GENETIC ALGORITHM AND BINARY MASKS FOR CO-LEARNING MULTIPLE DATASETS IN DEEP NEURAL NETWORKS

DERİN SİNİR AĞLARINDA ÇOKLU VERİ SETLERİNİN BİRLİKTE ÖĞRENİLMESİ İÇİN GENETİK ALGORİTMA VE İKİLİ MASKELER

ÖZNUR TURAN

PROF. DR. MEHMET ÖNDER EFE

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

January 2024

ABSTRACT

GENETIC ALGORITHM AND BINARY MASKS FOR CO-LEARNING MULTIPLE DATASETS IN DEEP NEURAL NETWORKS

Öznur Turan

Master of Science, Computer Engineering

Supervisor: Prof. Dr. Mehmet Önder EFE

January 2024, 93 pages

This study addresses the challenges of 'catastrophic forgetting' and 'multi-task learning' encountered in the field of data classification and analysis, particularly with the use of Convolutional Neural Networks (CNNs). The aim of the study is to use genetic algorithms to solve these problems. Methodologically, an optimization strategy has been developed that employs layer-based binary masks to customize CNNs models for multiple datasets. Genetic algorithms have been utilized as a heuristic search method to optimize a binary mask for each dataset. Experiments have been conducted on widely-used dataset such as MNIST, Fashion MNIST, and KMNIST. The obtained results are notably impressive, with classification accuracies of 76.25% for MNIST, 76% for Fashion MNIST, and 74.43% for KMNIST. These findings demonstrate that the proposed approach can create high-performance models not only for a single task but also for multiple tasks.

Keywords: Convolutional Neural Networks, Catastrophic Forgetting, Multi-Task Learning, Genetic Algorithms, Binary Masks

ÖZET

DERİN SİNİR AĞLARINDA ÇOKLU VERİ SETLERİNİN BİRLİKTE ÖĞRENİLMESİ İÇİN GENETİK ALGORİTMA VE İKİLİ MASKELER

Öznur Turan

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Prof. Dr. Mehmet Önder EFE

Ocak 2024, 93 sayfa

Bu çalışma, veri sınıflandırma ve analizi alanında, özellikle Evrişimli Sinir Ağları (CNN'ler) kullanımında karşılaşılan 'felaket unutma' ve 'çoklu görev öğrenme' zorluklarını ele almaktadır. Çalışmanın amacı, bu sorunları genetik algoritmalar kullanarak çözmektir. Yöntemsel olarak, birden fazla veri seti için CNN modellerini özelleştirmek için katman bazlı ikili maskeler kullanan bir optimizasyon stratejisi geliştirilmiştir. Genetik algoritmalar, her veri seti için ikili bir maske optimize etmek amacıyla sezgisel bir arama yöntemi olarak kullanılmıştır. MNIST, Fashion MNIST ve KMNIST gibi yaygın kullanılan veri setlerinde deneyler yapılmıştır. Elde edilen sonuçlar oldukça etkileyicidir; MNIST için %76.25, Fashion MNIST için %76 ve KMNIST için %74.43 sınıflandırma doğruluğu elde edilmiştir. Bu bulgular, önerilen yaklaşımın sadece tek bir görev için değil, birden çok görev için de yüksek performanslı modeller oluşturabileceğini göstermektedir.

Keywords: Evrişimli Sinir Ağları, Yıkıcı Unutma, Çoklu Görev Öğrenme, Genetik Algoritmalar, İkili Maskeler

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my esteemed advisor, Prof. Dr. Mehmet Önder EFE, for their patience, knowledge, and experience they shared with me during this process. Their contributions have significantly enriched this thesis, making it much more meaningful and valuable.

I would also like to express my heartfelt thanks to my family; I am grateful to my mother, father, and siblings for their support throughout this process. Their presence and love gave me the strength and inspiration I needed to succeed.

Öznur TURAN

CONTENTS

Page

TABLES

FIGURES

ABBREVIATIONS

ADAM	:	**A**daptive **D**ecrease of **A**cceleration **M**ethod
CNNs	:	**C**onvolution **N**eural **N**etworks
Conv	:	**Conv**olutional
Fashion MNIST	:	**Fashion M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology Database
FMNIST	:	**F**ashion **MNIST**
GA	:	**G**enetic **A**lgorithm
KMNIST	:	**K**uzushiji **M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology Database
MNIST	:	**M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology Database
ReLU	:	**R**ectified **L**inear **U**nit

1. INTRODUCTION

In the constantly evolving landscape of machine learning, CNNs have emerged as a cornerstone technology, revolutionizing applications in data classification and analysis. Despite their widespread success, CNNs face significant challenges, particularly in areas of 'catastrophic forgetting' and 'multi-task learning'. Catastrophic forgetting, a phenomenon where a network loses previously acquired knowledge upon learning new information, severely limits the adaptability of CNNs in dynamic environments. Similarly, multi-task learning, which involves simultaneous learning of multiple related tasks, poses a challenge in efficiently balancing and optimizing performance across different tasks.

The motivation behind this thesis springs from these critical challenges. By integrating genetic algorithms into the fabric of CNNs, this research aims to address these issues, proposing a novel approach to enhance the efficiency and adaptability of these networks. The use of genetic algorithms, known for their robust search capabilities in complex spaces, provides a promising avenue for optimizing CNNs for specific tasks while retaining learned knowledge from previous tasks.

This thesis sets forth a comprehensive study of this innovative approach. The scope of this research encompasses the development of a dynamic learning strategy, an exploration of layer-based binary masks for CNN customization, and an extensive series of experiments across widely recognized datasets. Through this exploration, the thesis contributes to the field of machine learning by offering novel solutions to the persistent problems of catastrophic forgetting and multi-task learning in CNNs.

The following chapters lay out the foundation and detailed examination of the proposed approach. Chapter 2 provides an extensive background overview, setting the stage for understanding the complexities of CNNs and the challenges at hand. Chapter 3 delves into related work, offering a critical perspective on existing methodologies and their limitations. Chapter 4 introduces our novel approach, describing the unique application of genetic algorithms in optimizing CNNs for multi-task learning and mitigating catastrophic

forgetting. This chapter outlines the methodology, the design of binary masks, and the specifics of our dynamic learning strategy. Chapter 5 showcases the experimental results, detailing the performance of the optimized CNNs on datasets like MNIST, Fashion MNIST, and KMNIST. Chapter 6 concludes the thesis and discussing the implications of this research in the field of deep learning and suggesting directions for future research.

1.1. Scope Of The Thesis

This thesis primarily focuses on addressing the critical challenges of 'catastrophic forgetting' and 'multi-task learning' in the realm of data classification and analysis, with a special emphasis on CNNs. The overarching objective of this research is to leverage genetic algorithms as a novel solution to these persistent issues.

Methodologically, this thesis introduces an innovative optimization strategy, employing layer-based binary masks to tailor CNN models for optimal performance across multiple datasets. The use of genetic algorithms as a heuristic search method plays a pivotal role in this process, facilitating the optimization of a binary mask for each dataset under consideration.

A series of comprehensive experiments forms the backbone of this research, utilizing widely-acknowledged datasets such as MNIST, Fashion MNIST, and KMNIST. These experiments are meticulously designed to evaluate the success of the proposed method. The results are significant, showcasing classification accuracies of 76.25% for MNIST, 76% for Fashion MNIST, and 74.43% for KMNIST.

These findings represent a significant breakthrough in the field and demonstrate that the proposed approach is capable of developing high-performance models that can successfully execute not just a single task, but multiple tasks simultaneously. This thesis contributes to the broader field of machine learning by providing a novel perspective and approach to overcoming some of the most challenging hurdles in data classification and analysis.

1.2. Contributions

This research aims to address existing deficiencies in the field of data classification and analysis, especially in the context of CNNs and their susceptibility to 'catastrophic forgetting' and challenges in 'multi-task learning'. The main contributions of this paper can be summarized as follows:

- Introduction of a Dynamic Learning Strategy: We propose a dynamic learning approach that adapts to different datasets in real-time. This method enhances the efficiency of CNNs by allowing them to learn at varying rates depending on the complexity and nature of the task at hand. This contrasts with traditional static learning rates, offering a more flexible and responsive learning mechanism.
- A Novel Perspective on Multi-Task Learning: Unlike most of the previous works which primarily focus on sequential or parallel learning strategies, our approach integrates these methodologies, enabling simultaneous learning across multiple tasks. This integration not only improves learning efficiency but also significantly reduces the occurrence of catastrophic forgetting.
- Pioneering Use of Genetic Algorithms for CNN Optimization: We employ genetic algorithms in optimizing layer-based binary masks for CNNs. This innovative application demonstrates a new horizon in the use of evolutionary computation for neural network optimization, particularly in addressing the nuances of multi-task learning and memory retention.
- Empirical Evidence Through Simulation Results: Our simulation results show impressive performance metrics, with notable classification accuracies across different datasets (76.25% for MNIST, 76% for Fashion MNIST, and 74.43% for KMNIST). These results validate the effectiveness of the proposed methods.

Overall, these contributions mark significant advancements in machine learning, particularly in enhancing the capabilities of CNNs in complex data classification and analysis tasks.

This research opens new pathways for future studies and applications in the domain of deep learning and neural network optimization.

1.3. Organization

The organization of the thesis is as follows:

- Chapter 1 presents our motivation for addressing the challenges in data classification and analysis using CNNs, specifically focusing on catastrophic forgetting and multi-task learning. It outlines the key contributions of this thesis and sets the scope, detailing the innovative approach employed and the significance of the research in the broader context of machine learning and neural networks.

- Chapter 2 provides a comprehensive background overview of the fundamental concepts and theoretical underpinnings relevant to this research. It delves into the intricacies of CNNs, genetic algorithms, and the prevalent challenges in the field, such as catastrophic forgetting and multi-task learning. This chapter aims to establish a solid foundation of understanding for the subsequent chapters.

- Chapter 3 gives a detailed review of the existing literature and previous studies in the realm of CNNs, catastrophic forgetting, multi-task learning, and genetic algorithm applications in neural network optimization. This chapter critically analyzes the strengths and limitations of current methods and identifies gaps that this research aims to fill.

- Chapter 4 introduces the methodology employed in this thesis. It details the innovative approach of using genetic algorithms for optimizing layer-based binary masks in CNNs and the dynamic learning strategy. This chapter elaborates on the theoretical aspects of the method and its expected advantages over existing techniques.

- Chapter 5 demonstrates the empirical findings of the research. It presents a thorough analysis of the experiments conducted using datasets like MNIST, Fashion MNIST,

and KMNIST. The chapter discusses the significance of the results obtained, including the achieved classification accuracies, and evaluates the effectiveness of the proposed methods in the context of the objectives set out in this study.

- Chapter 6 states the summary of the thesis and possible future directions.

2. BACKGROUND OVERVIEW

2.1. Genetic Algorithm

GA are computer science algorithms developed by drawing inspiration from the fundamental principles of the natural evolutionary process, used for solution searching and optimization problems. These algorithms mimic the main elements of biological evolution: selection, crossover, and mutation. GA evolves the solutions of various problems within a population. Solutions are modeled as a chromosome sequence that represents an individual and carries genetic information. The algorithm iteratively develops this solution population, selecting the most suitable solutions, applying crossover, and performing random mutations in each iteration. This process continues until a certain fitness criterion or a number of iterations is reached. GA offers an effective alternative, especially in problems where the search space is large and complex and where traditional optimization methods fail.

2.1.1. Inspiration from Natural Evolution

2.1.1.1. Biological Analogy

GA draw direct inspiration from theory of natural selection. In nature, organisms evolve over generations to adapt to their environment. The fittest individuals are more likely to reproduce, passing on their genes to the next generation. GA use this concept of survival of the fittest to find optimal solutions to a problem.

2.1.1.2. Chromosomes and Genes

In GA, potential solutions to a problem are encoded as a set of 'chromosomes,' similar to the genetic makeup in organisms. Each chromosome is made up of 'genes,' which are essentially parameters of the solution.

2.1.2. The Process of Genetic Algorithm

The process of a genetic algorithm is a complex and adaptable computational method, capable of tackling a wide range of problems. Its strength lies in mimicking natural evolutionary strategies, allowing it to explore vast search spaces and converge on optimal or near-optimal solutions efficiently. The flexibility in its implementation and the ability to tailor its components to specific problems make it a powerful tool in the area of modern computational techniques. Figure 2.1 depicts the schematic representation of the process:

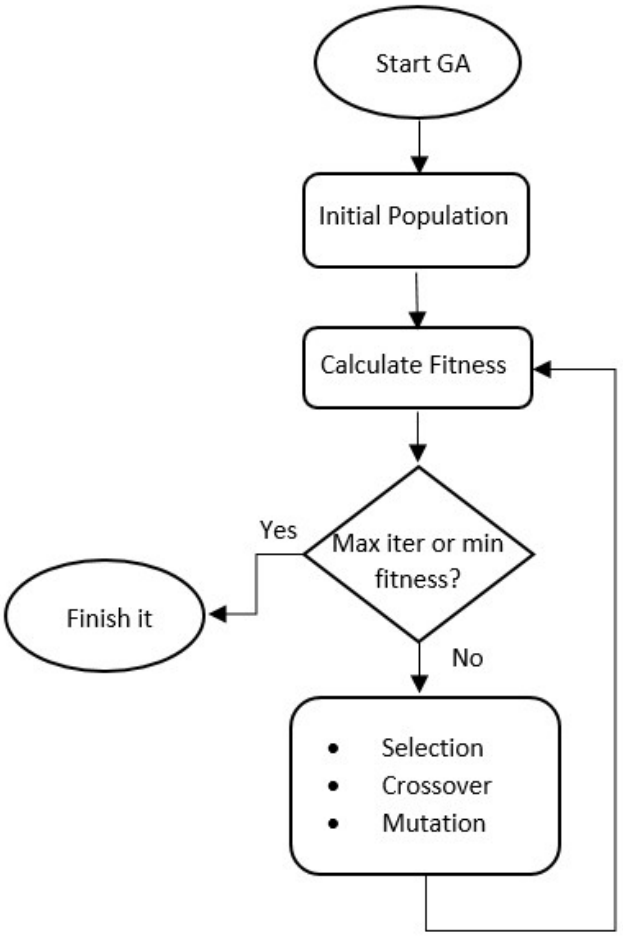

Figure 2.1 The General Flowchart of a GA

2.1.2.1. Initialization

Initialization is the first step in the genetic algorithm (GA) process, where the initial population of potential solutions is created. This stage is crucial as it lays the foundation for the evolutionary search process. A well-chosen initial population can significantly enhance the efficiency and effectiveness of the GA. The details of this stage are as follows:

Population Generation: The initial population, comprising multiple solutions (chromosomes), is generated randomly or using heuristic information. Each chromosome represents a potential solution to the problem.

Population Size: The size of the initial population can greatly influence the GA's performance. A larger population provides greater genetic diversity, enhancing the exploration of the solution space. However, a larger population also means more computational effort per generation. Hence, it's important to find a balance based on the problem's complexity and available computational resources.

Encoding Scheme: The method of representing solutions is crucial. Encoding schemes depends on the problem's nature.

- Chromosome Representation: Deciding how to represent potential solutions is a critical aspect of initialization. Common encoding schemes include binary, integer, real-valued, or even more complex structures.
- Problem-Specific Design: The encoding scheme should be designed to reflect the nature of the problem effectively. For instance, binary encoding is common in simple optimization problems, while permutation encoding is used in routing or scheduling problems.

Initialization Methods:

- Random Initialization: The most straightforward method is to generate the initial population randomly. This ensures a diverse set of starting points for the evolutionary process. However random initialization, while simple, might not be efficient for all problems, especially those with a vast search space or specific constraints.
- Heuristic-Based Initialization: Incorporating Problem Knowledge: Using heuristic methods or domain-specific knowledge to generate the initial population can lead to a more focused search. For example, in a routing problem, initial routes might be generated based on the nearest-neighbor heuristic rather than purely randomly. This can speed up convergence, especially in complex or highly constrained problems.
- Hybrid Initialization: Often, a combination of random and heuristic-based methods can be used. Part of the population is generated randomly, and the rest is created using problem-specific heuristics. This approach balances exploration and exploitation from the onset of the algorithm.

Diversity Considerations:

- Maintaining Diversity: It's important to ensure that the initial population covers a wide range of the solution space. Lack of diversity can lead to premature convergence.
- Techniques: Techniques like ensuring a minimum distance between initial solutions or varying the parameters in heuristic methods can be employed to maintain diversity.

Initialization in genetic algorithms is more than just a random starting point. It is a critical phase where the foundation for the evolutionary process is laid. The choice of population size, encoding scheme, and initialization method should be tailored to the specific characteristics of the problem at hand. A well-considered initialization strategy can importantly improve the effectiveness of the genetic algorithm, leading to faster convergence and better solutions.

2.1.2.2. Fitness Evaluation

Fitness calculation is a pivotal component in GA, serving as the measure for evaluating how well a given solution (chromosome) solves the problem at hand. The fitness function assigns a score to each solution, guiding the selection process towards more promising regions of the solution space. The different aspects and methods of fitness calculation are examined below.

Defining Fitness Function: The fitness function quantitatively evaluates how close a solution is to meeting the objectives. It varies significantly depending on the specific problem being solved.

- Alignment with Objectives: The fitness function should be closely aligned with the problem's objectives. In optimization problems, it often directly represents the objective to be maximized or minimized.
- Complexity and Efficiency: While the function needs to be comprehensive enough to evaluate solutions accurately, it should also be computationally efficient to allow for quick evaluations.

Types of Fitness Functions:

- Simple Mathematical Functions: For basic optimization problems, the fitness function might be a simple mathematical equation representing the problem's objective. Typically employed in straightforward optimization tasks where the goal is clear and quantifiable. Examples:
  - Maximization/Minimization: Functions like linear equations, quadratic functions, or more complex algebraic expressions where the objective is to find the maximum or minimum value.

  - Benchmark Functions: Standard test functions used in optimization literature, like the Rastrigin, Rosenbrock, or Ackley functions, which are known for specific challenges they pose (e.g., local minima, steepness).

- Simulation-Based Evaluation: In more complex scenarios, like evolutionary robotics, fitness might be determined based on the performance of a solution in a simulated environment. Examples:

  - Evolutionary Robotics: Fitness might be determined by how well a robot navigates a terrain or accomplishes a task in a simulated environment.
  - Game Strategies: In designing AI for games, fitness could be based on the AI's performance in simulated game scenarios.

- Problem-Specific Functions: In domain-specific problems, fitness functions can include various parameters and criteria unique to that domain, such as distance in routing problems or profit in investment strategies. Examples:

  - Routing Problems: In problems like the Traveling Salesman Problem, fitness could be the inverse of the total distance traveled.
  - Financial Models: For investment or portfolio optimization, fitness might be based on the return on investment or risk-adjusted return.
  - Machine Learning Models: In evolving neural networks, fitness could be based on accuracy, precision, recall, or loss functions like mean squared error or cross-entropy.

- Dynamic Fitness Functions: In dynamic optimization problems, where the objective changes over time, fitness functions can adapt to these changes, ensuring the GA remains effective throughout the process. Examples:

  - Adaptive Control Systems: Fitness functions that adjust according to varying control objectives or environmental conditions.

  - Real-time Strategy Games: Where the objectives might shift based on the evolving state of the game.
  - Changing Market Conditions: In financial applications, where the fitness function adapts to fluctuating market dynamics.

Handling Constraints: In problems with constraints, penalty functions are often used to reduce the fitness of solutions that violate these constraints.

- Penalty Methods: For problems with constraints, fitness functions often incorporate penalty terms that decrease the fitness of solutions violating these constraints.
- Feasibility-Based Methods: Another approach is to separate feasible and infeasible solutions, where only feasible solutions are considered for reproduction.
- Noise and Uncertainty: In real-world problems, fitness evaluations may include noise or uncertainty. Robustness to these factors is often an important consideration in designing the fitness function.

Normalization and Scaling:

- Normalization: In multi-objective optimization, different objectives might be scaled to comparable ranges to prevent any one objective from dominating the fitness evaluation.
- Fitness Scaling: Techniques like rank-based scaling or sigma scaling are used to adjust fitness values, especially in cases where raw fitness scores vary widely. This helps maintain a healthy selection pressure.

Fitness calculation in genetic algorithms is a nuanced process that requires careful consideration and design. The fitness function must accurately reflect the objectives and constraints of the problem, be computationally efficient, and be capable of guiding the GA towards optimal or near-optimal solutions. Whether dealing with simple mathematical

problems, complex simulations, or real-world scenarios with multiple objectives and constraints, the fitness function plays a crucial role in the success of a genetic algorithm. The choice of fitness calculation method can importantly impact the performance, making it a critical area of focus in GA design and implementation.

2.1.2.3. Selection

This process mimics natural selection, where the fittest individuals are selected to form new individuals. In GA, the fitness of each solution is evaluated based on a predefined criterion, and the best-performing solutions are selected for the next phase. Solutions with higher fitness are more likely to be selected for reproduction. Selection in GA is a critical step that significantly influences the algorithm's convergence and performance. Various selection methods exist, like roulette wheel selection, tournament selection, etc. Here, we delve deeper into various selection methods, each with its unique approach and implications for the evolutionary process.

Roulette Wheel Selection: This is a method where each slice corresponds to an individual in the population, like a roulette wheel. The size of each slice is proportional to the individual's fitness. Higher fitness equates to a larger slice.

A random number determines which slice the wheel lands on, thus selecting the individual. This process is repeated to select multiple individuals.

Favors fitter individuals but still gives a chance to less fit ones, maintaining diversity. Can be problematic if the fitness values vary greatly (fitness scaling may be necessary). It might also lead to premature convergence if one or a few individuals dominate early.

Tournament Selection: A set number of individuals are chosen randomly from the population, and a 'tournament' is held among them.

The individual with the highest fitness in this subset is selected. This process is repeated multiple times to select the required number of individuals.

The size of the tournament can be adjusted. Larger tournaments tend to favor fitter individuals, while smaller ones maintain more diversity.

Less susceptible to the takeover by dominant individuals compared to roulette wheel selection. Generally computationally efficient more than roulette wheel selection.

Rank-Based Selection: Instead of selecting based on fitness, individuals are ranked according to their fitness.

The selection probability is then assigned based on this rank. The highest-ranked individual has the highest chance of being selected, but unlike fitness-proportionate selection, the exact fitness values are not considered.

Reduces the problems caused by a few super-fit individuals. It is beneficial in situations where the fitness scale causes issues in selection. Helps in maintaining diversity over generations, as the relative difference between individuals is more controlled.

Stochastic Universal Sampling: A variation of roulette wheel selection, designed to provide more balanced selection pressure.

Multiple pointers are spaced evenly around the wheel, and the wheel is spun once. Each individual selected by a pointer is chosen for the next generation.

This method reduces the chance fluctuations that can occur in roulette wheel selection, providing a more representative sample of the population.

Elitism: Sometimes, the best-performing chromosomes are carried over to the next generation unchanged to ensure that the quality of solutions does not degrade. A certain number of the fittest individuals from the current generation are guaranteed a place in the next generation. These individuals are copied directly without undergoing crossover or mutation. Ensures that the best solutions found so far are not lost, which can accelerate

the convergence of the GA. It must be used carefully to avoid premature convergence and to maintain sufficient genetic diversity.

The selection mechanism in genetic algorithms plays a pivotal role in guiding the evolutionary process. The choice of selection method can significantly impact the balance between exploration and exploitation within the GA. Each method has its strengths and weaknesses, and the choice often depends on the specific requirements and characteristics of the problem at hand. By understanding these nuances, the genetic algorithm can be effectively tailored to achieve optimal performance.

2.1.2.4. Crossover

Crossover, also known as recombination, is a fundamental genetic operator in GA. It simulates the biological process of reproduction, where offspring inherit a combination of genes from their parents. The process enables the GA to explore new regions of the solution space by combining existing solutions in novel ways. This step involves combining parts of two or more selected chromosomes to create new offspring. Selected solutions undergo crossover to produce new offspring is applied to introduce variability. It's akin to biological reproduction, where offspring inherit traits from both parents. The hope is that by combining good traits from different solutions, even better solutions can be created. Here, we delve into various crossover methods, their mechanisms, and their advantages and disadvantages.

Single-Point Crossover: First, a random crossover point is selected. Segments after this selected point are exchanged between parent chromosomes to create offspring. It is advantageous in terms of simplicity and ease of implementation. However, it can lead to limited exploration of the solution space, as only the genes at the end of the chromosome have a high chance of being swapped.

Two-Point or Multi-Point Crossover: Similar to single-point crossover, but differs in having two or more crossover points. The segments between these points are swapped

between the parents. This allows for more mixing of parental genes, potentially exploring the solution space more thoroughly. However, there is an increased complexity and a potential to disrupt important gene combinations (building blocks).

Single-point and multi-point crossover strategies are simple to implement, but they may not effectively explore the solution space.

Uniform Crossover: Each gene is considered independently. For each gene, a coin is flipped to decide which parent it will be inherited from. This provides a more uniform exploration of the solution space and maintains a better balance. It can perform well in exploration but may disrupt advantageous gene combinations found in parents.

Whole Arithmetic Recombination: Used for real-valued chromosomes. Offspring are created as a weighted average of the parent chromosomes. Good for problems where a blend of parent traits is desirable. May converge slowly and is not suitable for problems where specific gene combinations are crucial.

Order-Based Crossover: Designed for problems like the traveling salesman problem, where the solution is an ordered list (like a route or schedule). Preserves relative order of genes, which is essential in order-based problems. More complex and can be less intuitive to implement.

Partially Mapped Crossover (PMX): Specifically designed for permutation encoding. It maintains the absolute position of some elements from one parent while preserving the order of elements from the other parent. Useful in problems where the position of elements is important. Can be computationally more intensive.

Whole Arithmetic and PMX are beneficial for specific problem types but may lack general applicability or be computationally intensive.

Crossover Rate: Not all pairs of chromosomes undergo crossover. This rate determines how frequently crossover occurs within the population.

Crossover in genetic algorithms is a versatile and powerful tool for generating new solutions by combining existing ones. The choice of crossover method can significantly impact the performance of a GA and is highly dependent on the nature of the problem being solved. While some methods are better for preserving gene combinations, others excel in exploring the solution space more thoroughly. The key is to select the crossover method that best aligns with the problem's requirements and the specific characteristics of the solution space.

2.1.2.5. Mutation

This is a random process where some genes in a chromosome are altered. Mutation introduces genetic diversity and helps prevent the algorithm from getting stuck in local optima. Selected solutions undergo mutation is applied to introduce variability.

Types and Methods of Mutation: Mutation in GA is a crucial mechanism that introduces variability into the population, aiding in the exploration of the search space and preventing premature convergence. It simulates random genetic mutations seen in natural evolution. This process involves altering one or more genes in a chromosome (solution) according to a predefined mutation probability. Here, we explore various aspects and methods of mutation in GA. Various techniques exist, such as,like bit-flipping in binary encoding or small changes in real-valued encoding. These techniques are listed below:

- Bit-Flip Mutation: In binary-encoded chromosomes, mutation often involves flipping bits (changing 0 to 1 or vice versa). A random gene (bit) is selected, and its value is inverted. This is akin to a point mutation in biological DNA.
- Real-Valued Mutation: For chromosomes represented by real numbers, mutation involves making small random changes to the numeric value. A gene's value is

perturbed by adding a small, random value, which can be drawn from a distribution like Gaussian or uniform.

- Swapping Mutation: Used mainly in problems like the traveling salesman problem, where the solution is a permutation of cities. Two genes (cities) are selected randomly, and their positions are swapped in the chromosome.
- Scramble Mutation: A subset of genes is chosen and their order is shuffled. Useful in routing or scheduling problems where the sequence of events or locations matters.

Implementation and Control:

- Mutation Rate: One of the most critical parameters in GA. It determines how frequently mutation occurs. This is typically a small probability, ensuring only a few genes are mutated to maintain diversity without disrupting the evolutionary process excessively. Usually set to a low value, as high mutation rates can turn the GA into a random search.
- Adaptive Mutation: In some advanced GA, the mutation rate is adapted based on the performance of the population. If the population is converging too quickly, the mutation rate can be increased to introduce more diversity.

Impact on Chromosome and Population:

- Introduction of New Traits: Mutation can introduce new genetic material into the population, potentially leading to better solutions.
- Diversity Maintenance: By altering chromosomes, mutation helps maintain genetic diversity within the population, which is vital for a healthy evolutionary process.
- Avoidance of Local Optima: Mutation helps the algorithm to escape local optima by providing a mechanism to explore new regions of the search space.

Mutation in Image-Based Problems:

- Image Optimization: In problems where the solutions are images (like evolving art or designs), mutation might alter pixel values, color intensities, or geometric shapes within the image.
- Effect on Output: These changes can lead to significant visual alterations, leading to diverse and often unexpected designs or patterns.

General Characteristics:

- Balance: The key is to balance the rate and extent of mutation. Too much mutation can destroy good solutions, while too little can lead to stagnation.
- Context-Dependent: The best approach to mutation often depends on the specific problem and the representation of solutions.

Mutation is a fundamental aspect of genetic algorithms, serving as a tool for introducing randomness and variability. It counteracts the homogenizing effect of crossover and selection, ensuring that the GA does not get trapped in sub-optimal regions of the search space. To ensure the success of any GA, the mutation method and rate must be chosen carefully. Understanding and effectively implementing mutation can significantly impact the performance and efficiency of genetic algorithms in solving complex optimization problems.

2.1.2.6. New Generation

A new population of solutions is formed, usually by replacing the older generation entirely or by a mixture of old and new solutions.

Replacement Strategies: The new generation of solutions can be formed in various ways, such as completely replacing the old generation or a mix of old and new.

Convergence Check: The algorithm checks for convergence, typically a state where additional iterations do not significantly improve solutions.

2.1.2.7. Termination

This process of selection, crossover, and mutation is repeated over several generations. The algorithm typically terminates when a satisfactory level of fitness is achieved or after a set number of generations.

Termination Criteria: Common criteria include the highest number of generations that can be achieved, a high level of fitness, or a plateau in fitness levels indicating convergence.

Result Selection: The best solution(s) from the final population are chosen as the output of the algorithm.

2.1.3. Applications and Examples

Genetic algorithms have been successfully applied in various domains such as:

2.1.3.1. Optimization Problems

Finding the best solution from a large set of possibilities, like scheduling, route planning, or resource allocation.

2.1.3.2. Machine Learning

Feature selection, neural network training, and other areas where traditional methods are computationally expensive.

2.1.3.3. Evolutionary Design

In engineering, for designing more efficient structures and systems by simulating evolutionary processes.

2.1.4. Variations and Advanced Techniques

2.1.4.1. Hybrid Approaches

GA are often combined with other optimization techniques, like local search algorithms, to refine solutions.

2.1.4.2. Parallel and Distributed GA

Parallel and Distributed GA: These are used to speed up the process and improve solutions by evolving multiple populations simultaneously.

2.1.4.3. Adaptive GA

Here, parameters like mutation and crossover rates are adapted dynamically based on the performance of the current population.

2.2. Convolutional Neural Networks

CNNs are a significant model used in the field of artificial intelligence and deep learning. They are fundamentally designed for processing and analyzing visual data through their multi-layered structure. In these artificial neural networks, each layer undertakes different tasks. The initial layers generally focus on detecting simple features, such as edges or color transitions. In deeper layers, these simple features are combined to define more complex characteristics. A CNN typically consists of a series of layers that transform the input image to produce an output prediction. These layers are: convolution, pooling, fully connected and

normalization layers. A general CNN model used for the image classification problem is shown in Figure 2.2

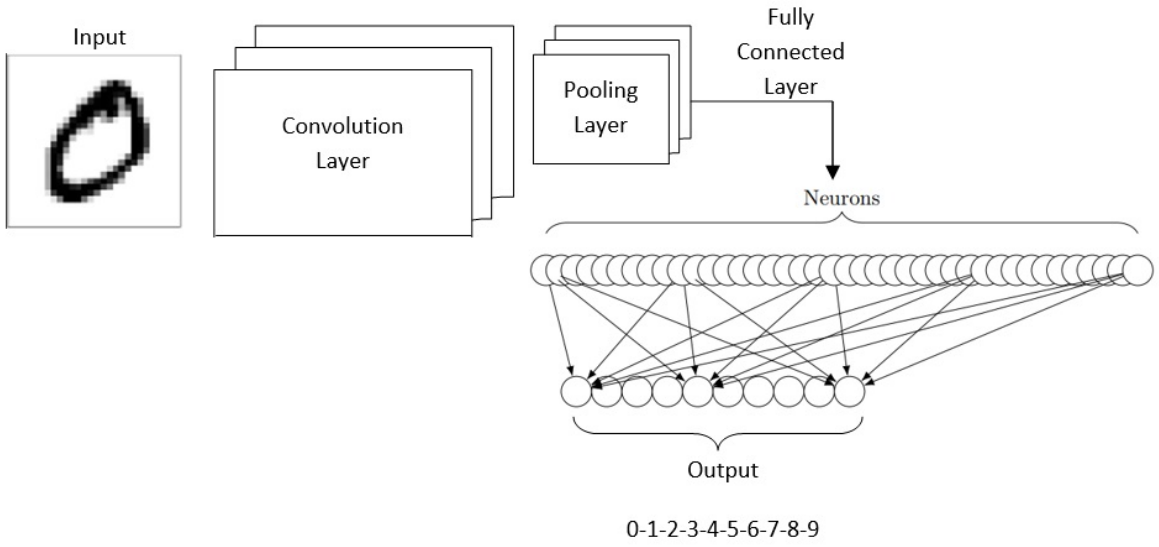

Figure 2.2 An Example CNN for Classifying Images

2.2.1. Convolution Layers

These are the core building blocks of a CNN. They apply a number of filters to the input image to create a feature map. This process captures the local dependencies in the original image. These layers perform convolution operations by sliding a filter over the image. As the filter moves across the image, it multiplies its values with the original pixel values. The results of these multiplications are summed up to produce a single pixel in the output feature map. For example, consider a 3x3 filter moving across a 10x10 image. At each position, the filter overlaps with a 3x3 section of the image. The convolution involves element-wise multiplication between the filter and the image section and sums up these values to form a single pixel in the output feature map. This process is illustrated in Figure 2.3.

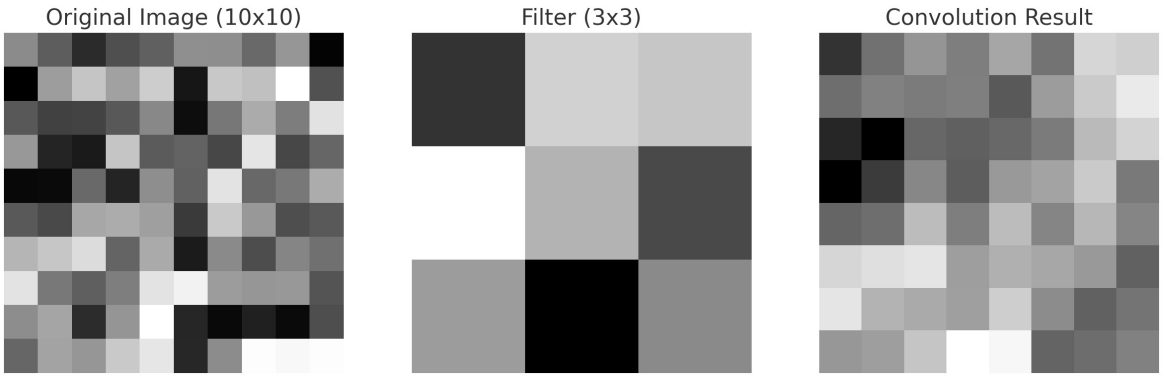

Figure 2.3 Representative Illustration of the Convolution Process

2.2.2. Activation Layers

Activation layers in neural networks, especially in CNNs, disrupt the linearity of the model, allowing it to learn more complex patterns in the data. Their primary roles are their contributions in whether a neuron is active or not. There are several types of activation functions, each with its unique characteristics and applications. These functions are examined below:

2.2.2.1. Rectified Linear Unit (ReLU)

ReLU is defined as:

$$O_{\text{relu}}(z) = \max(0, z) \tag{1}$$

It outputs the input directly if it is positive; otherwise, it outputs zero. ReLU is widely used for its simplicity and efficiency. It helps in alleviating the vanishing gradient problem. This activation is applied pixel-wise to the feature. This process is illustrated in Figure 2.4

Original Feature Map: The first image shows the original feature map with a mix of positive (red) and negative (blue) values. The color bar indicates the range of values.

Negative Values in Feature Map: The middle image highlights the negative values in the feature map. Here, negative values are shown in blue, while positive values are made transparent (NaN) to emphasize areas that will be affected by the ReLU function.

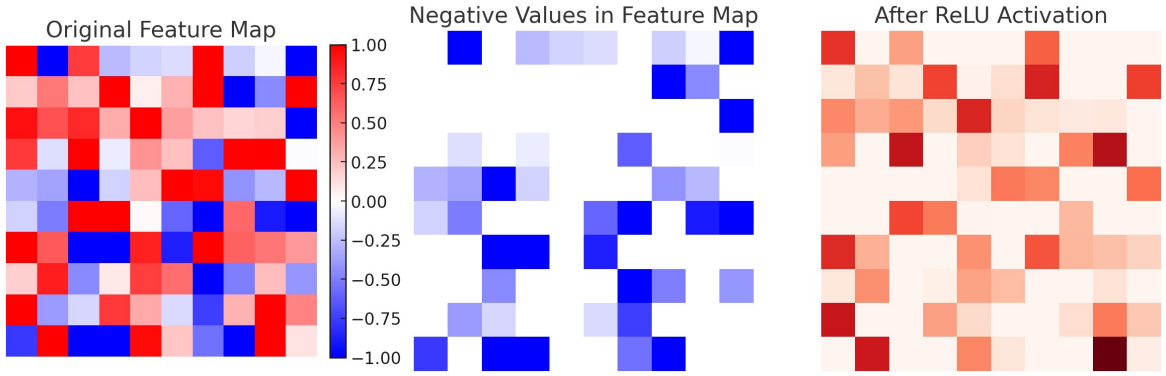

Figure 2.4 Representative Illustration of the ReLU Activation Process

After ReLU Activation: The final image on the right displays the feature map after applying the ReLU activation. In this map, all negative values (previously shown in blue) have been set to zero, as indicated by the dark areas, while the positive values remain unchanged. This set of images illustrates how ReLU effectively zeroes out negative values in the feature map, a key aspect of its functionality in neural networks.

2.2.2.2. Leaky ReLU

A variant of ReLU, it is defined as:

$$P_{\text{leaky}}(y) = \begin{cases} y, & \text{if } y > 0 \\ \beta y, & \text{otherwise} \end{cases} \tag{2}$$

where β is a small constant. Leaky ReLU addresses the issue of dying neurons in ReLU (where some neurons only output zero) by allowing a small gradient when the unit is not active.

2.2.2.3. Parametric ReLU (PReLU)

PReLU and Leaky ReLU are almost the same, but the coefficient α is learned during training rather than being predefined. PReLU is defined as:

$$P_{\text{PReLU}}(y) = \begin{cases} y, & \text{if } y > 0 \\ \beta y, & \text{otherwise} \end{cases} \tag{3}$$

where β is a learnable parameter during training rather than being a predefined constant.

PReLU adapts its behavior to the specific task, potentially leading to better performance in certain scenarios. This learnable parameter α allows the function to be more flexible, as it can adjust during the training process, potentially improving the learning and performance of the neural network.

2.2.2.4. Sigmoid Function

The sigmoid function is also called the logistic function and is defined as:

$$S_{\text{sigmoid}}(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

It outputs a value between zero (0) and one (1), making it useful for models where we need probabilities. Commonly used in the output layer of binary classification problems, though less common in hidden layers due to the vanishing gradient problem.

2.2.2.5. Hyperbolic Tangent

The Hyperbolic Tangent (tanh) function and sigmoid function are almost the same, but outputs values between -1 and 1. The tanh function is defined as:

$$T_{\text{tanh}}(w) = \tanh(w) = \frac{2}{1 + e^{-2w}} - 1 \tag{5}$$

It is often used in hidden layers of a neural network as it centers the data, making learning for the next layer easier.

2.2.2.6. Softmax

The Softmax function compresses the outputs of each unit between 0 and 1. In this respect, it is a generalization of the sigmoid function, and also divides by their sum, effectively giving the probability distribution over a set of outputs. Primarily used in the output layer for multi-class classification problems. It is defined as:

$$S_{\text{softmax}}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{6}$$

for $i = 1, \ldots, n$, where $\mathbf{z} = [z_1, z_2, \ldots, z_n]$ is the input vector, and n is the number of classes.

It converts the logits (i.e., raw prediction scores) into probabilities by dividing the exponential of a given score by the sum of exponentials of all scores in the vector. This ensures that the output probabilities sum to 1, making it a valid probability distribution. It is particularly useful for cases where classes are mutually exclusive.

2.2.2.7. Swish

Swish is defined as:

$$S_{\text{swish}}(y) = y \cdot \text{sigmoid}(\gamma y) \tag{7}$$

where γ is a learnable parameter or a fixed value. Swish tends to work better than ReLU on deeper models across a number of datasets and models.

2.2.2.8. Exponential Linear Unit

Exponential Linear Unit (ELU) is defined as:

$$E_{\text{elu}}(z) = \begin{cases} z, & \text{if } z > 0 \\ \delta(e^z - 1), & \text{otherwise} \end{cases} \tag{8}$$

where δ is a scaling parameter. ELU aims to combine the benefits of ReLU and its variants (efficient computation, non-saturating form) and the sigmoid functions (smooth output for negative inputs).

2.2.3. Pooling Layers

Pooling layers come after convolution layers. By reducing dimensionality, it reduces the number of parameters in the network. Thus, the computational burden is reduced. This also helps in making the detection of features invariant to scale and orientation changes. Pooling can be done in different types. Most commonly used average and maximum pooling methods. For example, in max pooling, a filter moves across the feature map and takes the maximum value from the section of the map it covers. If a 2x2 filter is applied to a 4x4 feature map, it will produce a 2x2 output where each pixel represents the maximum value of a 2x2 area in the input feature map. This process is illustrated in Figure 2.5

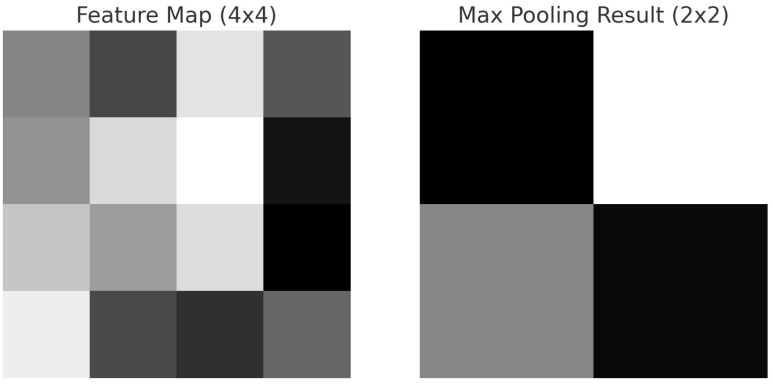

Figure 2.5 Representative Illustration of the Pooling Process

This shows the result of applying a 2x2 max pooling filter to the 4x4 feature map. In max pooling, the filter moves across the feature map and selects the maximum value from each 2x2 area it covers. The output is a 2x2 matrix where each pixel represents the maximum value of a 2x2 area in the input feature map.

2.2.3.1. Max Pooling

In max pooling, a filter (usually 2x2 or 3x3 in size) moves across the input feature map. At each position, the maximum value within the window of the filter is selected and output to the new, reduced-size feature map.

2.2.3.2. Average Pooling

Similar to max pooling, but instead of taking the maximum value, it computes the average of the values in the filter window. This results in a pooled feature map where each element is the average of the corresponding window elements in the input map.

2.2.3.3. Global Pooling

In global pooling, instead of a small window, the entire feature map is considered. The most common types include global mean and global maximum pooling. They take the maximum or average of the entire feature map, respectively, reducing each map to a single value. This is particularly useful for reducing the dimensions of the feature maps before a fully connected layer.

The primary task of each type of pooling is to reduce the size of feature maps. In this way, the number of parameters of the network is reduced. Additionally, the computational burden is reduced. This not only helps in reducing overfitting but also ensures that the network becomes somewhat invariant to small translations in the input image.

2.2.4. Fully Connected Layers

After the convolution and pooling layer, there are fully connected layers. The most important parameters for the decision mechanism are extracted through these layers. There are connections between all layers. For example, if the previous layer outputs a 5x5x10 feature map (10 different 5x5 feature maps), this might be flattened into a 250-element vector (5x5x10) and connected to a fully connected layer with, for instance, 100 neurons. Each of these neurons will be connected to all 250 elements. This is illustrated in Figure 2.6

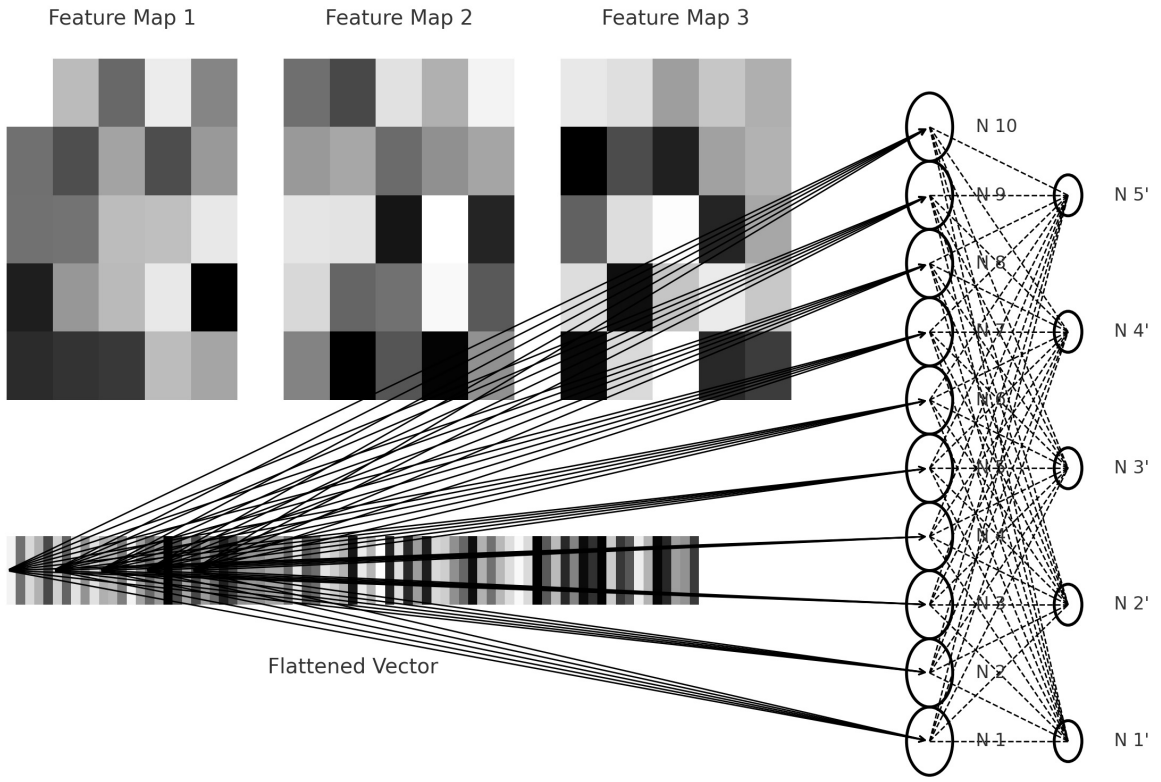

Figure 2.6 Representative Illustration of the Fully Connected Process

2.2.5. Normalization Layers

Normalization layers in CNNs play a crucial role in stabilizing and speeding up training. There are several types of normalization methods, each with its own mechanism and purpose.

These methods normalize the input layer by adjusting and scaling activations.They improve the stability and performance of the neural network. Normalization methods are examined in detail below:

2.2.5.1. Batch Normalization

This method normalizes the inputs of a layer across the batch dimension. It adjusts the activations of a previous layer by scaling them to have a mean of zero (0) and a standard deviation of one (1), based on the statistics of the current batch of data. The distribution of network activations may change during training. Batch normalization helps reduce this variation. This leads to faster training and reduces the sensitivity to network initialization. In batch normalization, the activations of a previous layer are normalized so that they have a mean of zero (0) and a variance of one (1), as shown in Figure 2.7. This is done for each batch of training data, hence the name 'batch normalization'.

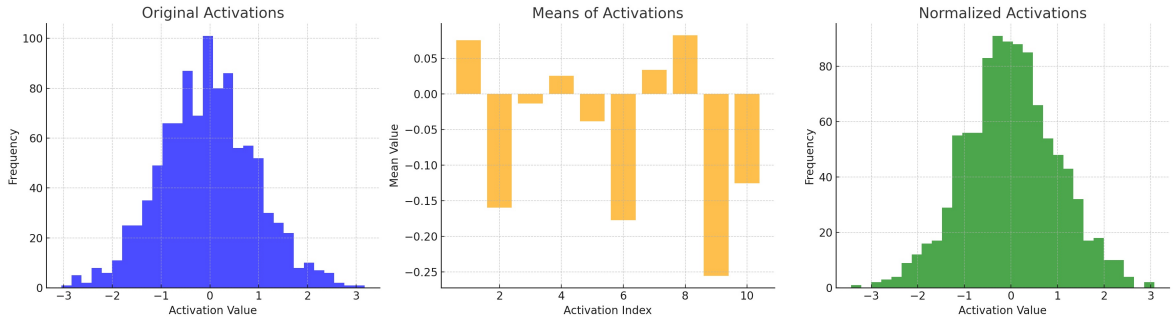

Figure 2.7 Representative Illustration of the Batch Normalization Process

- Original Activations: The histogram on the left shows the distribution of the original activation values from a batch. These values are randomly generated and resemble a normal distribution.
- Means of Activations: The bar plot in the middle displays the mean values of the activations before normalization. Each bar corresponds to one of the 10 features/activations, and you can see that the means are not centered at zero.

- Normalized Activations: The histogram on the right shows the distribution of activation values after batch normalization. The normalization process has adjusted the activations so that they now have a zero (0) and a variance of one (1), which is the goal of batch normalization.

This process helps the network to be trained in a faster and more stable manner. Specifically, batch normalization:

- Speeds Up the Learning Process: Allows the network to be trained faster using higher learning rates.
- Reduces Dependency on Weight Initialization: Decreases the need for selecting a good initial weight value.
- Creates a Regularization Effect: The normalization over mini-batches creates a regulatory effect and can help reduce overfitting.

2.2.5.2. Layer Normalization

Unlike batch normalization, layer normalization performs normalization across the features instead of the batch dimension. Here, the normalization is applied for each data point separately, and all the features are normalized together. This type of normalization is particularly effective in recurrent neural networks (RNNs) and is less dependent on batch size, making it suitable for tasks with a smaller batch size or where batch size can vary.

2.2.5.3. Instance Normalization

Instance normalization normalizes each individual data instance in each channel separately. It's similar to layer normalization but normalizes across each channel in each training example. It has been found particularly useful in style transfer applications in neural networks.

2.2.5.4. Group Normalization

This method divides the channels into groups and normalizes the features within each group. It's a middle ground between layer normalization and instance normalization. Group normalization is effective in cases where the batch size is small, making batch normalization ineffective.

2.2.5.5. Weight Normalization

This involves normalizing the weights of each neuron in a layer, which is different from normalizing the layer's activations. It decomposes the weight vector into a parameter representing its length and a parameter representing its direction. Weight normalization accelerates convergence of stochastic gradient descent by decoupling the length of the weight vectors from their direction.

Normalization layers, in general, help in smoothing the optimization landscape and enable the use of higher learning rates, which can lead to faster convergence of the network. They also play a role in regularizing the model, which can help reduce overfitting. Each normalization method has its own advantages and is chosen based on the specific requirements of the neural network architecture and the nature of the task at hand.

2.2.6. The Image Processing Process in a CNNs

Filters, also known as kernels, are small matrices used to transform an image through a convolution operation. During convolution, the filter slides over the input image. At each position, the filter is applied to the corresponding part of the image. This involves element-wise multiplication between the values in the filter and the pixel values of the image. These products are then summed up to get a single value, which forms a pixel in the output feature map. These filters are designed to capture specific types of features from the image, like edges, textures, or patterns. For instance, certain filters might detect vertical edges, while

others might be sensitive to horizontal or diagonal edges. The representative illustration can be found in Figure 2.8

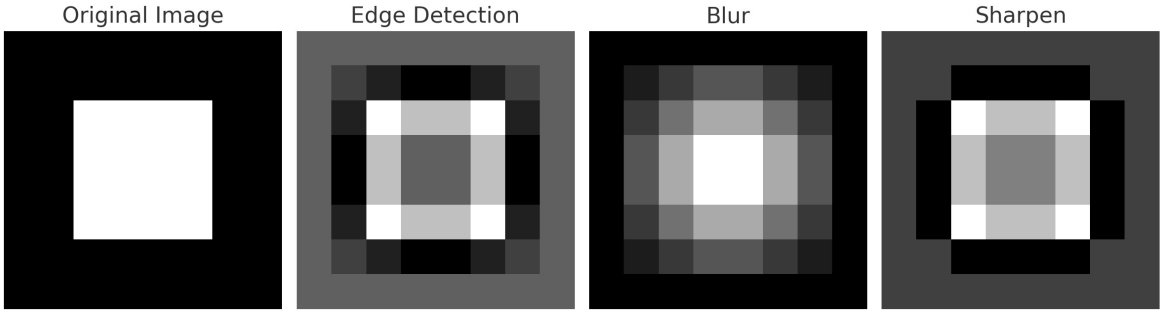

Figure 2.8 Process of Filtering in Convolution Layers

In the image, the original black and white image and the results of three different filters applied to this image can be seen: edge detection, blurring, and sharpening. Each filter creates a different effect on the image.

In a trained CNN, the values that the filters can take are learned during the training process. The network adjusts these values to minimize the difference between the its actual and predictions data.

In the next step, an activation function (ReLU) and pooling operation are applied to these convolved images. ReLU transforms negative values to zero while leaving positive values unchanged. The pooling operation, on the other hand, has reduced the size of the image and preserved important features. Figure 2.9 shows images where the ReLU activation function has been applied.

Finally, the pooling operation has been applied. Pooling is used to condense the information in the image and reduce its size. Here, the most common method, max pooling, has been used. This step reduces the size of the image by taking the largest value in each small window, preserving the most important features. Figure 2.10 shows the results of the max pooling operation.

These figures demonstrate a typical image processing sequence within a CNNs:

- Original Image: A simple black and white image used to start the processing.

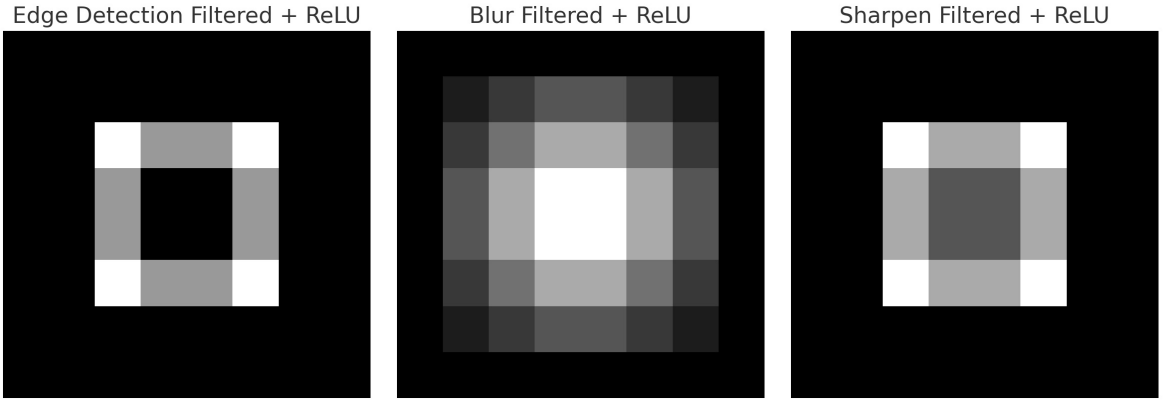

Figure 2.9 Activation Process in Convolved Images

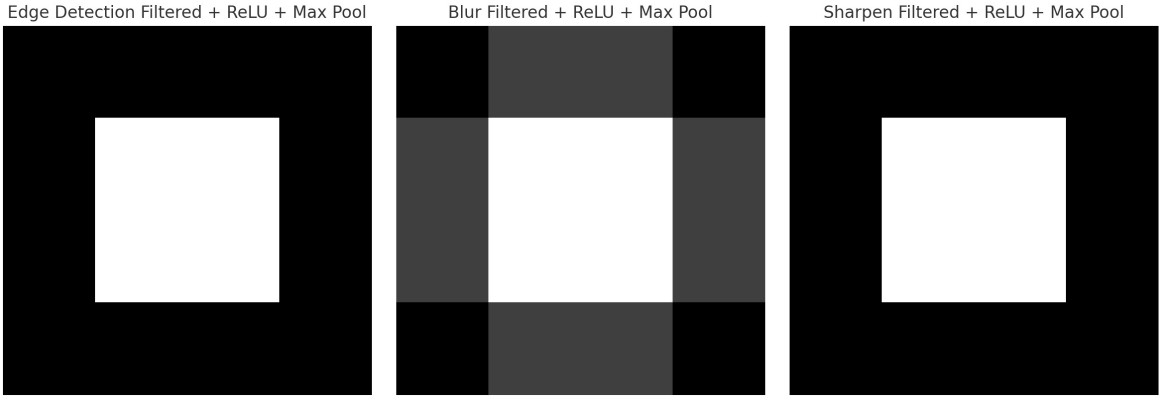

Figure 2.10 Pooling Process in Convolved Images

- Convolution Layers: Various filters are applied to the image to extract different features.
- ReLU Activation Function: Transforms negative values to zero while leaving positive values unchanged.
- Max Pooling: Takes the largest value in each window to condense information in the image and reduce its size.

These steps form the basic building blocks of CNNs and are commonly used in more complex visual recognition tasks.

The image processing process in CNNs is detailed in Figure 2.11, which includes numerical values in the images.

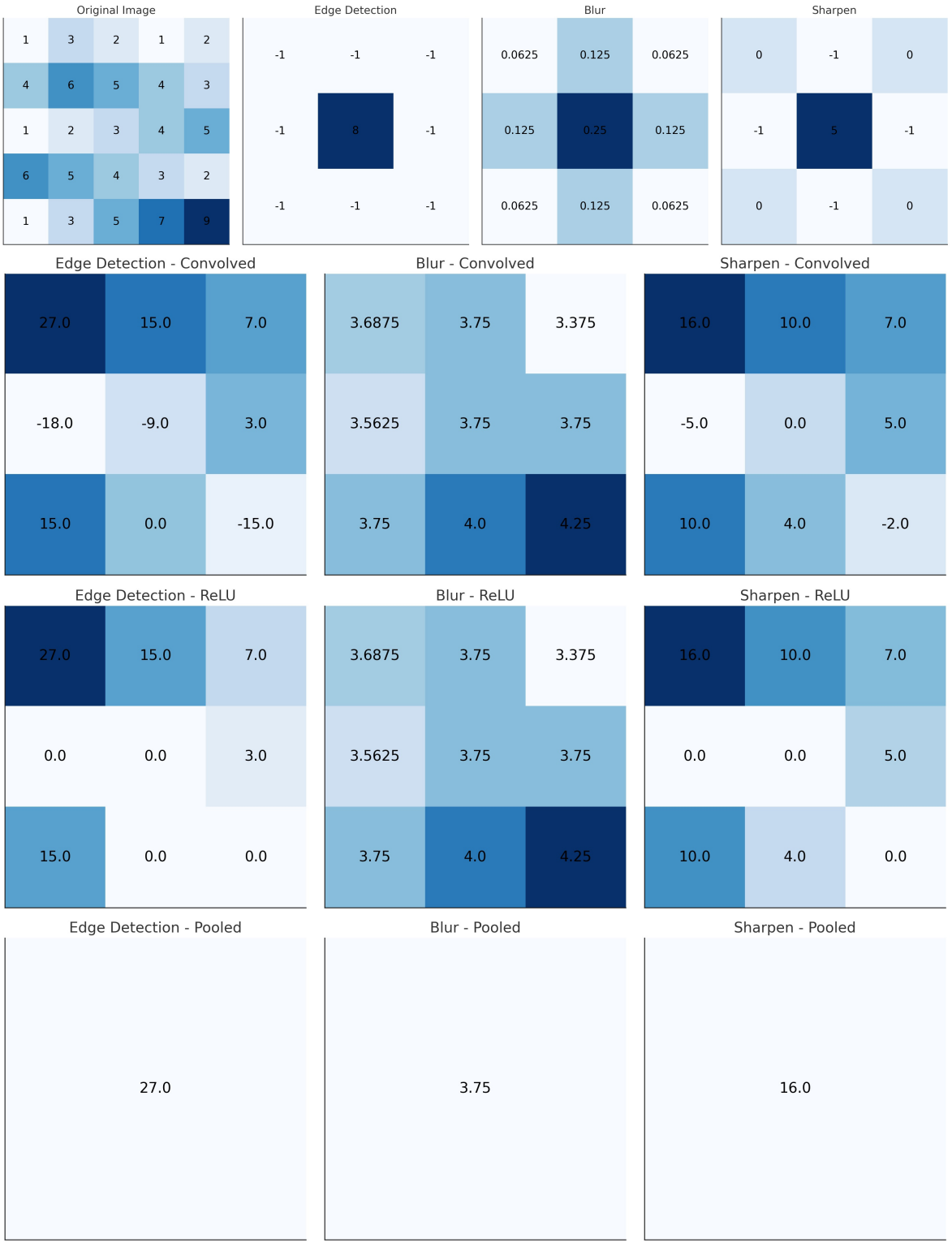

Figure 2.11 The Image Processing Process in a CNNs

- Original Image and Filters: The first row shows a more complex original image and three different types of filters (Edge Detection, Blurring, Sharpening). Each filter is designed to create a different effect on the image.
- Convolution Results: The second row shows the results of applying each filter to the original image. This reflects the effect of the filter on the image.
- ReLU Activation Results: The third row demonstrates the application of the ReLU activation function to each matrix obtained after convolution. In this step, negative values are zeroed out while positive values remain unchanged.
- Pooling Results: The fourth row displays the application of max pooling to each matrix obtained after the ReLU step. In this step, the size of the image is reduced while important features are preserved.

These figures present a more detailed view of the numerical effects and outcomes of convolution, activation, and pooling processes in CNNs. Each process is designed to extract different types of information from the image and provide necessary features for subsequent layers.

Each of these layers plays a crucial role in the CNN's ability to learn from visual data. By stacking these layers, CNNs can effectively learn hierarchical representations of the input data, which is essential for complex tasks like image recognition.

CNNs are used effectively in a wide variety of tasks, including image recognition (including video), image analysis and classification, and even playing board and video games. Their ability to learn feature representations directly from data, eliminating the need for manual feature extraction, is one of their most significant advantages.

The advancements in CNNs have largely been driven by increasing computational power, availability of large-scale image datasets like ImageNet, and improvements in training algorithms. As research continues, CNNs are becoming more efficient, accurate, and capable of handling more complex tasks in various domains.

2.3. Masking Techniques

Masking is a technique used to select and process specific parts of an image (generally used in image processing and computer vision). This process is typically done using a mask, which determines the areas of the image to be processed.

The most commonly used type of mask is binary masks. These contain only two values: usually '1' representing the areas to be processed, and '0' representing the areas not to be processed. However, there are also more complex masks like grayscale masks and colored masks.

Masks are usually created according to a specific criterion or condition. For example, by setting a threshold value, all pixels above this value can be marked as '1', and those below as '0'. In artificial intelligence and machine learning applications, masks are often automatically generated, especially using deep learning models. Once a mask is created, it is applied to the relevant image. This is usually done through a multiplication operation: each pixel of the mask and the image is multiplied, so pixels with a '0' value in the mask are ignored in the image, while those with a '1' value are preserved.

Masking is used in a variety of image processing tasks such as object detection, segmentation, image editing, and filtering. For example, in medical imaging, only the area within the mask may be processed for a specific tissue or abnormality, while the rest is disregarded.

Masking and Data Augmentation: Masking can also be used in data augmentation processes. For instance, during the training of models, masks can be used to hide or emphasize certain parts of images to improve the generalization ability of the model.

Masking is a versatile and powerful tool in the fields of computer vision and image processing. Understanding its fundamental principles and applying them correctly enhances the effectiveness and efficiency of these techniques. Particularly in AI and machine learning applications, the proper use of masking is critical for more accurate results and effective analyses. Various masking techniques are as follows:

2.3.1. Binary Masks:

The simplest form of masking is binary masks. These masks are used to create sharp distinctions in images and are commonly used in image segmentation, object detection, and some types of image filtering.

2.3.2. Grayscale Masks:

A more complex masking technique involves grayscale masks. These masks provide more detail and intensity level information compared to binary masks, enabling smoother transitions and more detailed image processing.

2.3.3. Color Masks and Alpha Masks:

Color masks target specific color ranges or color features. Alpha masks contain transparency information and are often used in visual effects. These masks are used to differentiate specific color tones or levels of transparency.

2.3.4. Edge Masks:

Edge masks are used to emphasize edges in images. These masks are typically created using edge detection algorithms and are used to more clearly define the boundaries of objects.

2.3.5. Soft Masks:

Soft masks allow for gradual transitions and provide more natural-looking blends. These masks are particularly important in image merging and composite creation processes.

2.3.6. Special Shape and Channel-Based Masks:

Special shape masks target a specific geometric shape or pattern. Channel-based masks focus on specific color channels of an image and are used for color-based processing.

2.3.7. Adaptive Masks:

In dynamic environments or under changing conditions, adaptive masks are used. These masks automatically adjust themselves based on the content and conditions of the image.

2.3.8. Artificial Intelligence and Deep Learning Approaches:

With the advancement of AI and deep learning technologies, masking techniques have also significantly evolved. Particularly, deep learning-based segmentation models can create much more accurate and detailed masks in complex images.

2.4. Binary Masks

Binary masks, fundamental tools in deep learning, particularly in computer vision and image processing applications, serve to isolate specific parts of an image. In essence, a binary mask is a two-dimensional array where each pixel's value is either 0 or 1. The '1' values indicate the regions of interest, while '0' values represent the background or irrelevant parts.

Binary masks are often created using thresholding techniques, where pixel values in an image that meet certain criteria are marked as 1 (or 'true'), and others as 0 (or 'false'). These criteria can be based on intensity, color, texture, or other image properties. Once created, these masks are applied to images through operations like bitwise operations (AND, OR), which combine the mask and the image to extract or suppress specific regions.

Binary masks are used in various fields within artificial intelligence (AI) as well as in deep learning. They offer versatile applications due to their ability to segment and highlight specific regions in data.

Despite their utility, binary masks have limitations. They are inherently binary, which can lead to a loss of information, especially in regions with gradual transitions. Also, creating accurate masks manually or with simple algorithms can be challenging for intricate or noisy images.

Binary masks are a simple yet powerful tool in image processing, offering a straightforward approach to isolating and manipulating specific regions within images. Their evolution, closely tied to advancements in computational techniques, continues to play a critical role in various applications, from basic image editing to complex tasks in medical imaging and autonomous vehicle navigation. Some of these applications are examined below.

2.4.1. Usage in Deep Learning

2.4.1.1. Image Segmentation

Perhaps the most common use of binary masks is in image segmentation tasks. In segmentation, the goal is to classify each pixel of an image into different categories. Binary masks make this task manageable by allowing algorithms to focus only on relevant parts of the image. For instance, in medical imaging, binary masks help in isolating tumors from MRIs or CT scans.

2.4.1.2. Object Detection and Localization

In object detection, binary masks assist in pinpointing the exact location of objects within an image. This is crucial in applications like autonomous vehicles, where accurately detecting and localizing obstacles (like pedestrians or other vehicles) is essential for safety.

2.4.1.3. Data Augmentation

In deep learning, data augmentation is a technique used to increase the diversity of a dataset without actually collecting new data. Binary masks can be used to blend or superimpose images, creating new training samples from existing ones.

2.4.1.4. Background Removal

Binary masks are instrumental in applications that require the extraction of foreground elements from the background. This technique is widely used in video conferencing tools, where the user's background is often blurred or replaced.

2.4.1.5. Implementation Techniques

Convolutional Neural Networks: CNNs, especially those designed for segmentation like U-Net, often output binary masks as part of their architecture. These networks are trained to understand the spatial hierarchies in images, enabling them to distinguish between relevant and irrelevant pixels.

Thresholding Techniques: In simpler applications, thresholding can be used to create binary masks. Here, pixel values above a certain threshold are set to '1', and others to '0'. This method is common in edge detection and basic object recognition tasks.

Region-Based Methods: Techniques like Region-based CNN and its variants use binary masks for more precise object detection and segmentation.

By isolating the regions of interest, binary masks reduce computational complexity, allowing deep learning models to focus only on important parts of the image. In tasks like medical image analysis, the precise delineation of areas (like tumors) can significantly impact diagnosis and treatment plans, making accuracy crucial. Binary masks are adaptable

to various applications, from simple background removal to complex medical image segmentation, demonstrating their versatility.

In conclusion, binary masks are vital in the realm of deep learning for image processing. Their ability to isolate and highlight specific regions of an image makes them indispensable tools in enhancing the accuracy and efficiency of various applications, ranging from autonomous vehicles to medical diagnostics.

2.4.2. Usage in Traditional Machine Learning

2.4.2.1. Feature Engineering

In traditional machine learning, binary masks can be used for feature selection or extraction. By masking certain features in a dataset, algorithms can focus on the most relevant attributes, improving model performance and reducing overfitting.

2.4.2.2. Preprocessing in Image Analysis

Prior to the advent of deep learning, binary masks were used in image analysis tasks like edge detection, object recognition, and morphological operations in computer vision, using techniques like thresholding and contour detection.

2.4.3. Usage in Natural Language Processing

2.4.3.1. Text Segmentation

Similar to image segmentation, binary masks in Natural Language Processing (NLP) can be used for tasks like sentence splitting or identifying specific parts of texts (like keywords or phrases).

2.4.3.2. Attention Mechanisms

In more advanced NLP systems, binary masks can facilitate attention mechanisms by masking out irrelevant parts of the input data, thereby allowing models to focus on the important segments of text for tasks like machine translation or summarization.

2.4.4. Usage in Robotics

2.4.4.1. Environmental Interaction

In robotics, binary masks can assist in environmental understanding, like distinguishing between navigable areas and obstacles in a robot's path planning algorithms.

2.4.4.2. Object Manipulation

Binary masks help in object recognition and manipulation tasks, enabling robots to identify and interact with specific objects in their environment.

2.4.5. Usage in Augmented and Virtual Reality

2.4.5.1. Image Composition

In Augmented and Virtual Reality (AR/VR), binary masks are used for blending real and virtual elements seamlessly. They help in isolating objects from the background, which can then be superimposed onto virtual environments.

2.4.5.2. Interaction Mechanics

Binary masks aid in detecting user interaction areas, such as hand gestures or eye movements, to enhance the interaction experience with virtual elements.

2.4.6. Usage in Data Visualization and Analysis

2.4.6.1. Masking Sensitive Information

In data analysis, binary masks can be employed to anonymize or hide sensitive information in datasets, ensuring privacy and compliance with data protection regulations.

2.4.6.2. Highlighting Key Data Points

They are also useful in visual analytics to highlight or filter specific data points or trends within large datasets, enhancing the interpretability of the visualized information.

Across these diverse AI applications, the primary purpose of binary masks remains consistent: to isolate and emphasize specific regions or features within a dataset, whether it's an image, text, or any other form of data. This isolation aids in reducing noise and focusing the AI system's processing on the most pertinent information, thereby improving efficiency, accuracy, and performance of the system. In tasks like robotics and AR/VR, binary masks play a crucial role in ensuring smooth interaction between the AI system and its environment or users, demonstrating their wide-ranging applicability and importance in the field of AI.

3. RELATED WORK

The deep learning has achieved extraordinary successes in a wide range of fields [1]. Moreover, in the area of image classification, deep learning models like CNNs have been groundbreaking, outperforming all other methods [2]. However, along with the power of these models come some challenges. One of the most important issues with deep learning models is the ”catastrophic forgetting problem” [3].

The catastrophic forgetting problem refers to the situation where a model forgets what it has learned in previous tasks as it learns a next task. This is particularly problematic in the context of multi-task learning. Multi-task learning contains training a single model to learn multiple tasks simultaneously and holds great potential for improving a model's generalization ability [4]. However, the forgetting problem can make it challenging to fully realize this potential.

In the literature, there are three main approaches to tackling catastrophic forgetting: regularization-based approaches, replay approaches using memory, and architectural approaches [5–7], [8–12], [13–27]. Each has its own advantages and disadvantages, but in many cases, a combination of methods is used to produce more effective solutions.

Regularization-based approaches generally aim to preserve the impact of old data and constrain parameter changes based on this data [5-7]. For example, [5] penalize the alteration of important parameters, while [7] preserve the activations of the network trained for the old task during training with new data.

Replay approaches facilitate learning by using a replay memory [8]. In addition to this, Maximally Interfering Retrieval (MIR) selects examples with high forgetfulness, while Gradient-based Memory Editing (GMED) makes them more forgettable [9, 10]. Hindsight Anchor Learning (HAL) gives higher priority to forgettable examples [11].

Architectural approaches attempt to solve the problem by altering or customizing the structural features of the network [13-27]. For example, PackNet uses weight-based pruning techniques [19], while [27] use genetic algorithms to find the task-specific optimal path.

In recent years, metaheuristic algorithms, especially genetic algorithms, have been widely used in artificial neural network optimization [28–39]. [29] have stated that optimization results obtained with genetic algorithms are better than those obtained with gradient descent algorithms. [32] used genetic algorithms in conjunction with backpropagation algorithms to improve the architecture and weights of a deep neural network. [36] use both neuroevolution and backpropagation together, while [35] have employed evolutionary algorithms to reduce computational cost. [37] used genetic algorithms in difficult reinforcement learning problems, and [38] used them to explore convolutional network architectures for the MNIST dataset.

Overall, many approaches have been suggested to solve continuous learning and catastrophic forgetting problems, and the integration of genetic and evolutionary algorithms with neural networks and deep learning is considered one of the promising methods in this field. The key emphasis is on the potential of these algorithms to overcome limitations encountered by traditional methods (e.g., backpropagation) particularly in areas like network architecture selection, hyperparameter optimization, and difficult optimization problems.

In this study, we propose to mitigate the catastrophic forgetting problem and enhance the multi-task learning capability of deep learning models by using genetic algorithms and binary masks. Binary masks allow us to selectively regulate the model parameters, thus enabling us to manage the information transfer between different tasks in the model. Various studies also exist where genetic algorithms have been used for the optimization of binary masks [40, 41].

4. PROPOSED METHOD

4.1. Model Initialization and Binary Mask Generation

4.1.1. Model Initialization

CNN model is initialized with a predefined architecture tailored for image classification tasks. The CNN model used in this study consists of 4 convolution layers and 2 fully connected layers. After each convolution layer, there is a ReLU activation function, a max pooling layer, and a batch normalization layer. The convolution layer is represented by equation (9).

$$O_{\text{conv}}(i,j) = \sum_{m=-\frac{k}{2}}^{\frac{k}{2}} \sum_{n=-\frac{k}{2}}^{\frac{k}{2}} \left[W(m,n) \times X(i-m, j-n) + b \right] \tag{9}$$

In this formula, O_{conv} represents the output of the convolutional layer, X represents the input image, W represents the weights of the convolution kernel, b is the bias term, and k represents the size of the kernel. For each position (i, j) on the input image, a multiplication operation is performed between the kernel weights and the relevant region of the image and the output value is obtained through a summation operation.

The ReLU activation function is a function that increases the learning capacity of the model by turning negative values present in the output of the convolution layer to zero. The ReLU is defined by equation (10).

$$O_{\text{relu}}(x) = \max(0, x) \tag{10}$$

This function involves comparing the input value x with 0 and selecting the larger value.

The max pooling layer is a layer used to reduce the size of the feature map obtained from the output of the convolution layer. This process is illustrated with the following by equation (11).

$$O_{\text{pool}}(i,j) = \max\left(O_{\text{relu}}(is : is+k, js : js+k)\right) \tag{11}$$

In this formula, O_{pool} designates the output of the max pooling layer, O_{relu} stands for the input image, s defines the stride size, and k delineates the size of the pooling kernel. It generates the output map by taking the maximum value within a specific window in the input image.

The batch normalization layer is a technique that accelerates the training process of the model and facilitates a more stable learning dynamic. This layer carries out normalization operations at the mini-batch level and is formalized by equation (12).

$$O_{\text{bn}}(x) = \gamma\left(\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}\right) + \beta \tag{12}$$

Here, O_{bn} represents the output of the batch normalization layer, x denotes the input, μ and $\sqrt{\sigma^2}$ represent the mean and variance of the mini-batch respectively, γ and β are tunable parameters, and ϵ is the stabilizing term.

Finally, the fully connected layer is the layer that performs a task such as classification or regression using the local features learned earlier. The functioning of this layer is shown by the following by equation (13).

$$O_{\text{fc}} = O_{\text{bn}}W + b \tag{13}$$

In this formula, O_{fc} specifies the output of the fully connected layer, O_{bn} represents the input, W is the weight matrix, and b designates the bias vector.

4.1.2. Binary Mask Generation

Binary masking in neural networks is a technique used for feature selection, regularization, and adaptability [22, 26, 42, 43]. Binary masking is the process of selectively ”disabling” specific nodes. This is achieved by performing an element-wise multiplication between the input data and the mask. The mask is a matrix consisting of zeros and ones. When an input feature is multiplied by ”1,” that feature becomes ”active” and is considered by the model. When a feature is multiplied by ”0,” it becomes ”inactive” and is ignored by the model [44].

Binary masks can be learned in various ways. In this study, the mask is learned during model training using GA. The mask is a part of the model and is trained along with the other parts of the model. Such masks are commonly referred to as ”learned masks.” Learned masks [45–47] allow the deep learning model to assign different weights to different parts of the model, thereby potentially improving the model's performance and generalization ability.

In our study, the binary mask is used to enable the model to learn from different datasets. A mask optimized for a specific dataset allow us to use different weights in the model

4.1.3. GA For Model And Binary Mask Optimization

In this study, GA is used to optimize the parameters and binary mask of our deep learning model. The mask regulates the transfer of information between different tasks. GA ensures that the mask is adjusted appropriately, allowing the model to balance information transfer across various tasks and enabling it to train on multiple tasks without encountering the catastrophic forgetting problem.

GA operates on a population of model-mask pairs, providing a potential solution for our optimization problems. The mask is optimized separately for each dataset and is applied during the training of the model. The model is trained on each dataset, and the fitness of the masks is evaluated based on the performance of the model on each dataset. The algorithm

continues until a specific stopping criterion is satisfied. The ability of masks to turn the parameters of the network on and off according to the datasets is shown in the Figure 4.1.

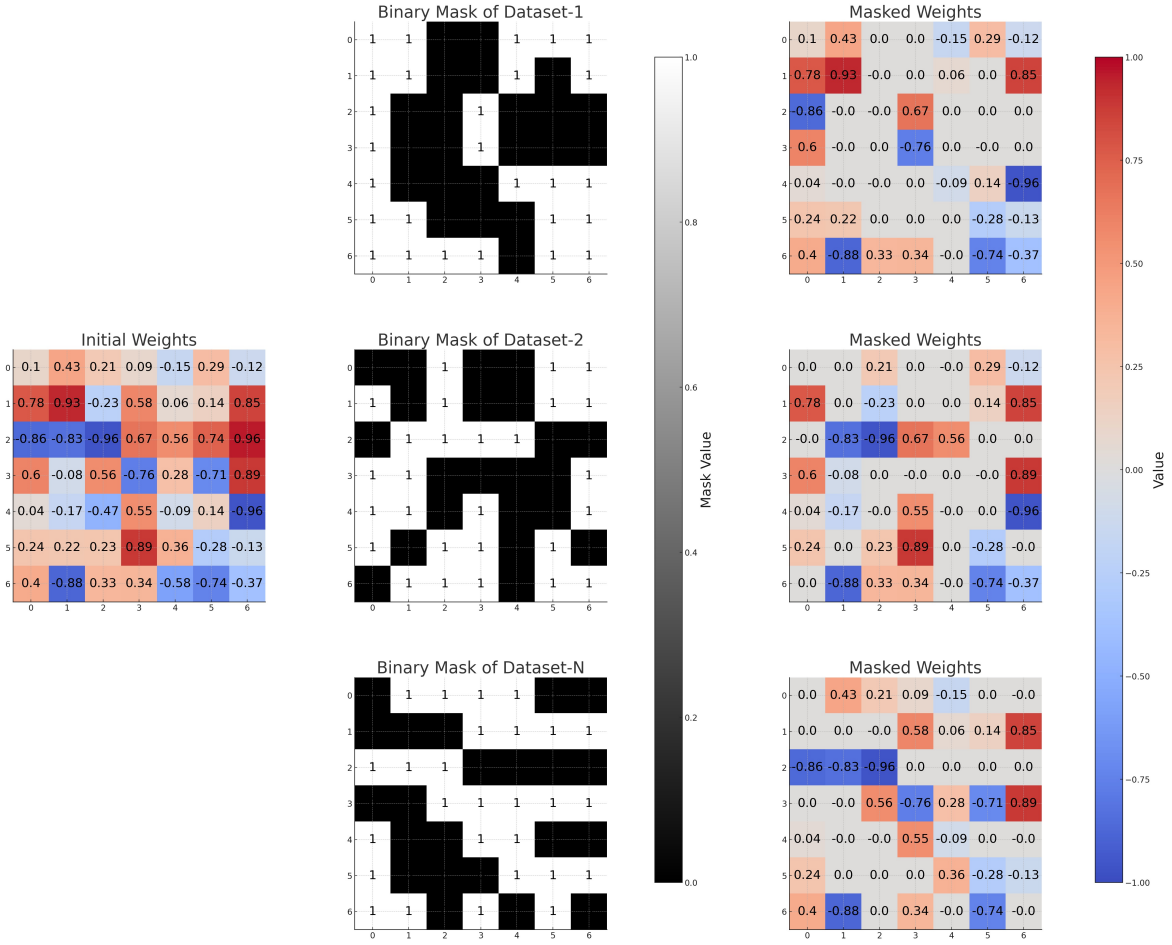

Figure 4.1 Using the Model by Applying Binary Masks to Different Datasets

We create binary masks of the same dimensions as well as the parameters of the model. Masks were initially created randomly for each dataset. To distinguish which mask belongs to which dataset, each mask is assigned a unique index. This index serves as an identifier linking each mask to its corresponding dataset.

4.2. Individual And Population Structure

Each individual in the population consists of a single CNN model and binary masks separately created for each dataset. Only weight values of the model have been masked.

The optimization process has been performed only on the weights of the convolution and dense layers. Figure 4.2 shows the population and individual structure.

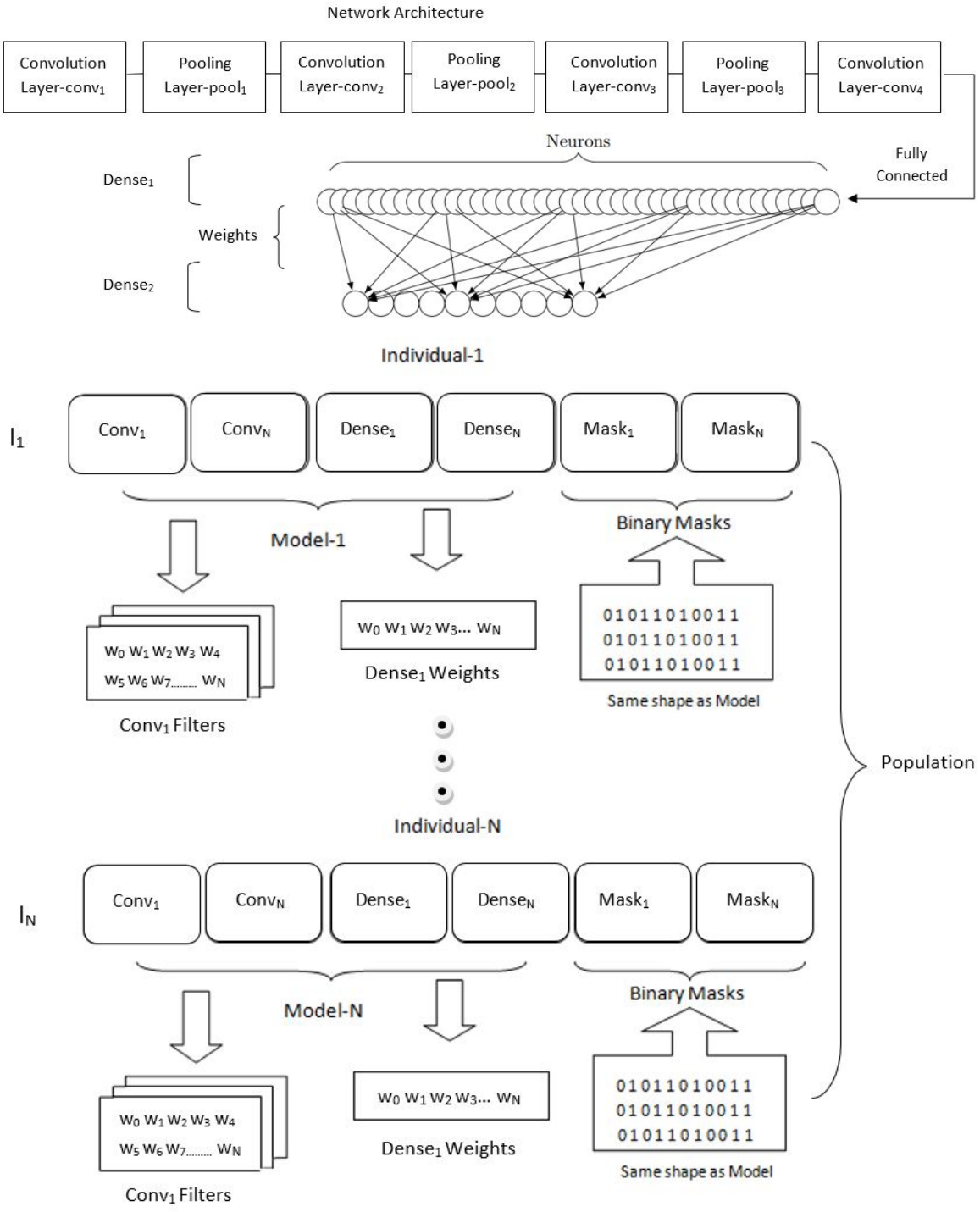

Figure 4.2 The Population and Individual Structure

4.3. Genetic Optimization Process

The best individuals in the population are selected from those with the highest accuracy values. The highest accuracy values are calculated by evaluating the model with the mask corresponding to each dataset. The average of the obtained accuracy values is taken. The performance of the individual is determined based on the average value. The elitism and tournament selection strategies are implemented. In parent selection, individuals other than the elite ones are selected according to tournament selection. A crossover strategy is applied to the selected individuals. Mutation is applied to the individuals other than the elite ones according to the mutation rate, and a new generation is formed. This process is shown in Figure 4.3.

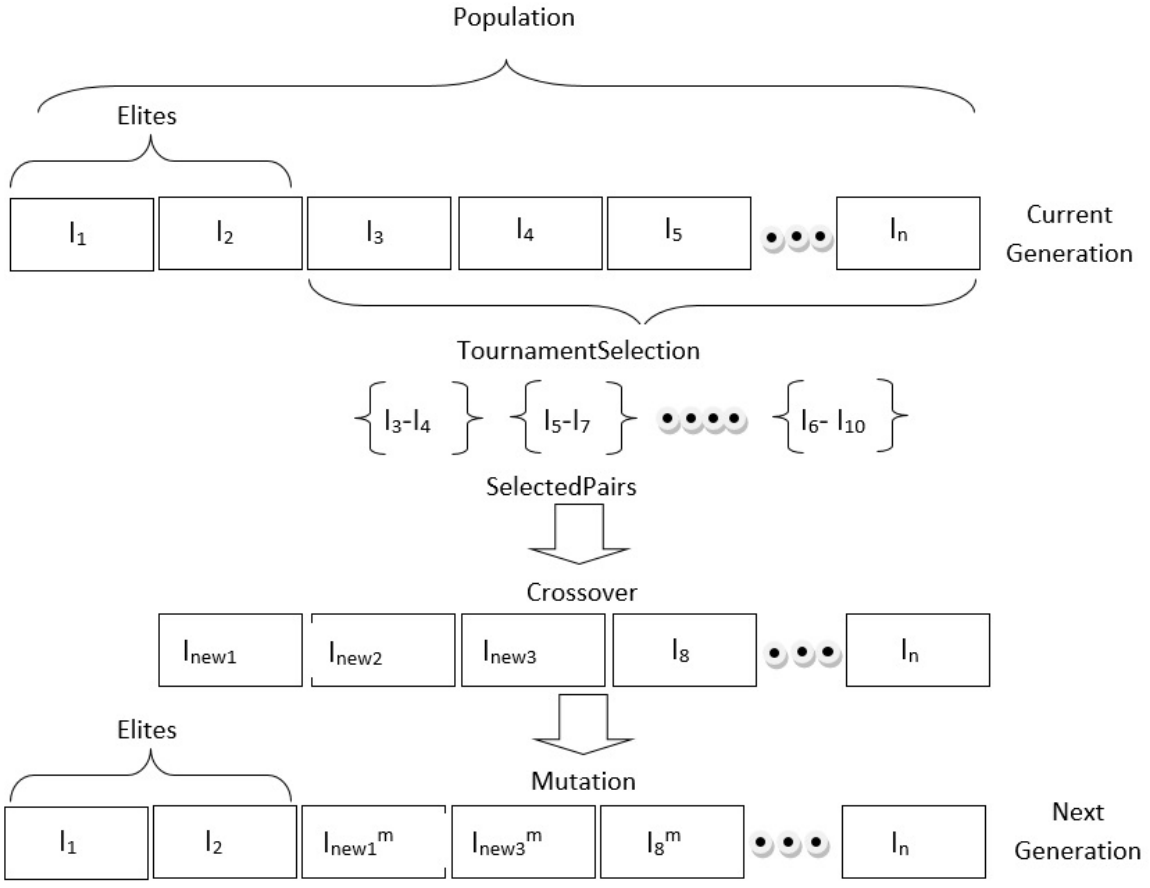

Figure 4.3 Genetic Optimization Process

4.3.1. Crossover

During the crossover process, both the models and masks of the individuals are subjected to processing. Different strategies are applied in the convolution and dense layers of the model [48].

Crossover for Conv2D Layers and Masks: If the layer is of Conv2D type, a random choice is made, and either the first or second chromosome's layer is added to the child model.

Crossover for Dense Layers: For dense layers, weights from chromosomes are taken and randomly merged. These new weights are assigned to the corresponding dense layer in the child model. The crossover strategy for dense layers is a random point crossover method, where each weight value (gene) is randomly selected column-wise and exchanged between two parents.

Other Layers: Layers that are neither Conv2D nor Dense are directly added to the child model. The process is explained in Figure 4.4.

4.3.2. Mutation

The body The mutation strategy applies mutations to randomly selected weights based on a certain probability and to randomly selected points in all masks. In the model, new weights are created by adding a randomly generated value with a normal distribution to the existing weights. In the masks, the value at the selected point is converted from 1 to 0 or from 0 to 1. The process is explained in Figure 4.5.

4.3.3. Binary Masking

When evaluating an individual, each dataset is element-wise multiplied with its own binary mask. This process deactivates some of the model's weights. Each dataset uses some weights commonly, while some are unique to itself. The model is evaluated based on the average of

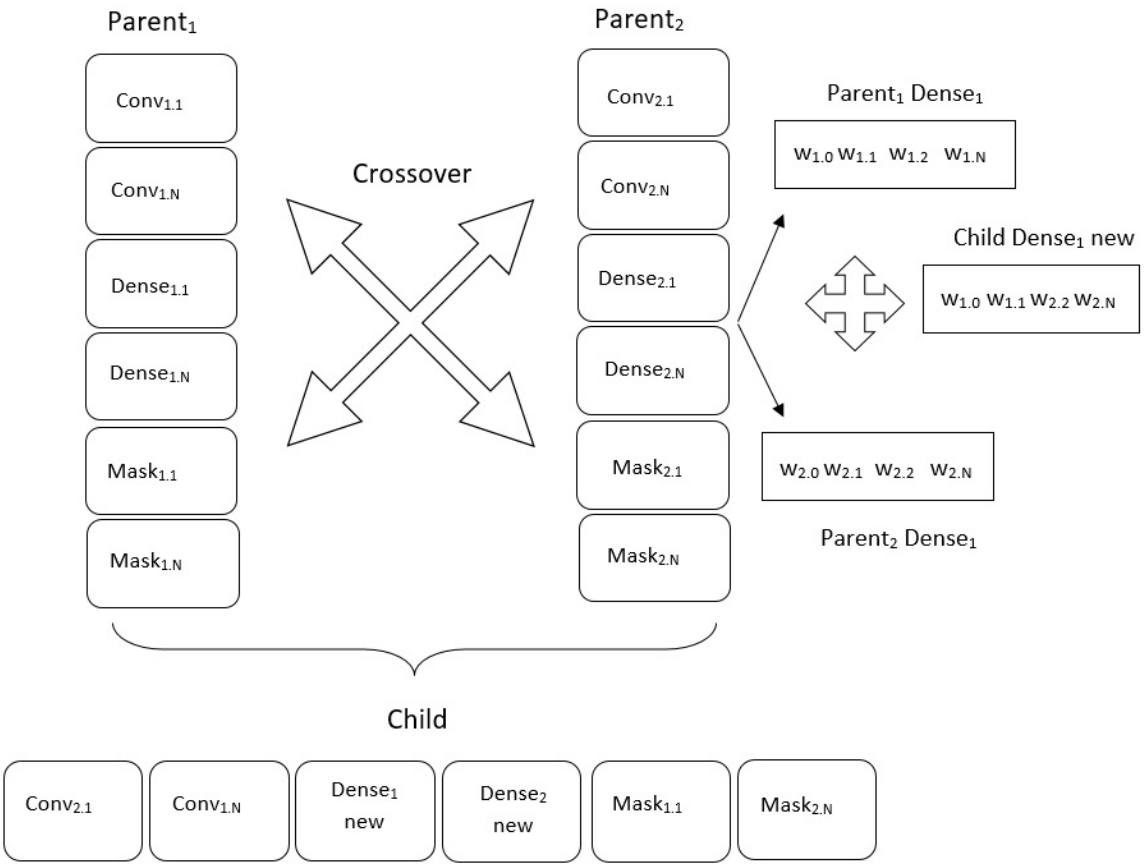

Figure 4.4 Genetic Crossover Process

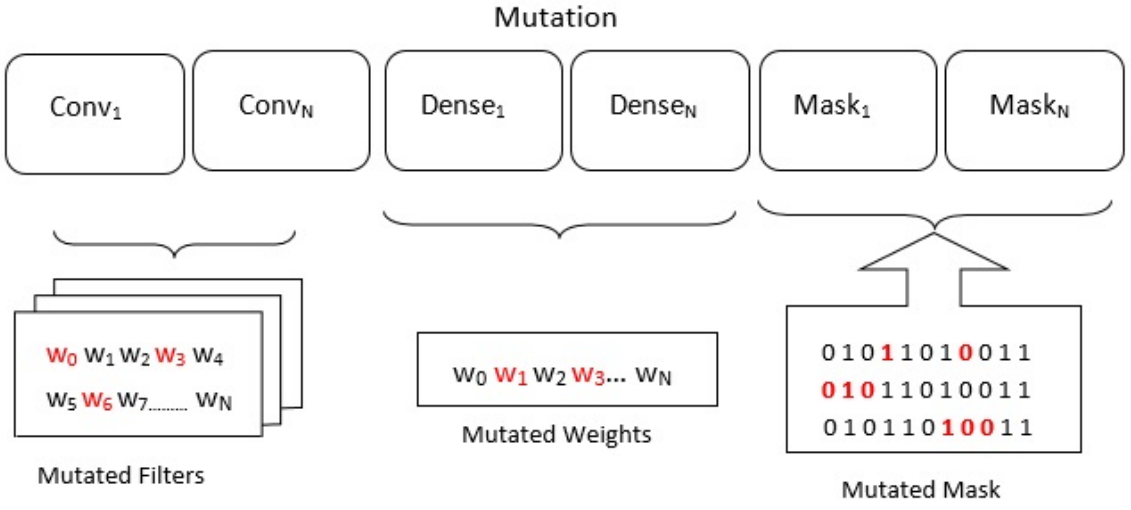

Figure 4.5 Genetic Mutation Process

the evaluation results, considering its average performance. The process is explained in Figure 4.6.

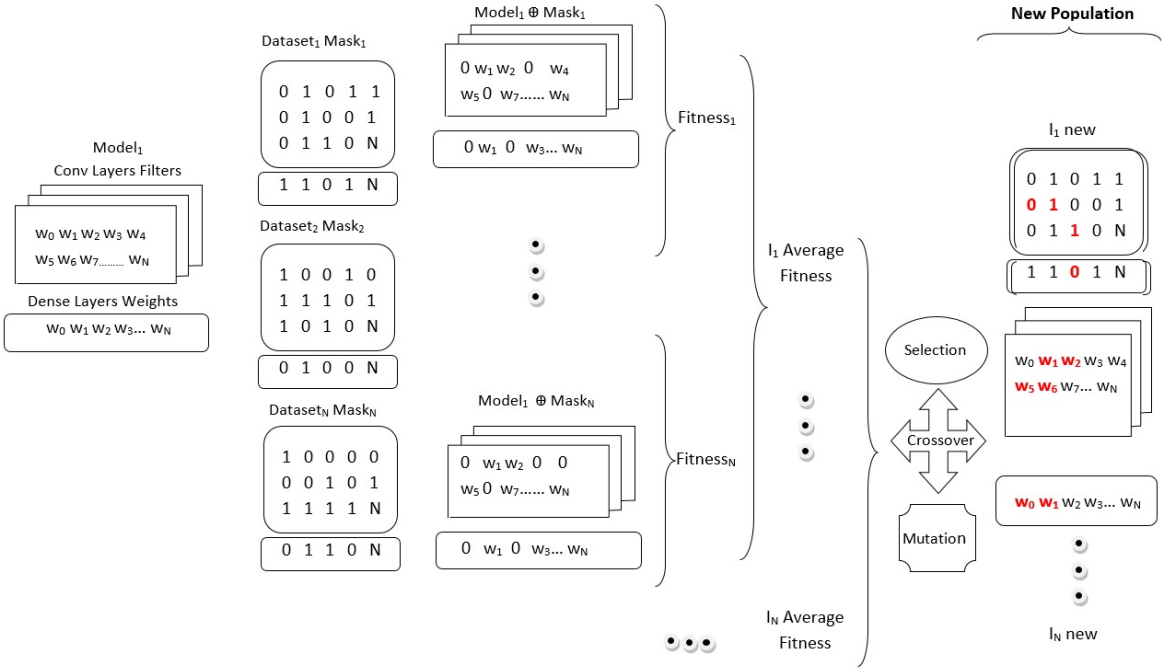

Figure 4.6 Binary Masking Process

The algorithm proposed in this work is given 4.7.

Algorithm 1: Genetic Algorithm and Binary Masks for Co-Learning Multiple Datasets in Deep Artificial Neural Networks

Input: CNN Model $C = \{Conv_1, Dense_1, \ldots, Conv_N, Dense_N\}$, $\{Conv_1, Dense_1 = \theta_1, \ldots, Conv_N, Dense_N = \theta_N\}$
Datasets $D = \{D_1, D_2, \ldots, D_N\}$
Masks $M = \{M_1, M_2, \ldots, M_N\}$
Individual $I = \{Conv_1, Dense_1, \ldots, Conv_N, Dense_N, M\}$
Hyperparameters: Population size= P_{size}, selected individuals= I_{selected}, elites= I_{elites}, mutation probability= mutprob, standard deviation for mutation= mutstd
Output: Optimized CNN Model C and Masks M
Best model and masks are selected based on the highest average accuracy across all datasets in D

Initialize population P_0 with P_{size} tuples: $\{(\theta_1, M_1), (\theta_2, M_2), \ldots, (\theta_{P_s}, M_{P_s})\}$ where each tuple contains a set of parameters θ_i representing a CNN model C and a set of mask $M_i = \{M_{i1}, M_{i2}, \ldots, M_{iN}\}$ corresponding to each D
for $g = 1$ *to max_generations* **do**
  for *each* (θ_i, M_i) *in* P_g **do**
    for *each* D_j *in* D **do**
      Apply mask M_{ij} to C with parameters θ_i using element-wise multiplication: $C(\theta_i) \odot M_{ij}$
      Evaluate accuracy $A(C(\theta_i), D_j)$
    Avg Accuracy$(\theta_i) \leftarrow \frac{1}{N} \sum_{j=1}^{N} A(C(\theta_i), D_j)$
  $I_{\text{elites}} \leftarrow P_g$
  $I_{\text{selected}} \leftarrow P_g$ except I_{elites}
  Perform tournament selection with I_{selected} to select individuals
  while *size of* $P_{new} < P_{size} - I_{elites}$ **do**
    Select parents (θ_{p1}, M_{p1}), (θ_{p2}, M_{p2}) randomly from $I_s elected$
    Initialize child parameters θ_c and mask M_c
    for *each layer* l *in* θ_{p1} *and* θ_{p2} **do**
      if l *is a Conv layer* **then**
        Select l from either θ_{p1} or θ_{p2} randomly for θ_c
      else if l *is a Dense layer* **then**
        Perform crossover of weights between θ_{p1} and θ_{p2} for θ_c
      else
        Add l from θ_{p1} to θ_c
    Select mask from either M_{p1} or M_{p2} randomly for M_c
    Add (θ_c, M_c) to P_{new}
  for *each* (θ_i, M_i) *in* $P_g - I_{elites}$ **do**
    Mutate θ_i parameters with probability mutprob and add noise with std mutstd
    for *each* M_{ij} *in* M_i **do**
      Mutate M_{ij} with probability mutprob by flipping bits
  $P_g \leftarrow P_{\text{new}}$
return *Best* C *and masks in* P_g

Figure 4.7 Optimization of Multiple Datasets in a CNN Model Using Binary Masks Through GA

5. EXPERIMENTAL RESULTS

5.1. Application

Our CNN model was implemented in Python and consists of four convolutional layers followed by two fully connected layers. The model's weights were initialized with the default starting values determined by Keras. Experimental computations have been performed on an NVIDIA A100 GPU provided by the Google Colab. The model used in the study is shown in table 5.1

Layers	**Output Shape**	**Number of Parameters**
Conv2D	(None, 28, 28, 32)	128
MaxPooling2D	(None, 14, 14, 32)	0
BatchNormalization	(None, 14, 14, 32)	64
Conv2D	(None, 14, 14, 32)	1024
MaxPooling2D	(None, 7, 7, 32)	0
BatchNormalization	(None, 7, 7, 32)	64
Conv2D	(None, 7, 7, 32)	1024
MaxPooling2D	(None, 3, 3, 32)	0
BatchNormalization	(None, 3, 3, 32)	64
Conv2D	(None, 3, 3, 16)	512
MaxPooling2D	(None, 3, 3, 16)	32
Flatten	(None, 144)	0
Dense	(None, 32)	4.608
Dense	(None, 10)	320
Total parameters		7.840

Table 5.1 Network Architecture Used for Three Datasets

5.2. Dataset Description

In this study, we use three widely-used datasets to test our deep learning model's ability to learn multiple tasks and overcome the catastrophic forgetting problem using GA and binary masks: MNIST, Fashion MNIST, and KMNIST. Each of these datasets has been chosen to test different aspects of our model's multi-task learning capability and its ability to address the catastrophic forgetting problem.

5.2.1. MNIST

The MNIST is a collection of handwritten digits. The MNIST consists of 60,000 training images and 10,000 test images. Each image is a 28x28 pixel grayscale image. Each image represents a single digit ranging from 0 to 9. In this study, we use the MNIST to test our model's ability to accurately classify digits. An example from the MNIST is shown in Figure 5.1.

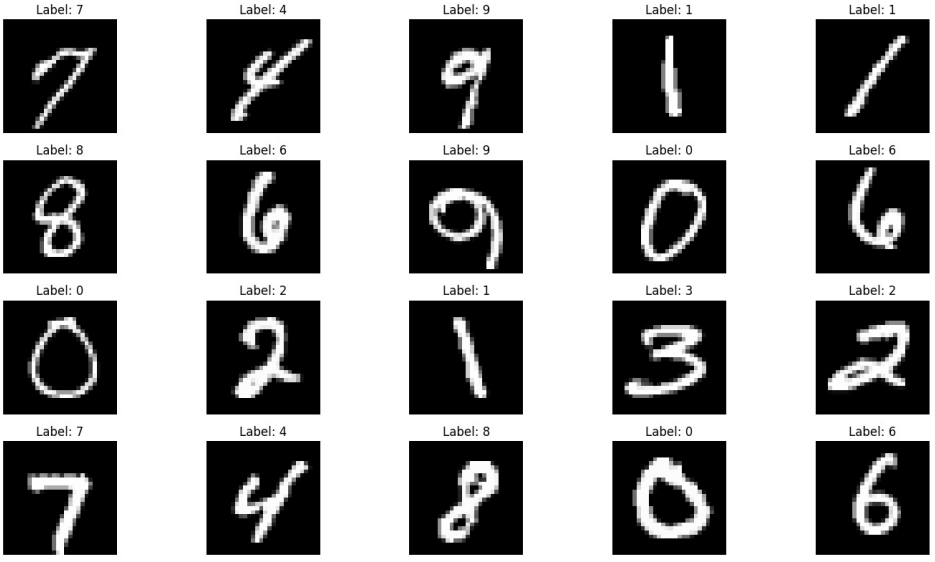

Figure 5.1 MNIST-Handwritten Digits

5.2.2. Fashion MNIST

Fashion MNIST is an alternative to the standard MNIST. The Fashion MNIST includes 60,000 training samples and 10,000 test samples. Each sample is a 28x28 grayscale image representing an article of clothing. There are 10 classes in the dataset (such as sweater, ankle boots, t-shirt, trousers, shirt, dress, bag, coat, sandals, sneakers). An example from the FMNIST is shown in Figure 5.2.

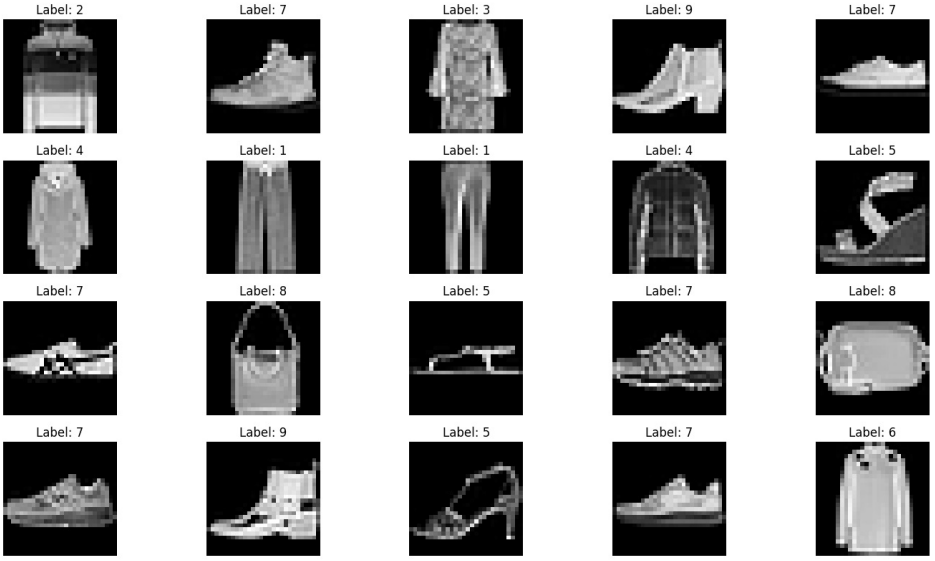

Figure 5.2 FMNIST-Articles of Clothing

5.2.3. KMNIST

The KMNIST is a collection of characters derived from ancient Japanese texts. The KMNIST consists of 60,000 training images and 10,000 test images. Each image is a 28x28 pixel grayscale image. In this dataset, each image represents a character from the classical Hiragana script of the Japanese language. In this study, we utilize the KMNIST to test our model's ability to accurately classify ancient Japanese script characters. An example from the KMNIST is shown in Figure 5.3.

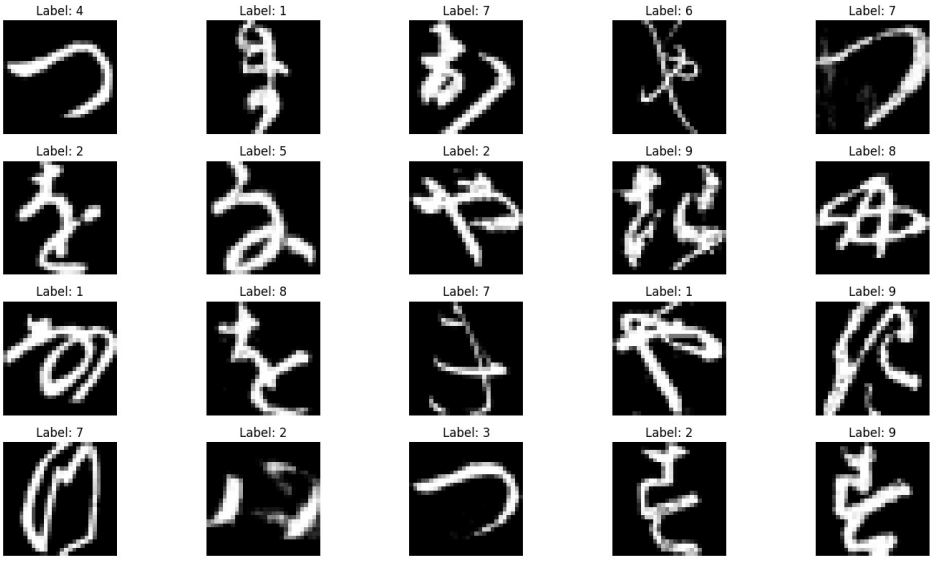

Figure 5.3 KMNIST-Collection of Characters Derived from Ancient Japanese Texts

5.3. Evaluation Metrics

In the field of artificial intelligence, evaluation metrics are used to measure the performance of algorithms. These metrics help us understand the success of the algorithm and its suitability for the application area. Some of the most commonly used metrics and their formulas are listed below.

5.3.1. Accuracy:

Measures the overall performance of a classification model on both positive and negative classes. The accuracy is represented by equation (14).

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total Predictions}} \tag{14}$$

5.3.2. Precision:

Indicates how many of the items classified as positive are actually positive. The precision is represented by equation (15).

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive + False Positive}} \tag{15}$$

5.3.3. Recall (Sensitivity):

Measures how much of the positive class is correctly identified. The recall is represented by equation (16).

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}} \tag{16}$$

5.3.4. F1 Score:

The harmonic mean of precision and recall, providing a balanced measurement. The F1 Score is represented by equation (17).

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{17}$$

5.3.5. ROC Curve and AUC:

Receiver Operating Characteristic (ROC) Curve is evaluates the performance of a classification model at various threshold levels. Area Under the Curve (AUC) is the area under the ROC curve quantitatively represents the classification performance of the model. Horizontal Axis (False Positive Rate - FPR) measures how many of the actual negative

instances are incorrectly classified as positive. Vertical Axis (True Positive Rate - TPR or Sensitivity) measures how many of the actual positive instances are correctly identified.

$$\text{FPR} = \frac{\text{False Positive}}{\text{False Positive + True Negative}} \tag{18}$$

$$\text{TPR} = \frac{\text{True Positive}}{\text{True Positive + False Negative}} \tag{19}$$

5.3.6. Mean Absolute Error (MAE):

In regression models, measures the average absolute difference between the predictions and the actual values. The MAE is represented by equation (20).

$$\text{MAE} = \frac{\sum |y_i - \hat{y}_i|}{n} \tag{20}$$

5.3.7. Root Mean Squared Error (RMSE):

The square root of the average of the squared differences between the actual and predictions values. It is represented by equation (21).

$$\text{RMSE} = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{n}} \tag{21}$$

These metrics are used to evaluate different types of AI models, and each highlights a different aspect of the model's performance. Accuracy, precision, recall, and F1 score are commonly used for classification problems, while MAE and RMSE are often used for regression problems.

In our research, the primary evaluation metric employed is classification accuracy. Accuracy was chosen due to its interpretability and because it is a standard metric for classification

problems. Moreover, in the optimization process using GA, accuracy was utilized as the fitness function. This approach enabled iterative improvements focused on accuracy to maximize the model's performance.

5.4. Experimental Results and Performance Evaluation

For optimization, the population size was set to 20. The models optimized with the genetic algorithm were trained and tested on each dataset for 2000 iterations. The best model achieved the performance shown in in table 5.2

Dataset	**Test Accuracy (%)**	**Test Loss**
MNIST	76,25	0.6927
Fashion MNIST	76,00	0.6912
KMNIST	74,43	0.7891

Table 5.2 Test Accuracy and Loss Values for Various Datasets Trained with the GA

Figure 5.4 and 5.5 shows the accuracy and loss values achieved by the best individual in the population at each iteration for three different datasets (MNIST, FMNIST, and KMNIST), as well as the changes in average accuracy and loss values for these dataset over the iterations.

As seen in figure 5.4 and 5.5, the performance of the MNIST is better compared to the other two datasets. There can be several reasons for this; MNIST is containing handwritten digits and involves relatively simpler and easier-to-separate examples. Therefore, it is an expected result for the model to achieve higher accuracy on this dataset. FMNIST and KMNIST are more complex datasets and hence have lower accuracy values. FMNIST includes items of clothing and KMNIST contains Japanese handwritten characters; both present more challenging recognition problems compared to MNIST.

As the iterations increase, accuracy values for all datasets have generally improved, indicating the effectiveness of GA in optimizing the weights. The average accuracy value

reflects the model's overall performance across different datasets and shows a measure of consistent improvement.

To address the catastrophic forgetting problem, binary masking methods used (sharing weights) have helped in transferring knowledge across various datasets, enabling the model to learn more general features and thus mitigating the catastrophic forgetting issue. Optimizing weights through genetic algorithm and using a separate mask for each dataset has helped in preserving dataset-specific weights. This method has been effective in reducing the catastrophic forgetting problem as it prevents overwriting weights that are important for each dataset.

Since all dataset are similar in size and structure, the model has been able to apply low-level features (like edges and corners) learned from the first dataset to the others. This has helped the model adapt quickly and achieve a better starting point. On the other hand, due to the different visual contents represented by the datasets, it is more challenging for the model to transfer high-level features between these datasets.

Starting the training with MNIST and then moving to FMNIST and KMNIST has enabled the model to embark on a learning journey from simpler to more complex. This has helped in enhancing the overall modeling capability, as the model first learns less complex features and then generalizes these features to more complex dataset.

These results demonstrate a significant improvement in classification accuracy in addressing the catastrophic forgetting problem when using GA-based mask optimization, thereby validating our methodology and approach.

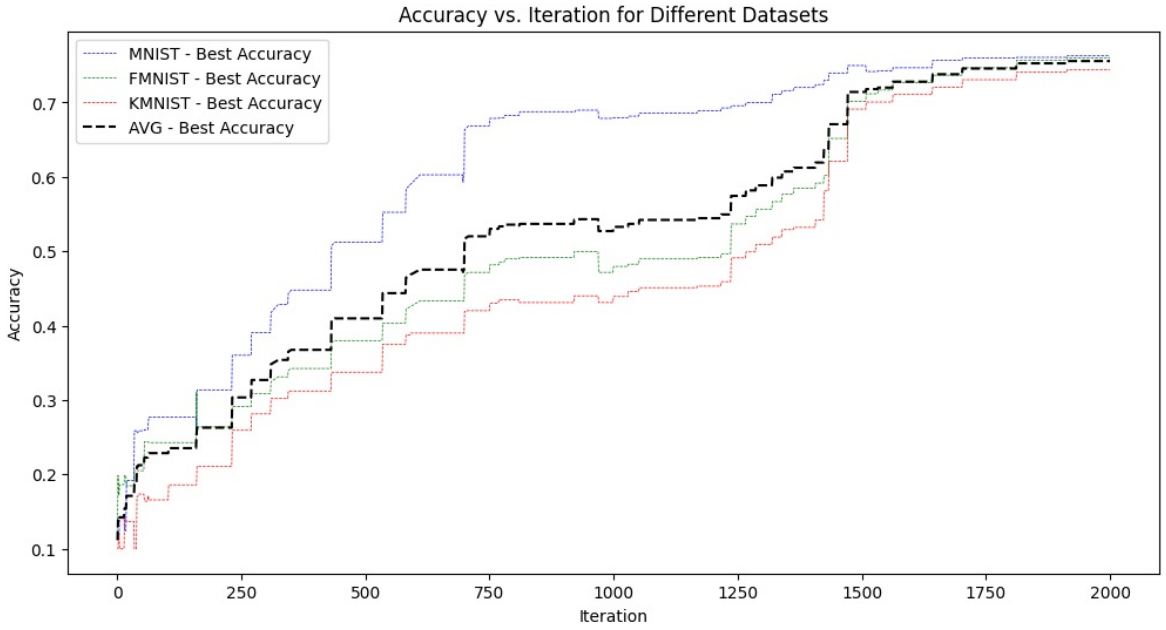

Figure 5.4 The Accuracy Values for the Best Individual in the Population at Each Iteration for Three Datasets

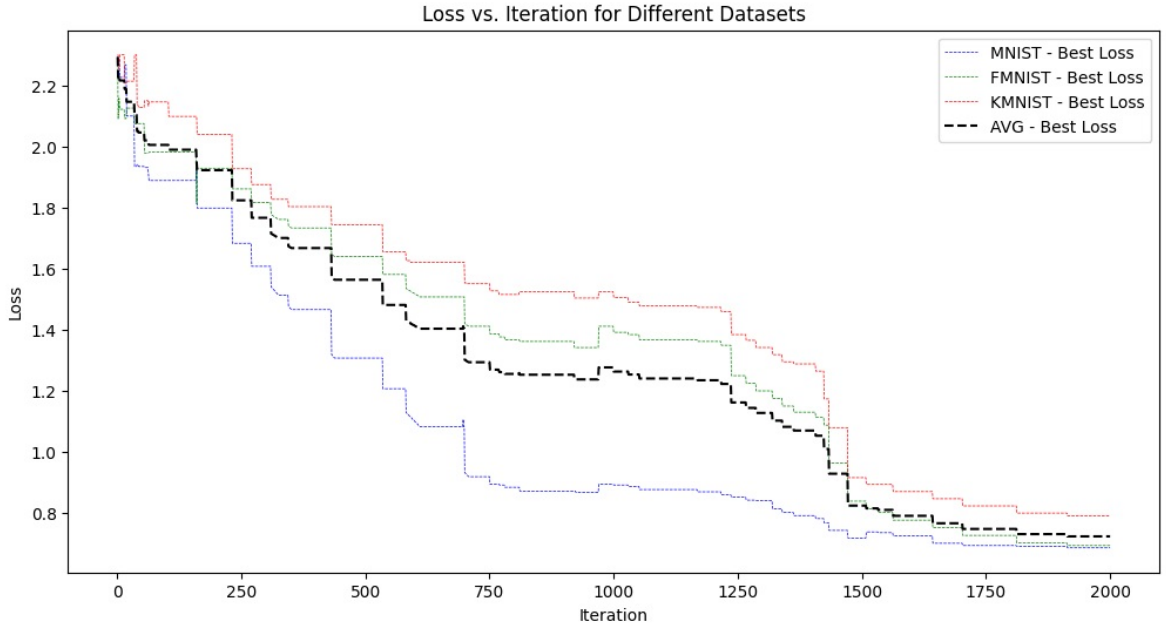

Figure 5.5 The Loss Values for the Best Individual in the Population at Each Iteration for Three Datasets

For comparison, the model was trained limited to 100 iterations without mask optimization

using traditional derivative-based methods (such as ADAM). The performance of the model on different dataset is shown in table 5.3

Dataset	Test Accuracy (%)	Test Loss
MNIST	74,02	0.7707
Fashion MNIST	73,37	0.7051
KMNIST	10,00	2.3026

Table 5.3 Test Accuracy and Loss Values for Various Datasets Trained with the ADAM Optimizer

Figure 5.6 and 5.7 illustrates the learning process conducted on three different datasets (MNIST, FMNIST, and KMNIST) using the widely preferred ADAM optimizer, which offers adaptive learning rates. The accuracy (figure 5.6) and loss (figure 5.7) values, as well as the changes in average accuracy and loss values for these datasets, are displayed for comparison, limited to 100 iterations. Experimental computations are conducted on Google Colab environment using NVIDIA A100 GPU. In GA optimization, an iteration with a population size of 20 takes approximately 10 minutes on average. To ensure a proper comparison with ADAM optimization, the population size of 20 is maintained. Under these conditions, one iteration takes an average of 4 minutes in the ADAM optimization.

Thanks to the adaptive learning capabilities of the ADAM optimizer, it has been observed that the model quickly achieved high accuracy rates in the training process for the MNIST and FMNIST. In the early stages of the training process, a rapid increase in accuracy rates was observed for MNIST and FMNIST, after which these values settled at a stable level. However, for the KMNIST, it was observed that the model did not show any improvement over the initial low accuracy rates. This indicates that the model was insufficient in learning the complexity of this particular dataset. Additionally, the absence of mask optimization has negatively affected the model's generalization performance.

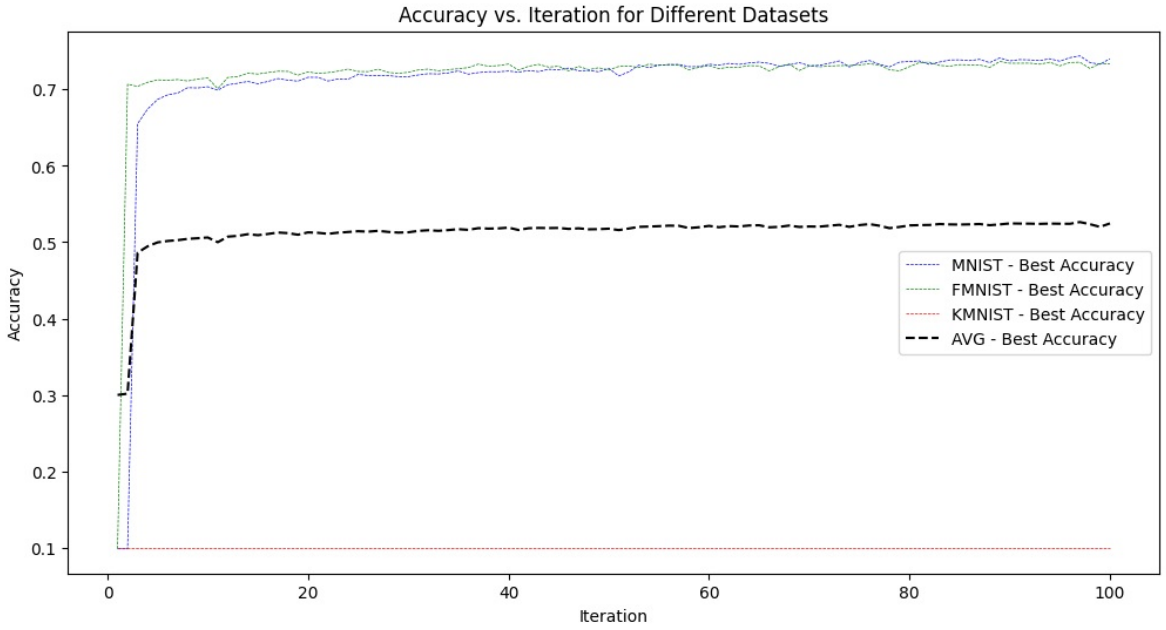

Figure 5.6 The Accuracy Values for Three Datasets Trained with the ADAM Optimizer

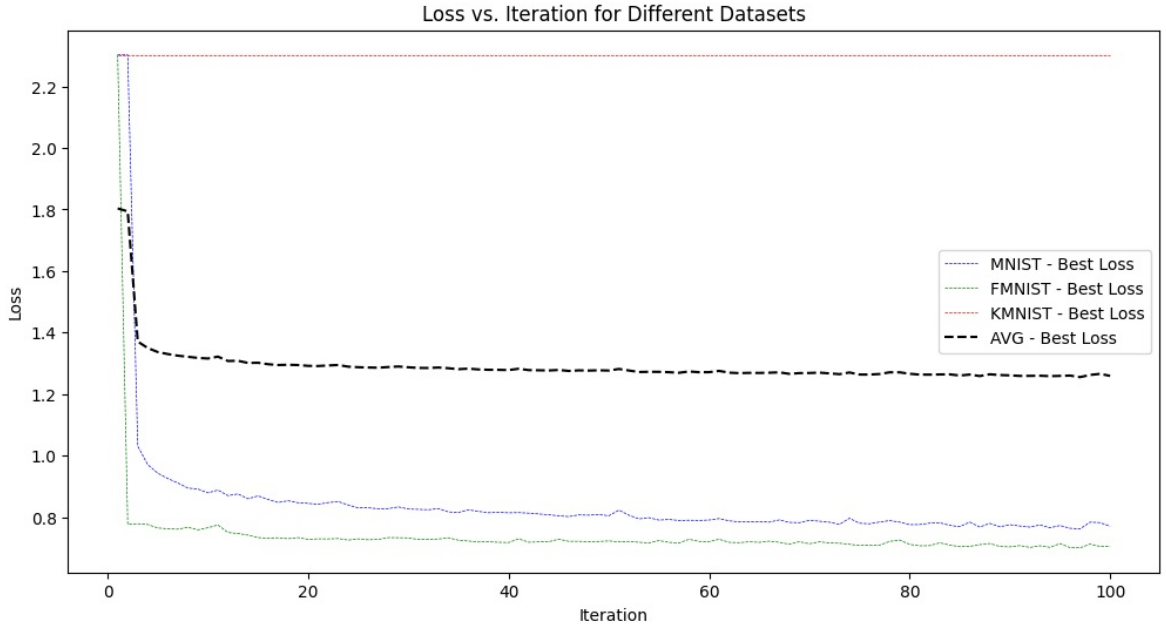

Figure 5.7 The Loss Values for Three Datasets Trained with the ADAM Optimizer

Figure 5.8 and 5.9 shows the accuracy and loss values for the first 100 iterations of GA optimization, for comparison purposes.

The optimization conducted with GA appears to be more successful compared to the ADAM optimizer, especially in terms of the results on the KMNIST. KMNIST is consisting of Japanese characters and is more complex than MNIST. When using the ADAM optimizer, no improvement was observed on the KMNIST, whereas some improvement in accuracy was noted when using GA. This suggests that GA, by optimizing the model's weights and masks simultaneously, can enhance the extraction of more complex features and generalization on specific dataset.

GA reduces the risk of getting stuck in local minima by creating a broad search space for masks and weights. This has allowed the model to better generalize on more challenging dataset like KMNIST. The ADAM optimizer tends to get stuck in local minima, especially when suitable steps are not taken during weight updates. On the other hand, the diversification of different individuals through genetic operations in GA has helped the model avoid local minima, thereby improving overall performance.

These results indicate that GA could be an approach capable of enhancing the performance of models, especially on complex and more challenging dataset, even in situations where traditional optimizers fail. However, GA also has disadvantages, such as higher computational cost, and may not be ideal for every use case.

Additionally, the success level of GA can vary depending on the type of genetic operations chosen, the crossover rate, mutation rate, and population size, and thus it should be carefully adjusted.

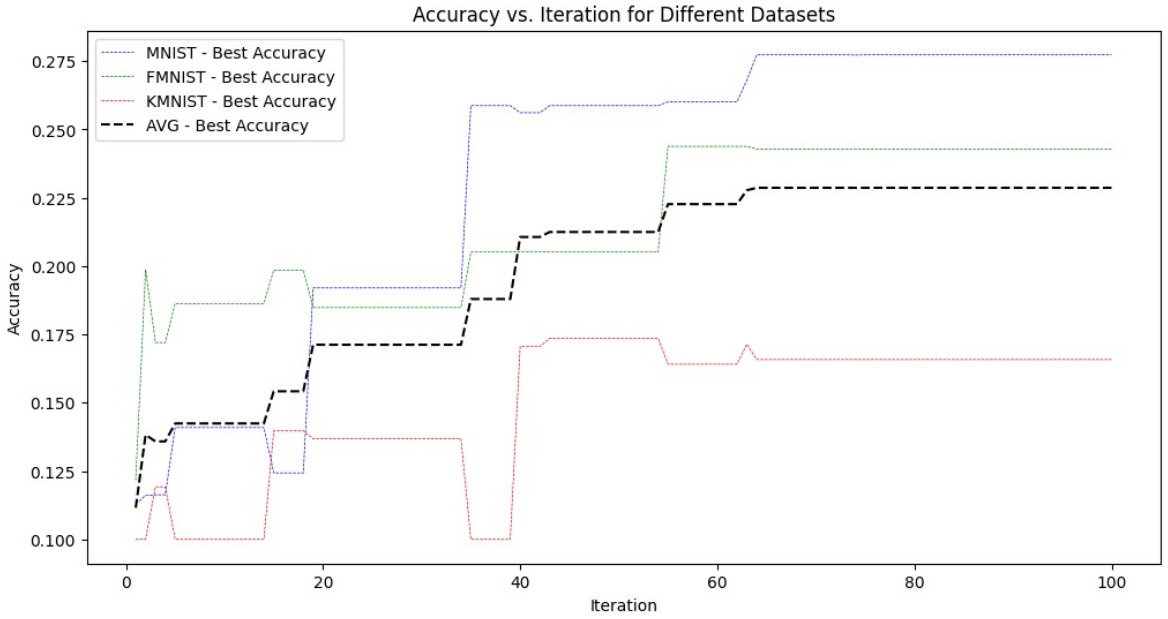

Figure 5.8 The Accuracy Values for Three Datasets Trained with the GA

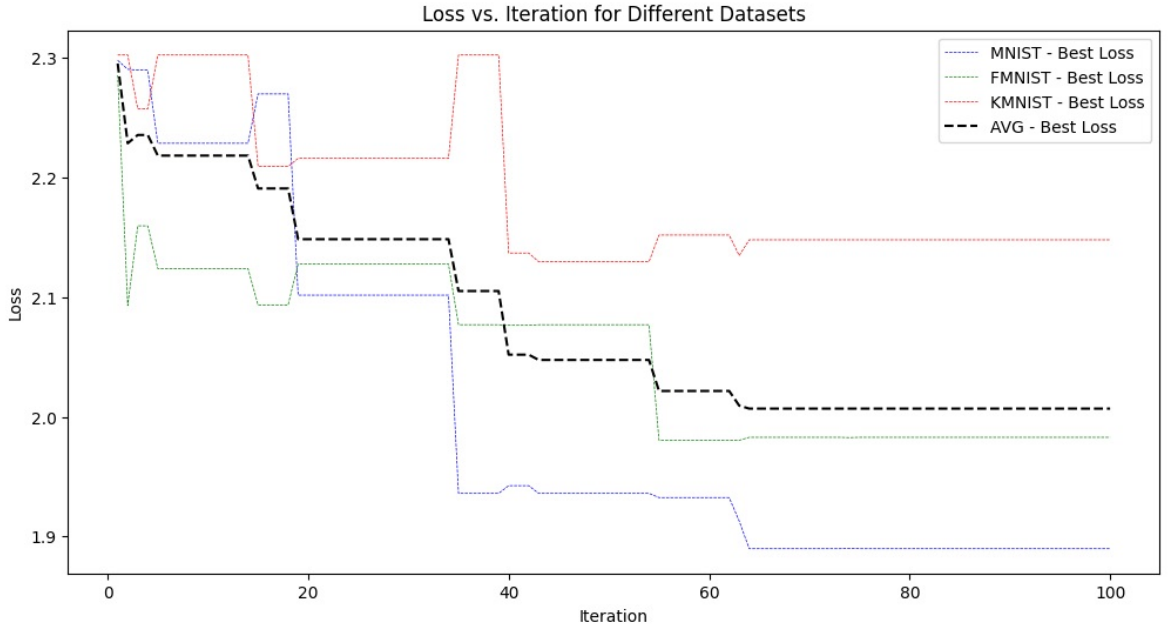

Figure 5.9 The Loss Values for Three Datasets Trained with the GA

6. CONCLUSION

This study presents a novel approach aimed at simultaneously learning multiple datasets with a deep learning model. The proposed method has been tested on widely used datasets and has achieved successful results. The approach's ability to generalize and its capacity for concurrent learning are promising. However, tuning of the method's parameters and more extensive experiments are required. This study offers a pathway to accelerate the learning process and improve generalization ability by combining deep learning and genetic algorithms. The results indicate significant strides in enabling deep learning models to learn multiple tasks at the same time. However, there are limitations to the method, such as the need for parameter tuning and computational intensity.

This study presents a new approach to the catastrophic forgetting problem. Future work should examine the scaling potential on larger datasets and complex tasks. Additionally, the applicability to different deep learning models should also be investigated.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, **2015**.

[2] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25. **2012**.

[3] M. McCloskey and N.J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Academic Press, **1989**.

[4] R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, **1997**.

[5] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A.A. Rusu, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, **2017**.

[6] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, **2017**.

[7] Z. Li and D. Hoiem. Learning without forgetting. In *European Conference on Computer Vision*. ECCV, **2016**.

[8] J.S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, **1985**.

[9] Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximally interfered retrieval. In *NeurIPS*. **2019**.

[10] X. Jin, A. Sadhu, J. Du, and X. Ren. Gradient-based editing of memory examples for online task-free continual learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 29193–29205. **2021**.

[11] A. Chaudhry, A. Gordo, P. Dokania, P. Torr, and D. Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6993–7001. **2021**.

[12] D. Lopez-Paz and M.A. Ranzato. Gradient episodic memory for continual learning. *Advances in Neural Information Processing Systems*, 30, **2017**.

[13] B. Cheung, A. Terekhov, Y. Chen, P. Agrawal, and B. Olshausen. Superposition of many models into one. *Advances in Neural Information Processing Systems*, 32, **2019**.

[14] M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, **2020**.

[15] D. Abati, J. Tomczak, T. Blankevoort, S. Calderara, R. Cucchiara, and B.E. Bejnordi. Conditional channel gated networks for task-aware continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3931–3940. **2020**.

[16] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *NeurIPS*. **2019**.

[17] A.A. Rusu, N.C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, **2016**.

[18] A.V. Terekhov, G. Montone, and J.K. O'Regan. Knowledge transfer in deep block-modular neural networks. In *Conference on Biomimetic and Biohybrid Systems*, pages 268–279. Springer, Cham, **2015**.

[19] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773. **2018**.

[20] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NIPS*. **2015**.

[21] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, et al. Dsd: Densesparse-dense training for deep neural networks. In *ICLR*. **2017**.

[22] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *European Conference on Computer Vision*. ECCV, **2018**.

[23] J. Serra, D. Suris, M. Miron, and A. Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557. PMLR, **2018**.

[24] S. Golkar, M. Kagan, and K. Cho. Continual learning via neural pruning. *ArXiv, abs/1903.04476*, **2019**.

[25] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, **2014**.

[26] E. Ergün and B.U. Töreyin. Continual learning with sparse progressive neural networks. In *2020 28th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, **2020**.

[27] C. Fernando, D.S. Banarse, C. Blundell, Y. Zwols, D.R. Ha, A.A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *ArXiv, abs/1701.08734*, **2017**.

[28] J.N. Gupta and R.S. Sexton. Comparing backpropagation with a genetic algorithm for neural network training. *Omega*, 27(6):679–684, **1999**.

[29] R.S. Sexton and J.N. Gupta. Comparative evaluation of genetic algorithm and backpropagation for training neural networks. *Information Sciences*, 129(1-4):45–59, **2000**.

[30] J.P. Ignizio and J.R. Soltys. Simultaneous design and training of ontogenic neural network classifiers. *Computers and Operations Research*, 23(6):535–546, **1996**.

[31] A. Bakhshi, N. Noman, Z. Chen, M. Zamani, and S. Chalup. Fast automatic optimisation of cnn architectures for image classification using genetic algorithm. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 1283–1290. IEEE, **2019**.

[32] K. Pawelczyk, M. Kawulok, and J. Nalepa. Genetically-trained deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. **2018**.

[33] J. David Schaffer, Darrell Whitley, and Larry J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE, **1992**.

[34] Omid E. David and Iddo Greental. Genetic algorithms for evolving deep neural networks. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. **2014**.

[35] Gregory Morse and Kenneth O. Stanley. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. In *2016 Proceedings of the Genetic and Evolutionary Computation Conference*. **2016**.

[36] Esteban Real et al. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*. PMLR, **2017**.

[37] Felipe Petroski Such et al. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, **2017**.

[38] Emmanuel Dufourq and Bruce A. Bassett. Eden: Evolutionary deep networks for efficient machine learning. In *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*. IEEE, **2017**.

[39] Edgar Galván and Peter Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence*, 2(6):476–493, **2021**.

[40] F. Takeda and S. Omatu. A neuro-paper currency recognition method using optimized masks by genetic algorithm. In *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century*, volume 5, pages 4367–4371. IEEE, **1995**.

[41] Y. Tian, X. Zhang, C. Wang, and Y. Jin. An evolutionary algorithm for large-scale sparse multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 24(2):380–393, **2019**.

[42] M. Mancini, E. Ricci, B. Caputo, and S. Rota Bulo. Adding new tasks to a single network with weight transformations using binary masks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0. **2018**.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, **2014**.

[44] F. Medhat, D. Chesmore, and J. Robinson. Masked conditional neural networks for environmental sound classification. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 21–33. Springer International Publishing, Cham, **2017**.

[45] Z. Niu, G. Zhong, and H. Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, **2021**.

[46] A. Gu, C. Gulcehre, T. Paine, M. Hoffman, and R. Pascanu. Improving the gating mechanism of recurrent neural networks. In *International Conference on Machine Learning*, pages 3800–3809. PMLR, **2020**.

[47] E. Ben-Iwhiwhu, S. Nath, P.K. Pilly, S. Kolouri, and A. Soltoggio. Lifelong reinforcement learning with modulating masks. *arXiv preprint arXiv:2212.11110*, **2022**.

[48] Parsa Esfahanian and Mohammad Akhavan. Gacnn: Training deep convolutional neural networks with genetic algorithm. *arXiv preprint arXiv:1909.13354*, **2019**.