our method using various parameters.

We have investigated the effect of T_σ parameter on the performance of our algorithm and compared it to the traditional approach without using batch normalization which we will refer to as NO-BN and also to the built-in batch normalization algorithm of Pytorch (it uses batch re-normalization and stores the momentum of the mean and variance values) which we will refer to as (BN). We have used a single batch normalization layer between the last two fully connected layers to demonstrate its effect without major modifications in the network.

The results can be seen in figure 5.4, where filtered batch normalization results higher test accuracies and faster convergence. Also one can notice that changing T_σ parameter would only affect the test results slightly. Of course with a large enough sigma which contains all outliers (e.g. $T_\sigma = 100$) we would get the original BN method back.

We have also investigated how the results depend on batch sizes and the hyperparameter T_σ. The results are depicted in Figure 5.5. As it can be seen the accuracy after a given number of training iterations depends heavily on the size of the mini-batch, but changing the parameter T_σ does not cause drastic alterations in accuracy.

In [112] a thorough investigation has been conducted searching for the reasons behind the success of BN. The authors demonstrate that inner covariate shift has

DOI:10.15774/PPKE.ITK.2023.005

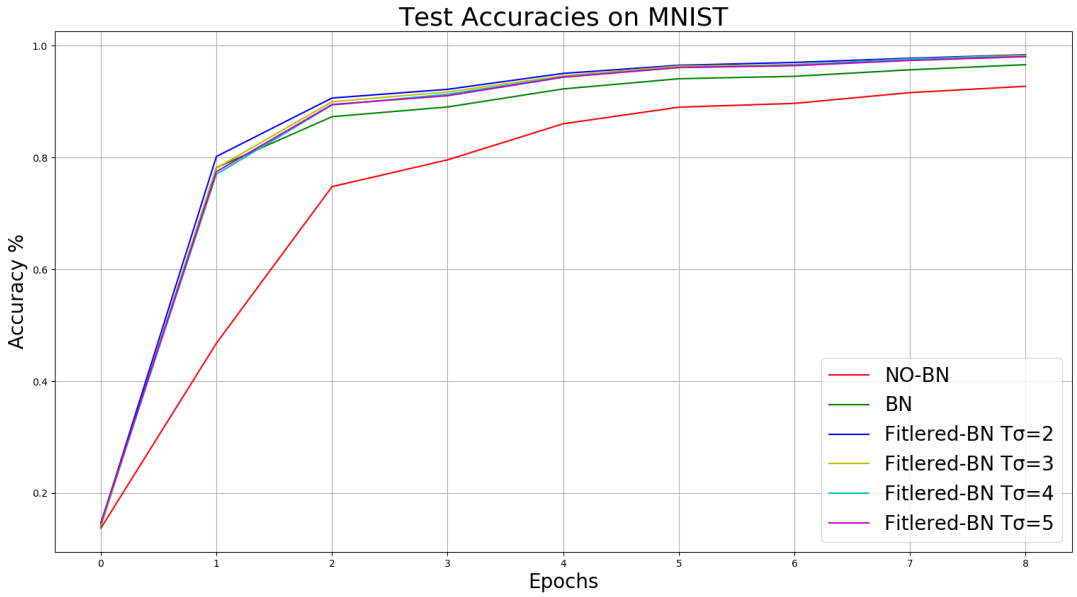

Figure 5.4: This figure depicts the test accuracies of our algorithm (Filtered-BN) with various T_σ parameters compared to the Pytorch built-in batch normalization (BN, green) and no batch normalization (NO-BN, red). Each of the displayed results is the average of ten independent trainings, which were executed with batches of 256.

DOI:10.15774/PPKE.ITK.2023.005

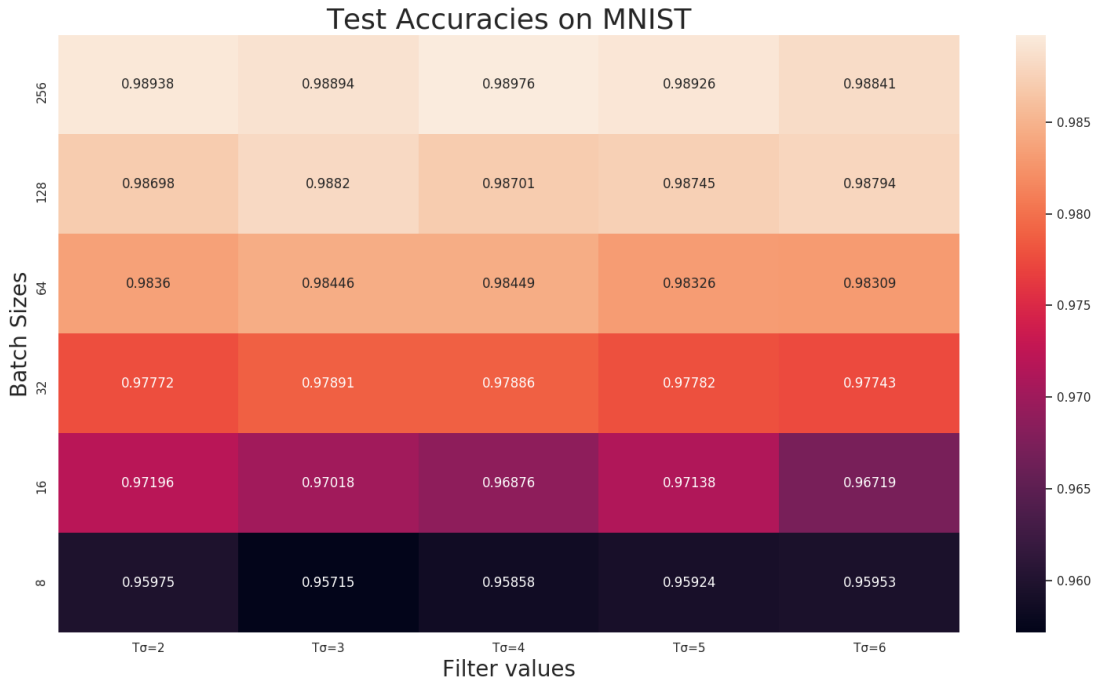

Figure 5.5: Test accuracies on the MNIST test set with different mini-batch sizes (rows) and T_σ values (columns) after 1000 iterations. As it can be seen accuracy values are fairly robust against the changes of T_σ, but depend heavily on mini-batch sizes.

DOI:10.15774/PPKE.ITK.2023.005

only a tenuous effect and BN mostly helps by creating a smooth loss and gradient landscape. To investigate this, we have also compared the loss and gradient surface of our method similarly as the authors did in [112]. The results can be seen on Figure 5.6. As we can observe, even in the case of a simple network and dataset, filtered-BN (in this experiments with $T_\sigma = 2$) results a smoother loss and gradient landscape.

5.4.2 CIFAR-10

We have also investigated the AlexNet architecture on CIFAR-10. We have examined three different architectures: the vanilla implementation of AlexNet without any normalization, one containing the built-in batch normalization after every layer (except the logit) and one containing filtered batch normalization (with $T_\sigma = 2$). All the training parameters, optimizer (gradient descent with momentum) and batch size (128) were the same.

The original 32×32 images of CIFAR-10 were rescaled to 227×227 to ensure the appropriate input dimensions for AlexNet. Ten independent runs were executed and the averaged test accuracies can be seen in figure 5.7. The network without BN achieved a test accuracy of 77% in average over 10 independent runs, with the built-in implementation of BN it achieved 82% and with filtered batch normalization the network has reached 84% accuracy.

Comparing the mean and variance values is difficult since they change between iterations because of two reasons. The difference can be caused by inner covariate shift and also by the variance of the input features in each mini-batch. Our aim was to minimize the later effect to ensure consistent moments for normalization which are close to the global mean and variance. Unfortunately, these values can not be calculated on the entire dataset during training.

To investigate the consistency of the mean and variance values of our approach which would imply a stable distribution, we have trained Alexnet using regular batch normalization with large batches (128) on CIFAR-10 (we will refer to this as BN128). In each training step, we have randomly selected a smaller mini-batch (16 samples) out of these 128 instances, then we have calculated the moments on

DOI:10.15774/PPKE.ITK.2023.005

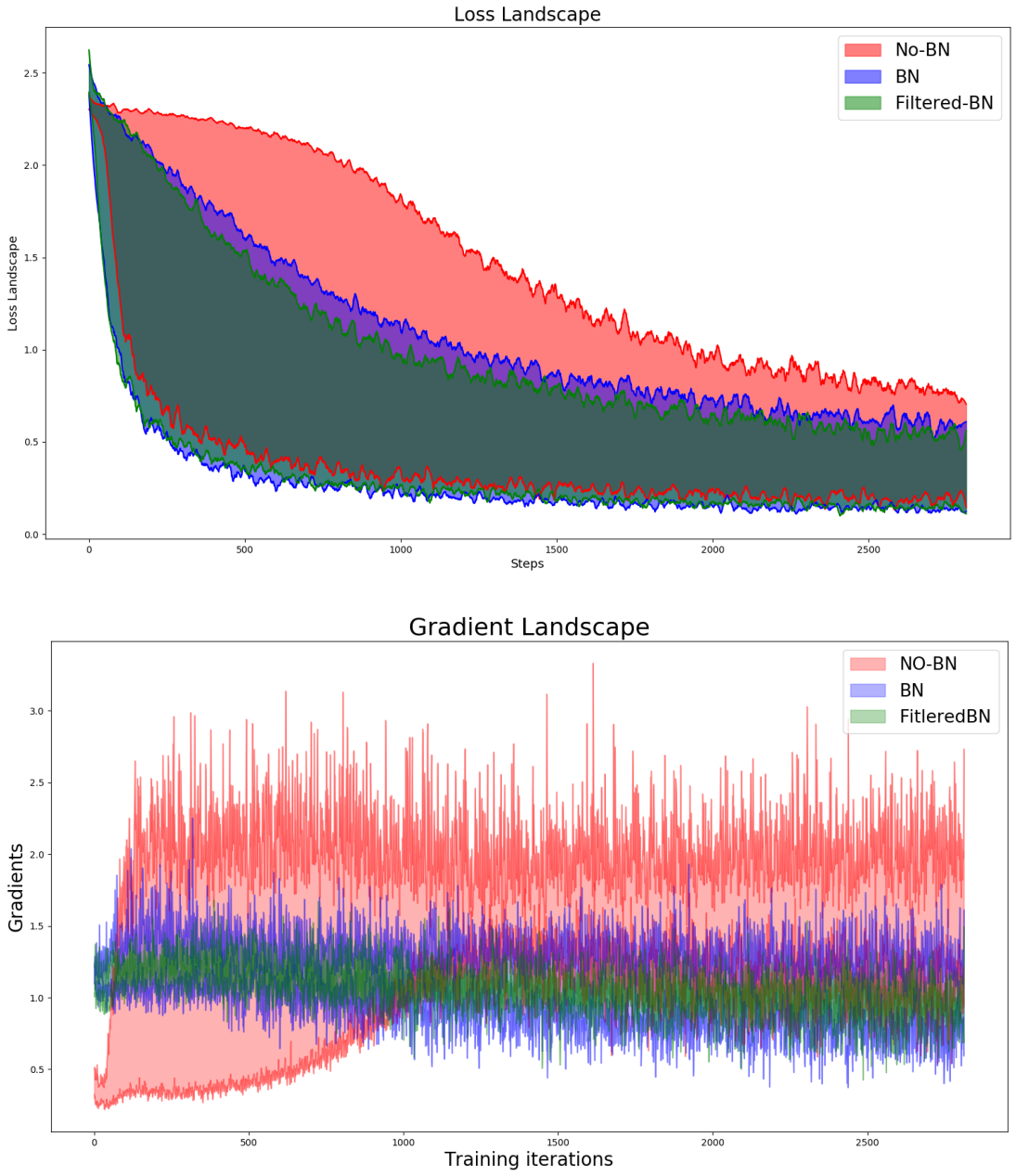

Figure 5.6: The loss and gradient landscapes for the first 3000 iterations on the MNIST dataset with batches of 64 are depicted on these figures. The left figure (Loss Landscape) displays the variance in loss as we move in the gradient direction at a particular training step. Meanwhile the right figure (Gradient Landscape) depicts the variance in ℓ_2 changes of the gradients in a similar setup. The variances were calculated between the current gradient and loss and as if a step would be made toward the gradient with step-size 0.02, 0.01, 0.005 and 0.001. As it can be seen the variance in the loss and gradient values are smaller using our method resulting smoother loss and gradient surfaces.

DOI:10.15774/PPKE.ITK.2023.005

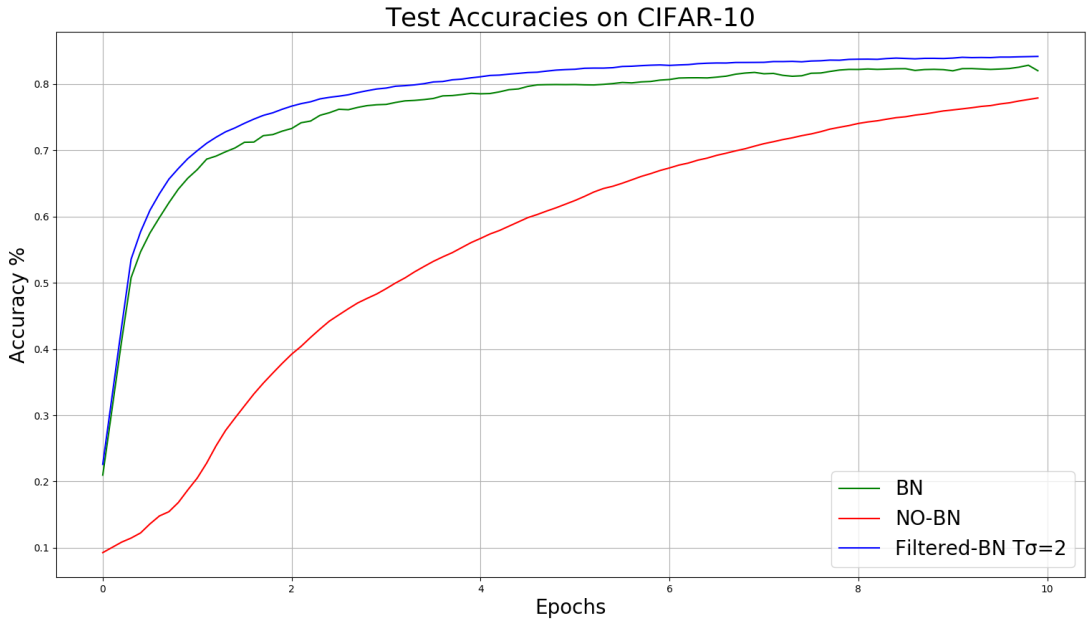

Figure 5.7: This Figure demonstrated test accuracies on CIFAR-10 with the AlexNet architecture using three different normalization methods: no normalization (NO-BN, red), traditional BN (green) and filtered BN with parameter $T_\sigma = 2$ (blue).

DOI:10.15774/PPKE.ITK.2023.005

this small mini-batch on the same network using batch normalization (we will refer to this as BN16) and filtered batch normalization (FBN16).

We have considered a larger mini-batch of 128 samples as a reference for the consistent mean and variance values[1] and compared them to the mean and variance values of the smaller mini-batch calculated by either regular BN or filtered BN. The difference between the moments of the large and the small batches using regular and filtered BN can be seen in figure 5.8.

5.4.3 ImageNet

We have also investigated the effect of filtered BN on the VGG-16 architecture on the ImageNet 2012 dataset. Normalizing layers were added after every convolution and fully connected layer (except the logit layer). Training was implemented with batches of 16 and the top-1 accuracies at different iterations on the validation set can be seen on Figure 5.9. In four million iterations the network has reached the following top-1 accuracies on the validation set: 69% with the application regular BN and 73% and 74% with the application of filtered batch normalization, with the corresponding $T_\sigma = 2$ and $T_\sigma = 4$ parameters. Meanwhile the loss landscape of VGG-16 on ImageNet with the two different normalization methods can be seen on figure 5.10.

5.4.4 Group Normalization

It was recently demonstrated that group normalization has superior performance in case of small mini-batches over BN. Group normalization uses the same method for mean and variance calculation. The only difference is the selection and grouping of the elements for normalization (S_i). In this method elements are selected across channels and positions, but from a single instance. Thus, our filtering method can be applied with group normalization after elements selection. Equation 1 and 2 of the original paper [111] can easily be changed according to the method described in Section 5.3. We will refer to the modified algorithm as filtered group normalization.

[1] Although divergence can happen even with large batch sizes but one can assume that larger values approximate the mean and variance values of the whole dataset better.

DOI:10.15774/PPKE.ITK.2023.005

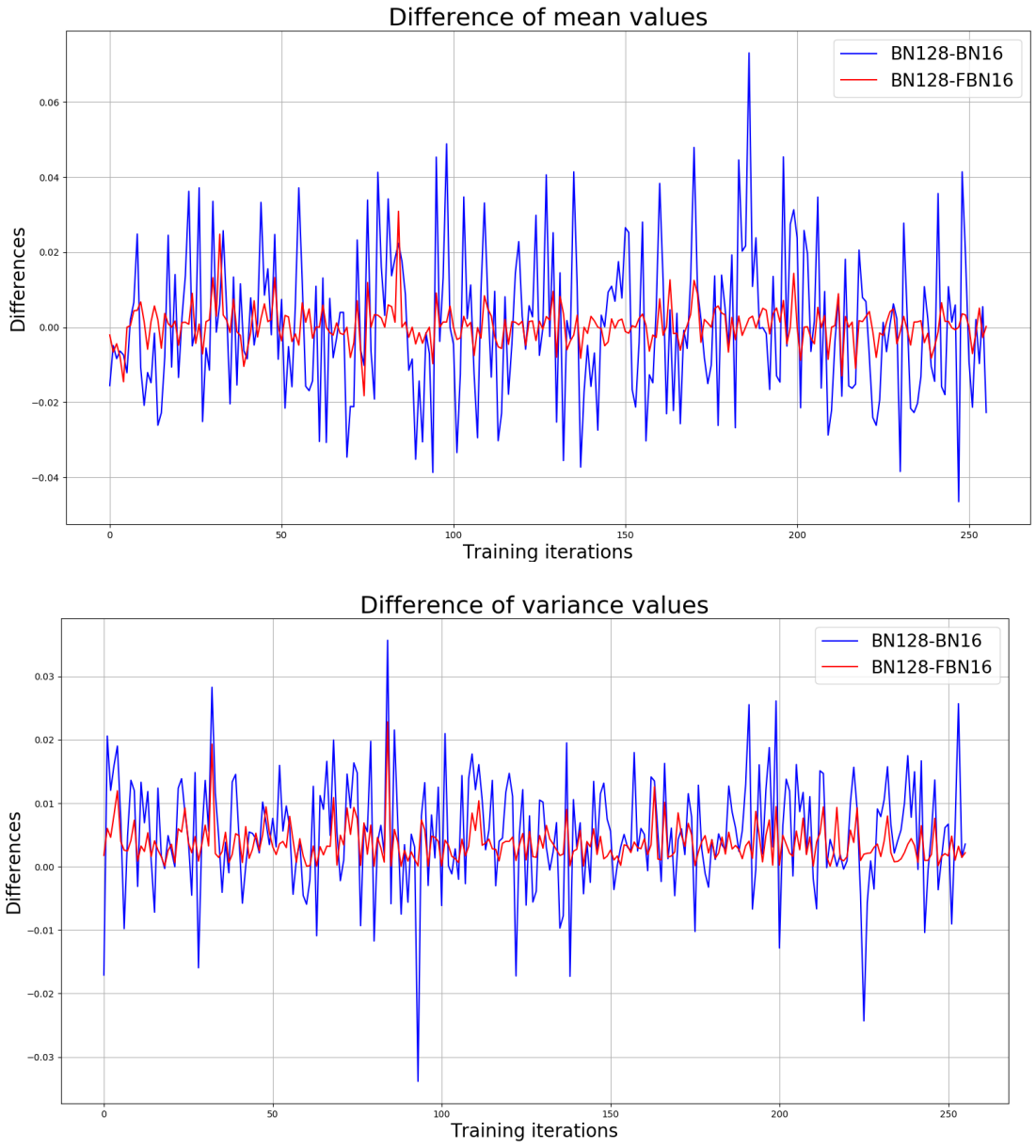

Figure 5.8: This figure depicts the mean and variance differences between traditional BN applied with mini-batches of 128 and mini-batches of 16 (blue), BN with batches of 128 and filtered BN with mini batches of 16 (red). As it can be clearly seen filtered batch normalization approximates the mean and variance values of larger batch sizes better than traditional BN.

DOI:10.15774/PPKE.ITK.2023.005

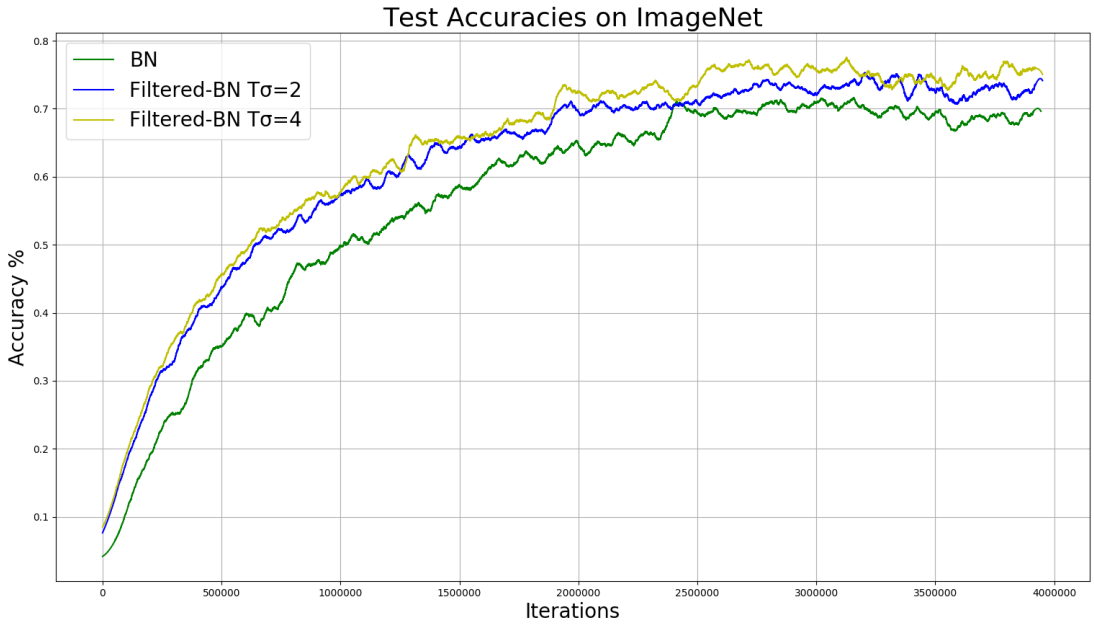

Figure 5.9: This figure depicts the accuracies on the validation set of ImageNet using the VGG-16-bn architecture with the traditional BN and filtered BN with T_σ equals two and four. As it can be seen from the results filtering out outliers in the activations has increased the convergence speed and overall accuracy of the network.

DOI:10.15774/PPKE.ITK.2023.005

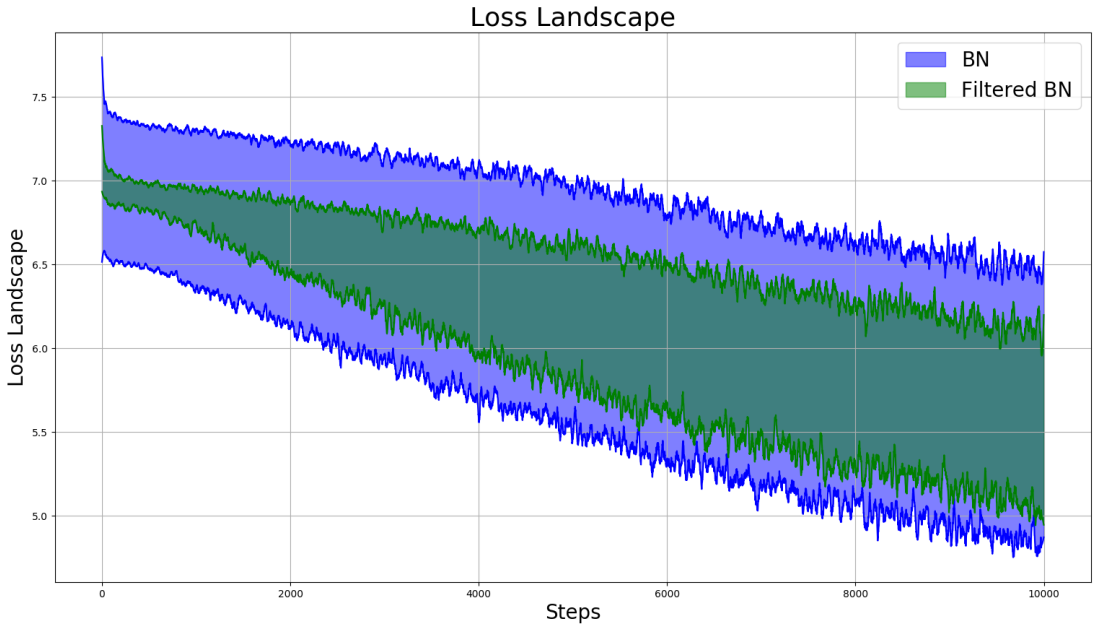

Figure 5.10: We have investigated how the loss function of the network varies in training according to training steps with different sizes (0.02, 0.01, 0.005, 0.001). These variances selecting the minimal and maximal values are depicted on this figure for regular batch normalization and filtered batch normalization with $T_\sigma = 2$ for the first 10k iterations of training with batches of 16.

DOI:10.15774/PPKE.ITK.2023.005

To compare these methods we have selected the ResNet-50 architecture and applied batch normalization, filtered batch normalization, group normalization and filtered group normalization on ImageNet with various batch sizes. The top-1 error rates on the validation set are depicted on figure 5.11. As it can be seen from the results, filtered batch normalization resulted the lowest error rate, also in case of small batches, filtered group normalization has outperformed group normalization.

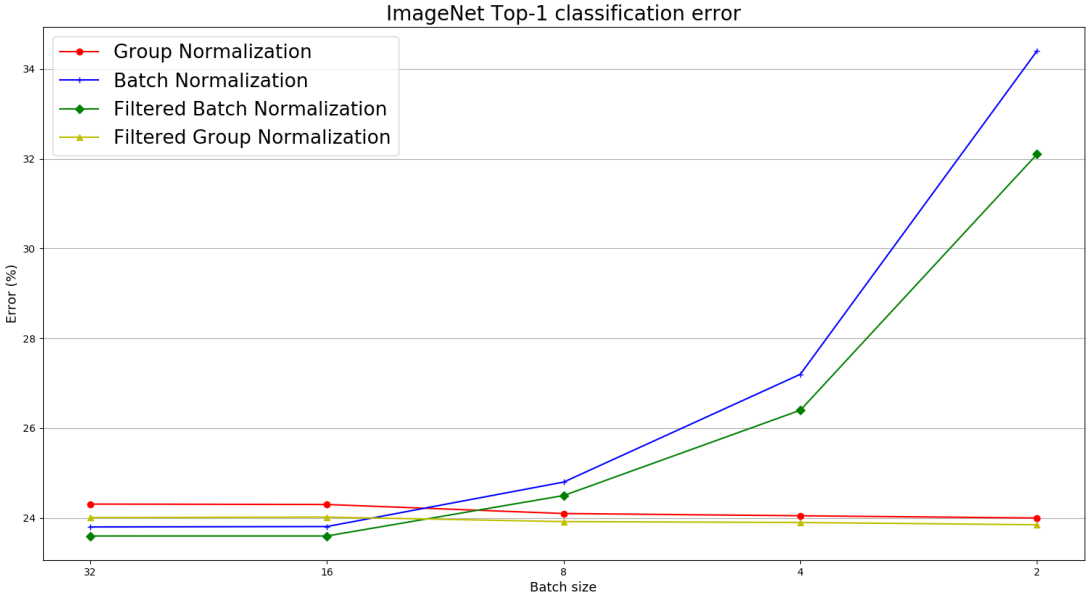

Figure 5.11: Top-1 error rate on the validation set of ImageNet using ResNet-50 and four different normalization method. As it can be seen the filtered approaches outperformed their original counterparts with all batch sizes.

5.4.5 Instance segmentation on MS-COCO

To test our method apart from classification tasks we have also applied it for instance segmentation and object localization on the MS-COCO dataset. We have selected the Detectron2 [123] framework for evaluation and used MASK R-CNN with ResNext-101 backbone with feature pyramid network. Training was executed for 270,000 iterations, with 2 images per batch and T_σ was set to five. The average

DOI:10.15774/PPKE.ITK.2023.005

precision results are displayed in Table 5.1. As it can be seen from the results, filtered batch normalization results higher AP at every iteration and also results better final accuracy after 270,000 iterations.

Table 5.1: Test accuracies for Mask-RCNN on MS-COCO on Segmentation (Seg) and object detection with bounding boxes (Box) tasks at different iterations (50000, 100000, 150000 and 270000). The columns show mean average precision at $IoU = 50 : .05 : .95$ (AP) and average precision at $IoU = 0.5$ (AP_{50}) with the traditional batch normalization (BN) and filtered batch normalization (F-BN)

	BNAP	BNAP_{50}	F-BNAP	F-BNAP_{50}
Seg(50k)	23.87	44.56	25.41	45.42
Seg(100k)	26.86	45.55	27.34	52.40
Seg(150k)	28.66	51.80	34.15	55.43
Seg(270k)	36.47	58.07	37.06	58.92
Box(50k)	23.63	41.90	27.86	47.84
Box(100k)	28.16	48.43	28.74	49.65
Box(150k)	30.53	50.79	34.24	53.13
Box(270k)	40.01	61.32	41.12	61.71

In this section we have demonstrated that the common assumption, that neural network activations of a selected layer follow Gaussian distribution is not entirely true and contradicts the specificity of neuron and convolutional kernels in deeper layers. We have shown the presence of high activations which can only be explained with unlikely low probabilities using Gaussian distribution. These, extremely out-of-distribution, seldomly occurring samples can result inconsistent mean and variance values in batch normalization.

We have introduced an algorithm, filtered batch normalization, to filter out these activations resulting faster convergence and higher overall accuracy as we have demonstrated using multiple datasets and network architectures. Our empirical results show that we can create more coherent output distributions in neural network layers by removing these outliers before mean and variance calculation, which results faster convergence and better overall validation accuracies. We also have to emphasise that comparing to batch normalization, our method adds only a minor computation overhead during training ($\sim 5\%$ in VGG-16), but does not require additional computation in inference mode.

DOI:10.15774/PPKE.ITK.2023.005

We have also showed that this normalization method results a smoother loss and gradient landscape than batch normalization. We have also demonstrated that our method can be applied with other normalization techniques as well, such as group normalization whose performance could also be increased by filtering out the out-of-distribution activations.

5.4.6 Thesis Point 4

According to the results of this chapter which were published in the International Conference on Pattern Recognition (ICPR) 2020 [124] , I formulated my fourth thesis point as the following: I Created a new method called filtered batch normalization which is a two steps batch normalization layer which filters out outlier values outside the σT range of the mean value, where T is a parameter of the algorithm. This normalization can be used with arbitrary neural network architectures to eliminate the out-of-distribution activations and by this it improved the classification accuracy of the investigated networks by 5% on ImageNet.

DOI:10.15774/PPKE.ITK.2023.005

Chapter 6

Summary

I worked on various topics utilizing neural network models investigating the possibility of solving some problems and enhancing a few shortcomings.

I have illustrated that using a gene-dependent local mutation operator where every gene has a different mutation rate induced from a heuristic and partial fitness function will speed up the convergence of the algorithm and yield more accurate final solution. I have investigated two common problems, traveling salesman problem (TSP) and N-Queens problem. In case of the N-Queens problem, Locus mutation has resulted better solutions in all cases, regardless of the investigated parameters. Even with a big population number of 254, locus mutation yields a 1.5 times lower error than its traditional counterpart. Similar results were obtained using locus mutation for the TSP problem where our approach has always surpassed the baseline solution.

I have presented a novel problem, class retrieval and the recovery from adversarial attacks along with a proposed solution, which can be used as a baseline approach in further experiments. Our retriever is a self-evident addition to adversarial attack detectors and the combination of these two methods can enable the practical applicability of deep network even in case of attacks. I investigated four different adversarial attacks (PGD, MPGD, Deepfool, TPGD and PGDDLR) on three different datasets (MNIST, CIFAR10 and ImageNet). The results are promising and consistent across all attacks and datasets, where the average accuracy is 72%, 65% and 65%, respectively. Our retrieval algorithm was not able to recover the original class in all cases but, as a preliminary concept, it clearly

DOI:10.15774/PPKE.ITK.2023.005

shows that it is possible to build an algorithm where the original class can be retrieved. I hope this can open the way for further development and fine tuning of class retrievals of adversarial attacks, which can increase the robustness of deep neural networks in real-world applications.

I have shown how a topographic metric can help in the increase of the accuracy of commonly applied image segmentation networks during training and results higher accuracy and precision in evaluation. I have shown on a simple dataset, inspired by CLEVR that the same network can achieve better accuracy and faster convergence using Wave loss, than pixel based loss functions. I have also shown on a more complex tasks, that the overall accuracy of instance segmentation could be increased by 3% on MS-COCO using the Mask-RCNN architecture, with a ResNet-101 backbone, modifying only the loss function from cross entropy to Wave loss. I have also demonstrated on the Cityscapes dataset that the inclusion of topographic information in the loss function can increase the test accuracy with 3% in average, which was observed in case of four different architectures (SegNet, DeepLab, DeepLabV3 and HRNet). These results are initial and further, detailed investigations are needed using various networks, datasets and parameter settings, but I believe they are promising and demonstrate that including topographic information in the loss calculation can result higher IOU measure in all segmentation problems.

I have introduced an algorithm, filtered batch normalization, to filter out out-of-distribution activations resulting faster convergence and higher overall accuracy as I have demonstrated using multiple datasets and network architectures. Our empirical results show that I can create more coherent output distributions in neural network layers by removing these outliers before mean and variance calculation, which results faster convergence and better overall validation accuracies. I also have to emphasise that comparing to batch normalization, our method adds only a minor computation overhead during training ($\sim 5\%$ in VGG-16), but does not require additional computation in inference mode. I have also demonstrated that our method can be applied with other normalization techniques as Ill, such as group normalization whose performance could also be increased by filtering out the out-of-distribution activations.

DOI:10.15774/PPKE.ITK.2023.005

Hopefully my thesis demonstrates that my work covers a wide range of topics presenting a small but crucial improvements in multiple areas. I hope that my work can facilitate the emergence of new applications in machine learning which can help with building a better and safer society. I will continue working on the development of machine learning and I hope that my work will further enhanced by other researchers.

DOI:10.15774/PPKE.ITK.2023.005

DOI:10.15774/PPKE.ITK.2023.005

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015. 1, 3.1, 4.1

[2] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3234–3243, 2016. 1, 3.1, 4.1

[3] N. Papernot, P. D. McDaniel, A. Sinha, and M. P. Wellman, "Towards the science of security and privacy in machine learning," *CoRR*, vol. abs/1611.03814, 2016. 1, 3.1

[4] F. Roche, T. Hueber, S. Limier, and L. Girin, "Autoencoders for music sound modeling: a comparison of linear, shallow, deep, recurrent and variational models," 2019. 1

[5] Y. Kim, J. Geng, and H. Ney, "Improving unsupervised word-by-word translation with language model and denoising autoencoder," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, (Brussels, Belgium), pp. 862–868, Association for Computational Linguistics, Oct.-Nov. 2018. 1

[6] J. Ni, Y. Chen, J. Zhu, D. Ali, and C. Weidong, "A survey on theories and applications for self-driving cars based on deep learning methods," *Applied Sciences*, vol. 10, 04 2020. 1, 3.1

DOI:10.15774/PPKE.ITK.2023.005

[7] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," 2016. 1, 3.2.1, 3.2.2, 9

[8] J. Lu, T. Issaranon, and D. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly," 2017. 1, 3.1, 3.2.2

[9] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010. 2.1

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 2.1

[11] P. Baldi, "Gradient descent learning algorithm overview: A general dynamical systems perspective," *IEEE Transactions on neural networks*, vol. 6, no. 1, pp. 182–195, 1995. 2.1

[12] Y.-A. Ma, Y. Chen, C. Jin, N. Flammarion, and M. I. Jordan, "Sampling can be faster than optimization," *arXiv preprint arXiv:1811.08413*, 2018. 2.1

[13] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, p. 4, ACM, 2015. 2.1

[14] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017. 2.1

[15] H. Talbi, M. Batouche, and A. Draa, "A quantum-inspired evolutionary algorithm for multiobjective image segmentation," *International Journal of Mathematical, Physical and Engineering Sciences*, vol. 1, no. 2, pp. 109–114, 2007. 2.1

[16] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Transactions on evolutionary computation*, vol. 9, no. 3, pp. 303–317, 2005. 2.1

DOI:10.15774/PPKE.ITK.2023.005

[17] S. Wang, Y. Wang, W. Du, F. Sun, X. Wang, C. Zhou, and Y. Liang, "A multi-approaches-guided genetic algorithm with application to operon prediction," *Artificial intelligence in medicine*, vol. 41, no. 2, pp. 151–159, 2007. 2.1

[18] K. Krawiec and M. Pawlak, "Genetic programming with alternative search drivers for detection of retinal blood vessels," in *European Conference on the Applications of Evolutionary Computation*, pp. 554–566, Springer, 2015. 2.1

[19] J. Buurman, S. Zhang, and V. Babovic, "Reducing risk through real options in systems design: the case of architecting a maritime domain protection system," *Risk Analysis: An International Journal*, vol. 29, no. 3, pp. 366–379, 2009. 2.1

[20] S. X. Zhang and V. Babovic, "An evolutionary real options framework for the design and management of projects and systems with complex real options and exercising conditions," *Decision Support Systems*, vol. 51, no. 1, pp. 119–129, 2011. 2.1

[21] D. H. Milone, J. J. Merelo, and H. Rufiner, "Evolutionary algorithm for speech segmentation," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 2, pp. 1115–1120, IEEE, 2002. 2.1

[22] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*, vol. 1, pp. 256–263, IEEE, 2000. 2.1

[23] X. Pan, J. Zhang, and K. Y. Szeto, "Application of mutation only genetic algorithm for the extraction of investment strategy in financial time series," *2005 International Conference on Neural Networks and Brain*, vol. 3, pp. 1682–1686, 2005. 2.1

[24] D. Corus and P. S. Oliveto, "Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms," *CoRR*, vol. abs/1708.01571, 2017. 2.1

DOI:10.15774/PPKE.ITK.2023.005

[25] O. Berger-Tal, J. Nathan, E. Meron, and D. Saltz, "The exploration-exploitation dilemma: a multidisciplinary framework," *PloS one*, vol. 9, no. 4, p. e95693, 2014. 2.1

[26] O. Abdoun, J. Abouchabaka, and C. Tajani, "Analyzing the performance of mutation operators to solve the travelling salesman problem," *CoRR*, vol. abs/1203.3099, 2012. 2.1

[27] A. Eiben, Z. Michalewicz, M. Schoenauer, and J. Smith, "Parameter Control in Evolutionary Algorithms," in *Parameter Setting in Evolutionary Algorithms* (F. G. Lobo, C. F. Lima, and Z. Michalewicz, eds.), vol. 54 of *Studies in Computational Intelligence*, pp. 19–46, Springer Verlag, 2007. http://www.springerlink.com/content/978-3-540-69431-1/. 2.1, 2.3

[28] B. Case and P. K. Lehre, "Self-adaptation in non-elitist evolutionary algorithms on discrete problems with unknown structure," 2020. 2.1

[29] M. Bezzel, "Proposal of 8-queens problem," *Berliner Schachzeitung*, vol. 3, no. 363, p. 1848, 1848. 2.2

[30] S. Gupta and P. Panwar, "Solving travelling salesman problem using genetic algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, pp. 376–380, 01 2013. 2.2

[31] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *Journal of heuristics*, vol. 4, no. 1, pp. 63–86, 1998. 2.2

[32] I. Korejo and S. Yang, "A comparative study of adaptive mutation operators for genetic algorithms," 06 2009. 8

[33] J. Il-Kwon and L. Ju-Jang, "Adaptive simulated annealing genetic algorithm for system identification," *Engineering Applications of Artificial Intelligence*, vol. 9, 10 1996. 2.3

[34] R. Hinterding, "Gaussian mutation and self-adaptation for numeric genetic algorithms," vol. 1, p. 384, 01 1995. 2.3

DOI:10.15774/PPKE.ITK.2023.005

[35] C.-Y. Lee and X. Yao, "Evolutionary programming using mutations based on the lÃĽvy probability distribution," *Evolutionary Computation, IEEE Transactions on*, vol. 8, pp. 1 – 13, 03 2004. 2.3

[36] T.-P. Hong, H.-S. Wang, and W.-C. Chen, "Simultaneously applying multiple mutation operators in genetic algorithms," *J. Heuristics*, vol. 6, pp. 439–455, 09 2000. 2.3

[37] Q. Fan and X. Yan, "Self-adaptive differential evolution algorithm with zoning evolution of control parameters and adaptive mutation strategies," *IEEE transactions on cybernetics*, vol. 46, no. 1, pp. 219–232, 2015. 2.3

[38] C. Li, S. Yang, and I. Korejo, "An adaptive mutation operator for particle swarm optimization," 2008. 2.3

[39] S. Yang, *Adaptive Mutation Using Statistics Mechanism for Genetic Algorithms*, pp. 19–32. 03 2004. 2.3

[40] S. Yang and A. Etaner-Uyar, "Adaptive mutation with fitness and allele distribution correlation for genetic algorithms," vol. 2, pp. 940–944, 01 2006. 2.3

[41] U. Sarkar and S. Nag, "An adaptive genetic algorithm for solving n-queens problem," *CoRR*, vol. abs/1802.02006, 2018. 2.4

[42] A. Hussain, Y. s. Muhammad, M. Nauman Sajid, I. Hussain, A. Shoukry, and S. Gani, "Genetic algorithm for traveling salesman problem with modified cycle crossover operator," *Computational Intelligence and Neuroscience*, vol. 2017, 08 2017. 2.4

[43] S. Patil and M. Bhende, "Comparison and analysis of different mutation strategies to improve the performance of genetic algorithm," 2.5.1

[44] S.-h. Zhan, J. Lin, Z.-j. Zhang, and Y.-w. Zhong, "List-based simulated annealing algorithm for traveling salesman problem," *Computational intelligence and neuroscience*, vol. 2016, 2016. 2.5.3

DOI:10.15774/PPKE.ITK.2023.005

[45] S. Hore, A. Chatterjee, and A. Dewanji, "Improving variable neighborhood search to solve the traveling salesman problem," *Applied Soft Computing*, vol. 68, pp. 83–91, 2018. 2.5.3

[46] D. Xu, T. Weise, Y. Wu, J. Lässig, and R. Chiong, "An investigation of hybrid tabu search for the traveling salesman problem," in *Bio-Inspired Computing-Theories and Applications*, pp. 523–537, Springer, 2015. 2.5.3

[47] M. A. O'Neil and M. Burtscher, "Rethinking the parallelization of random-restart hill climbing: a case study in optimizing a 2-opt tsp solver for gpu execution," in *Proceedings of the 8th workshop on general purpose processing using GPUs*, pp. 99–108, 2015. 2.5.3

[48] R. Dawkins, "The selfish gene," 1989. 2.5.4

[49] J. Al-Afandi and A. HorvÃąth, "Adaptive gene level mutation," *Algorithms*, vol. 14, no. 1, 2021. 2.5.6

[50] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013. 3.1, 3.2, 3.2

[51] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow, "Adversarial spheres," 2018. 3.1

[52] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014. 3.1, 3.2, 3.2.1

[53] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," *CoRR*, vol. abs/1610.08401, 2016. 3.1

[54] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," *CoRR*, vol. abs/1707.07397, 2017. 3.1

[55] S. Sankaranarayanan, A. Jain, R. Chellappa, and S. N. Lim, "Regularizing deep networks using efficient layerwise adversarial training," 2018. 3.1

DOI:10.15774/PPKE.ITK.2023.005

[56] A. S. Ross and F. Doshi-Velez, "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients," 2017. 3.1, 3.2.2

[57] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," 2018. 3.1, 3.2.1

[58] A. Rozsa, E. M. Rudd, and T. E. Boult, "Adversarial diversity and hard positive generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 25–32, 2016. 3.2, 3.2.1

[59] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," 2016. 3.2

[60] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1322–1333, 2015. 3.2

[61] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9185–9193, 2018. 3.2.1, 9

[62] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," *CoRR*, vol. abs/2003.01690, 2020. 3.2.1, 9

[63] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan, "Theoretically principled trade-off between robustness and accuracy," *CoRR*, vol. abs/1901.08573, 2019. 3.2.1, 9

[64] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," 2018. 3.2.1

[65] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, "A study of the effect of jpg compression on adversarial images," 2016. 3.2.2

DOI:10.15774/PPKE.ITK.2023.005

[66] Y. Luo, X. Boix, G. Roig, T. Poggio, and Q. Zhao, "Foveation-based mechanisms alleviate adversarial examples," 2016. 3.2.2

[67] J. Gao, B. Wang, Z. Lin, W. Xu, and Y. Qi, "Deepcloak: Masking deep neural network models for robustness against adversarial samples," 2017. 3.2.2

[68] X. Li and F. Li, "Adversarial examples detection in deep networks with convolutional filter statistics," 2017. 3.2.2

[69] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," 2017. 3.2.2

[70] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. Wang, "Detecting adversarial image examples in deep neural networks with adaptive noise reduction," 2019. 3.2.2

[71] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," 2017. 3.3

[72] A. Rozsa, M. Gunther, and T. E. Boult, "Towards robust deep neural networks with bang," 2018. 3.3

[73] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, P. Frossard, and S. Soatto, "Analysis of universal adversarial perturbations," 2017. 3.3

[74] F. TramÃĺr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," 2020. 3.3

[75] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2017. 9

[76] H. Kim, "Torchattacks: A pytorch repository for adversarial attacks," *arXiv preprint arXiv:2010.01950*, 2020. 9

[77] J. Al-afandi and H. András, "Class retrieval of detected adversarial attacks," *Applied Sciences*, vol. 11, no. 14, p. 6438, 2021. 3.4.6

DOI:10.15774/PPKE.ITK.2023.005

[78] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017. 4.1, 4.5.2

[79] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2980–2988, IEEE, 2017. 4.1, 5.1

[80] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE transactions on pattern analysis and machine intelligence*, 2018. 4.1

[81] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015. 4.1

[82] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning rich features from rgb-d images for object detection and segmentation," in *European Conference on Computer Vision*, pp. 345–360, Springer, 2014. 4.1

[83] Y. Zhu, Y. Tian, D. N. Metaxas, and P. Dollár, "Semantic amodal segmentation.," in *CVPR*, vol. 2, p. 7, 2017. 4.1

[84] M. Schmidt, G. Fung, and R. Rosales, "Fast optimization methods for l1 regularization: A comparative study and two new approaches," in *European Conference on Machine Learning*, pp. 286–297, Springer, 2007. 4.1

[85] X. Hu, L. Fuxin, D. Samaras, and C. Chen, "Topology-preserving deep image segmentation," *arXiv preprint arXiv:1906.05404*, 2019. 4.1

[86] J. Clough, N. Byrne, I. Oksuz, V. A. Zimmer, J. A. Schnabel, and A. King, "A topological loss function for deep-learning based image segmentation using persistent homology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 4.1

DOI:10.15774/PPKE.ITK.2023.005

[87] S. Shit, J. C. Paetzold, A. Sekuboyina, I. Ezhov, A. Unger, A. Zhylka, J. P. Pluim, U. Bauer, and B. H. Menze, "cldice-a novel topology-preserving loss function for tubular structure segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16560–16569, 2021. 4.1

[88] H. Kervadec, J. Bouchtiba, C. Desrosiers, E. Granger, J. Dolz, and I. B. Ayed, "Boundary loss for highly unbalanced segmentation," in *International conference on medical imaging with deep learning*, pp. 285–296, PMLR, 2019. 4.1

[89] T. yi Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, and C. L. Zitnick, "Microsoft coco: Common objects in context," 2014. 4.1, 4.5.3

[90] R. W. Hamming, "Error detecting and error correcting codes," *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950. 4.2

[91] J. Henrikson, "Completeness and total boundedness of the hausdorff metric," *MIT Undergraduate Journal of Mathematics*, vol. 1, pp. 69–80, 1999. 4.2, 4.2

[92] R. Zhao, B. Qian, X. Zhang, Y. Li, R. Wei, Y. Liu, and Y. Pan, "Rethinking dice loss for medical image segmentation," in *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 851–860, IEEE, 2020. 4.2, 4.5.4

[93] M. Berman, A. R. Triki, and M. B. Blaschko, "The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4413–4421, 2018. 4.2

[94] N. Abraham and N. M. Khan, "A novel focal tversky loss function with improved attention u-net for lesion segmentation," in *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pp. 683–687, IEEE, 2019. 4.2

DOI:10.15774/PPKE.ITK.2023.005

[95] I. Szatmári, C. Rekeczky, and T. Roska, "A nonlinear wave metric and its cnn implementation for object classification," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 23, no. 2-3, pp. 437–447, 1999. 4.2.1

[96] T. Roska and L. O. Chua, "The cnn universal machine: an analogic array computer," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 40, no. 3, pp. 163–173, 1993. 4.2.1

[97] J. Al-Afandi and A. Horvath, "Application of the nonlinear wave metric for image segmentation in neural networks," in *CNNA 2018; The 16th International Workshop on Cellular Nanoscale Networks and their Applications*, pp. 1–4, VDE, 2018. 4.2.1

[98] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998. 4.4

[99] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 1988–1997, IEEE, 2017. 4.4

[100] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016. 4.5.2

[101] B. Cheng, B. Xiao, J. Wang, H. Shi, T. S. Huang, and L. Zhang, "Higherhrnet: Scale-aware representation learning for bottom-up human pose estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5386–5395, 2020. 4.5.2

[102] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014. 4.5.2

DOI:10.15774/PPKE.ITK.2023.005

[103] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017. 4.5.2

[104] C. Wang, Y. Zhang, M. Cui, J. Liu, P. Ren, Y. Yang, X. Xie, X. Hua, H. Bao, and W. Xu, "Active boundary loss for semantic segmentation," *arXiv preprint arXiv:2102.02696*, 2021. 4.5.4

[105] S. Al Arif, K. Knapp, and G. Slabaugh, "Shape-aware deep convolutional neural network for vertebrae segmentation," in *International Workshop on Computational Methods and Clinical Applications in Musculoskeletal Imaging*, pp. 12–24, Springer, 2017. 4.5.4

[106] Ã. KovÃącs, J. Al-Afandi, C. Botos, and A. HorvÃąth, "Wave loss: A topographic metric for image segmentation," *Mathematics*, vol. 10, no. 11, 2022. 4.5.5

[107] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012. 5.1

[108] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016. 5.1

[109] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016. 5.1

[110] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015. 5.1

[111] Y. Wu and K. He, "Group normalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018. 5.1, 5.2.1, 5.3, 5.4.4

DOI:10.15774/PPKE.ITK.2023.005

[112] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," in *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018. 5.1, 5.4.1

[113] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," in *Advances in neural information processing systems*, pp. 1945–1953, 2017. 5.1, 5.3

[114] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 5.1

[115] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017. 5.1

[116] S. Singh and S. Krishnan, "Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks," *arXiv preprint arXiv:1911.09737*, 2019. 5.1

[117] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, vol. 3, no. 3, p. e10, 2018. 5.2.2

[118] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Towards better understanding of gradient-based attribution methods for deep neural networks," *arXiv preprint arXiv:1711.06104*, 2017. 5.2.2

[119] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 5.2.2

[120] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L. Fei-Fei, "Imagenet large scale visual recognition competition 2012 (ilsvrc2012)," *See net. org/challenges/LSVRC*, p. 41, 2012. 5.2.2

DOI:10.15774/PPKE.ITK.2023.005

[121] K.-H. Yuan and P. M. Bentler, "Robust mean and covariance structure analysis," *British Journal of Mathematical and Statistical Psychology*, vol. 51, no. 1, pp. 63–88, 1998. 5.3

[122] P. Kokic and P. Bell, "Optimal winsorizing cutoffs for a stratified finite population estimator," *Journal of Official Statistics*, vol. 10, no. 4, p. 419, 1994. 5.3

[123] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," 2019. 5.4.5

[124] A. Horváth and J. Al-Afandi, "Filtered batch normalization," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 6778–6785, IEEE, 2021. 5.4.6