Studies in Big Data 12

M. Arif Wani
Bisma Sultan
Sarwat Ali
Mukhtar Ahmad Sofi

Advances in Deep Learning, Volume 2

Springer

Studies in Big Data

Volume 12

The series “Studies in Big Data” (SBD) publishes new developments and advances in the various areas of Big Data- quickly and with a high quality. The intent is to cover the theory, research, development, and applications of Big Data, as embedded in the fields of engineering, computer science, physics, economics and life sciences. The books of the series refer to the analysis and understanding of large, complex, and/or distributed data sets generated from recent digital sources coming from sensors or other physical instruments as well as simulations, crowd sourcing, social networks or other internet transactions, such as emails or video click streams and other. The series contains monographs, lecture notes and edited volumes in Big Data spanning the areas of computational intelligence including neural networks, evolutionary computation, soft computing, fuzzy systems, as well as artificial intelligence, data mining, modern statistics and Operations research, as well as self-organizing systems. Of particular value to both the contributors and the readership are the short publication timeframe and the world-wide distribution, which enable both wide and rapid dissemination of research output.

The books of this series are reviewed in a single blind peer review process.

Indexed by SCOPUS, EI Compendex, SCIMAGO and zbMATH.

All books published in the series are submitted for consideration in Web of Science.

M. Arif Wani · Bisma Sultan · Sarwat Ali ·
Mukhtar Ahmad Sofi

Advances in Deep Learning, Volume 2

Springer

M. Arif Wani
Department of Computer Science
University of Kashmir
Srinagar, Jammu and Kashmir, India

Bisma Sultan
Department of Computer Science
University of Kashmir
Srinagar, Jammu and Kashmir, India

Sarwat Ali
Department of Computer Science
University of Kashmir
Srinagar, Jammu and Kashmir, India

Mukhtar Ahmad Sofi
BVRIT Hyderabad College of Engineering
for Women
Hyderabad, Telangana, India

ISSN 2197-6503 ISSN 2197-6511 (electronic)
Studies in Big Data
ISBN 978-981-96-3497-2 ISBN 978-981-96-3498-9 (eBook)
https://doi.org/10.1007/978-981-96-3498-9

Preface

Deep learning has revolutionized the Artificial Intelligence (AI) field, offering unparalleled performance across a variety of complex tasks. Its ability to automatically learn features from vast amounts of data makes it one of the suitable learning techniques for numerous applications. Several areas and applications of deep learning have witnessed significant progress, this includes Neural Architecture Search (NAS), Steganography, and Medical Applications.

Neural Architecture Search has significantly contributed to the design of deep learning models by automating the process of architecture creation. Traditionally, designing deep learning networks required expert knowledge and trial-and-error methods, which were both time-consuming and limited by human intuition. NAS overcomes these limitations by employing algorithms to explore a vast search space of potential architectures and selecting the best architecture.

Steganography, the practice of hiding information within digital media, has long been used for secure communication. Traditional methods of steganography were often susceptible to detection or offered limited capacity for hidden data. However, with the advent of deep learning, new techniques have emerged that significantly improve the robustness, capacity, and security of hidden information.

The ability to analyze vast amounts of medical data has opened new frontiers in healthcare, encompassing disease diagnosis, personalized medicine, and drug discovery. Deep learning models are now integral to interpreting medical images, predicting patient outcomes, and identifying potential therapeutic targets, significantly enhancing both accuracy and efficiency of healthcare services.

This book aims to explore these three areas, highlighting the significant work accomplished, the impact made, and the future research scope in these areas. The book is organized into thirteen chapters.

Chapter 1 discusses the impact of deep learning in three important areas: Neural Architecture Search (NAS), Steganography, and Medical Applications. The chapter introduces NAS, an approach that automates the neural network architecture search process, resulting in the design of efficient and high-performance models. It then discusses steganography, where impact of advanced methods based on deep learning for secure data embedding within digital media is outlined. The chapter highlights

the achievements of deep learning in the medical field. The chapter also outlines the future research directions in these three important areas.

Chapter 2 explores the fundamental concepts of Evolutionary Algorithm-Based Neural Architecture Search (NAS). By employing principles of natural selection, evolutionary NAS iteratively evolves optimal neural architectures across generations, effectively exploring a vast search space of architectures. The integration of techniques such as mutation, crossover, and selection enhances the diversity and adaptability of architectures for complex tasks like image classification. The chapter also evaluates the performance of various evolutionary NAS methods, comparing their effectiveness and identifying promising avenues for future research and development.

Chapter 3 focuses on Gradient-Based Neural Architecture Search, an approach that automates the design of neural network architectures. It examines the DARTS methodology in-depth, a foundational gradient-based NAS technique that formulates architecture search as a continuous optimization problem. An experimental analysis showcases the efficiency and effectiveness of gradient-based NAS, highlighting its practical applications. The chapter concludes with a discussion on future directions for research, underscoring the importance of balancing accuracy, efficiency, and computational costs in advancing this powerful architecture search paradigm.

Chapter 4 presents a new training methodology aimed at improving the performance of deep learning models. The approach utilizes a coarse-to-fine-tuning strategy that incorporates selective freezing techniques, specifically Simple Selective Freezing (SSF) and Progression-Based Selective Freezing (PSF). Initially, coarse training is performed on deep learning architectures, followed by the application of these selective freezing methods to fine-tune the model. This approach can be applied to architectures obtained either manually or through Neural Architecture Search (NAS) methods. The experiments on the CIFAR-10 dataset, using an architecture derived from DARTS, reveal that the coarse-to-fine-tuning approach outperforms traditional training methods.

Chapter 5 discusses various Generative Adversarial Networks (GANs) architectures that have been used in image steganography. Generative Adversarial Networks have gained considerable attention in image steganography mainly because these networks can encode and decode secret information using digital images efficiently. Various GAN-based techniques to embed and extract secret data seamlessly using images, offering a robust solution for secure communication and data concealment, are discussed.

Chapter 6 discusses advanced deep learning-based image steganography techniques. The three key parameters (security, embedding capacity, and invisibility) for measuring the quality of an image steganographic technique are described. Various steganography techniques, with emphasis on the above three key parameters, are described. The results reported by researchers on benchmark datasets, CelebA, Bossbase, PASCAL-VOC12, CIFAR-100, ImageNet, and USC-SIPI, are used to evaluate the performance of various steganography techniques. Analysis of the results shows that there is scope for new suitable deep learning architectures that can improve the capacity and invisibility of image steganography.

Chapter 7 discusses a new two-stage depth-balanced GAN architecture designed for steganography. The new architecture addresses the inherent challenges in steganographic systems by leveraging the strengths of GANs to achieve a balance between data embedding capacity and imperceptibility. The approach improves the embedding capacity and it is robust against steganalysis, making it more secure than the traditional steganography methods.

Chapter 8 describes a new approach for multiple secret messages image steganography. The approach can conceal multiple secret messages for multiple recipients in a cover image. A separate extractor model is used to extract the secret message intended for each receiver. The generator model is developed using a two-stage approach. A flattened cover image is used in the first stage to generate the reconstructed cover image. The output of the first stage is concatenated with the multiple secret messages (intended for multiple receivers) before being input to the second stage of the generator network to produce the stego image.

Chapter 9 explores the potential of deep learning in computational biology and healthcare through three distinct case studies involving protein secondary structure prediction, pan-cancer classification using gene expression data, and brain tumor detection from MRI scans. It highlights the effectiveness of cutting-edge architectures such as CNNs and RNNs in improving accuracy and efficiency for these complex tasks.

Chapter 10 discusses selected deep learning architectures tailored for handling sequence and image data. It delves into Convolutional Neural Networks (CNNs) for visual tasks, Recurrent Neural Networks (RNNs) for sequence data, and their advanced variants such as LSTMs and GRUs, addressing the challenges of long-term dependencies. It emphasizes the practical applications of these architectures in bioinformatics and medical diagnostics, showcasing their versatility in addressing complex real-world problems.

Chapter 11 proposes a new hybrid deep learning architecture integrating inception modules, Bidirectional Gated Recurrent Units (BGRUs), and attention mechanisms to enhance Protein Secondary Structure Prediction (PSSP). Experimental evaluations on benchmark datasets, including CB6133, CB513, CASP10, and CASP11, demonstrate significant improvements in accuracies, underscoring the potential of the proposed method to advance protein secondary structure prediction.

Chapter 12 presents an ensemble deep learning approach for classifying cancer types using gene expression data from the Pan-Cancer Atlas. To enhance prediction accuracy, diverse aggregation strategies such as max voting, average voting, and weighted average voting are employed. The study addresses key challenges, including handling data imbalance through weight balancing. The approach highlights the potential of ensemble methods to advance pan-cancer classification and improve genomic analysis.

Chapter 13 presents a lightweight deep learning approach for classifying brain tumors using T1-weighted MRI scans. The approach integrates transfer learning for

enhanced feature extraction and computational efficiency. It highlights the effectiveness of lightweight models in real-time medical image analysis on resource-constrained devices.

Srinagar, India M. Arif Wani
Srinagar, India Bisma Sultan
Srinagar, India Sarwat Ali
Hyderabad, India Mukhtar Ahmad Sofi

Contents

About the Authors

Dr. M. Arif Wani received an M.Tech. degree in Computer Technology from the Indian Institute of Technology, Delhi, and a Ph.D. degree in Computer Vision from Cardiff University, UK. Currently, he is a Professor at the University of Kashmir, having previously served as a Professor at California State University, Bakersfield. His research interests are in the area of machine learning, with a focus on neural networks, deep learning, inductive learning, support vector machines, computer vision, pattern recognition, and classification tasks. He has published many papers in reputed journals and conferences in these areas. Dr. Wani has co-authored the book *Advances in Deep Learning*, and co-edited many books on Machine Learning and Deep Learning applications. He is a member of many academic and professional bodies.

Bisma Sultan earned a B.Tech. in Computer Science and Engineering from the University of Kashmir, an M.Tech. in Computer Science from the University of Jammu, and completed her Ph.D. in Computer Science at the University of Kashmir. She qualified National Eligibility Test (NET) and the Graduate Aptitude Test in Engineering (GATE). She served as a Senior Research Fellow in the Department of Computer Sciences at the University of Kashmir. Currently, she holds a faculty position within the Department of Computer Sciences at the University of Kashmir. Dr. Bisma's research interests encompass a broad spectrum of fields, including machine learning, deep learning, information security, web technologies, and networking. She has made contributions to these areas by publishing a number of research articles in high-ranking academic journals and conferences.

Sarwat Ali has a Bachelor's and Master's degree in Computer Applications from the University of Kashmir, Hazratbal, Srinagar. She has worked as a Research Associate in the Artificial Intelligence Research Center project supported by Rashtriya Uchchatar Shiksha Abhiyan (RUSA 2.0), an initiative of the Government of India. Currently, she is dedicated to her Ph.D. studies in Neural Architecture Search (NAS) at the Post Graduate Department of Computer Science (PGDCS), University of Kashmir. She has contributed to publications in prestigious journals and research

conferences in this specialized field. Sarwat's academic journey has been marked by achievements, including the receipt of gold medals for both her Bachelor's and Master's degrees from the University of Kashmir. She has also achieved success in the (University Grants Commission National Eligibility Test) UGC NET examination.

Mukhtar Ahmad Sofi received MCA and M.Tech. in Computer Science and engineering degrees from Pondicherry Central University and a Ph.D. in Computer Science from the University of Kashmir. Currently, he is an Assistant Professor in the Information Technology Department at the BVRIT Hyderabad College of Engineering for Women. Dr. Sofi's diverse research interests span data mining, machine learning, deep learning, natural language processing, computational biology, and bioinformatics. He has made notable contributions to these areas through numerous research articles published in prestigious academic journals and presentations at internationally acclaimed conferences. Dr. Sofi continues to make impactful contributions to the academic and research community.

Chapter 1
Introduction to Deep Learning Applications

1.1 Introduction

Deep learning has emerged as a cornerstone of modern Artificial Intelligence (AI), offering unparalleled performance across a variety of complex tasks. Its ability to automatically learn features from vast amounts of data has transformed industries ranging from computer vision to natural language processing. Among the numerous applications of deep learning, three areas have witnessed significant progress: Neural Architecture Search (NAS), steganography, and medical applications. This chapter aims to explore these three areas, highlighting the significant work accomplished, the impact made, the achievements realized, and the future potential of deep learning in these domains.

Neural Architecture Search (NAS) has significantly contributed to the design of deep learning models by automating the process of architecture creation. Traditionally, designing deep learning networks required expert knowledge and trial-and-error methods, which were both time-consuming and limited by human intuition. NAS overcomes these limitations by employing algorithms to explore a vast space of potential architectures, optimizing model design for specific tasks. This process has led to the discovery of highly efficient architectures that outperform manually designed counterparts, driving innovation in fields like image recognition, natural language processing, and beyond.

The field of steganography has also seen significant advancements through deep learning. Steganography, the practice of hiding information within digital media, has long been used for secure communication. Traditional methods of steganography were often susceptible to detection or offered limited capacity for hidden data. However, with the advent of deep learning, new techniques have emerged that significantly improve the robustness, capacity, and security of hidden information, making it nearly imperceptible to human and machine detection. This advancement has broad implications for data privacy and secure communication in the digital age.

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_1

Lastly, the medical applications of deep learning represent perhaps the most profound impact of this technology. The ability to analyze vast amounts of medical data has opened new frontiers in healthcare, encompassing disease diagnosis, personalized medicine, and drug discovery. Deep learning models are now integral to interpreting medical images, predicting patient outcomes, and identifying potential therapeutic targets, significantly enhancing both the accuracy and efficiency of healthcare services. As the volume of medical data continues to grow, the role of deep learning in medicine is set to expand, offering innovative possibilities for treatment and care.

In conclusion, this chapter highlights the transformative potential of deep learning across NAS, steganography, and medical applications. By highlighting the significant advancements, impacts, and achievements within these domains, we gain valuable insights into how deep learning is reshaping various fields and paving the way for future innovations.

1.2 Neural Architecture Search (NAS)

Neural Architecture Search (NAS) represents a major leap in deep learning, automating the design of neural networks by exploring vast design spaces. Traditional neural network design requires significant human expertise to manually configure layers, connections, and other architectural components. However, NAS eliminates much of this manual work by automatically discovering optimal architectures. This process uses advanced algorithms to explore a wide range of potential configurations, aiming to identify structures that outperform human-designed models. NAS can revolutionize how deep learning models are created by making architecture design more efficient and uncovering novel architectures that might not be intuitive to human designers.

NAS works by employing two core processes: Architecture Selection and Performance Evaluation, as shown in Fig. 1.1. In the architecture selection phase, NAS explores different architectural configurations using strategies that guide it through the vast space of possible designs. Once a set of potential architectures is selected, they are rigorously evaluated through training and validation on specific datasets. Performance metrics such as accuracy, efficiency, and generalization are assessed to identify the architectures that perform best.

1.2.1 Evolution of NAS

The field of Neural Architecture Search (NAS) has undergone significant evolution, driven primarily by advancements in search strategies designed to navigate the vast space of potential architectures more efficiently. Early NAS approaches, such as Grid Search and Random Search, provided foundational methodologies but lacked the efficiency required for large-scale applications. Over time, these basic

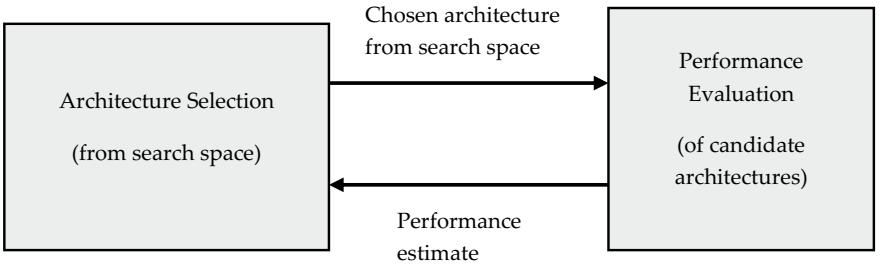


Fig. 1.1 Core processes in NAS

methods were replaced by more optimized strategies like Evolutionary Algorithm (EA), Reinforcement Learning (RL), and Gradient-based methods, each of which has contributed to improving the speed, accuracy, and computational efficiency of NAS.

Grid Search, one of the earliest NAS techniques, systematically explores every possible combination of network hyperparameters across a pre-defined range. While this brute-force approach guarantees comprehensive coverage, it is computationally prohibitive, especially in high-dimensional search spaces, limiting its practical utility in modern NAS applications. In contrast, Random Search offers a more practical alternative by randomly sampling the search space to identify optimal architectures. Despite its simplicity, Random Search has proven effective in certain cases, particularly when only a subset of parameters significantly impacts model performance. However, its unsystematic nature often means it cannot guarantee the best results.

As NAS matured, Evolutionary Algorithms emerged as a powerful search strategy inspired by natural selection. Evolutionary NAS involves evolving candidate architectures over generations using operations such as mutation, crossover, and selection. This method excels at balancing exploration and exploitation, allowing architectures to adapt and improve progressively. Evolutionary NAS is particularly advantageous when exploring large and complex search spaces, as it refines architectures over time without the need for exhaustive searching, making it a robust solution in diverse applications.

Reinforcement Learning (RL) further revolutionized NAS by framing the search process as a decision-making problem. In RL-based NAS, an agent iteratively makes architectural decisions, guided by a reward function (such as validation accuracy). Over time, the agent learns to optimize these decisions, producing high-performance models. RL has been particularly successful in discovering architectures that excel in tasks such as image classification and language modeling, offering a flexible and powerful approach to neural architecture design.

More recent advancements in NAS have focused on Gradient-based methods, such as Differentiable Architecture Search (DARTS). These methods re-define the search process by continuously relaxing the search space, allowing architecture parameters to be optimized using Gradient Descent alongside network weights. This drastically reduces the computational resources required, making Gradient-based NAS one of

the most efficient techniques. Its scalability to large datasets and complex architectures offers a compelling balance between search efficiency and performance, positioning it as one of the leading methods in modern NAS.

In brief, the progression of NAS search strategies—from the early methods of Grid and Random Search to the more sophisticated approaches of Evolutionary Algorithms, Reinforcement Learning, and Gradient-based NAS—has dramatically improved the process of architecture discovery. While early approaches provided important groundwork, the advanced methods employed today make NAS far more accessible, efficient, and effective across a wide range of applications.

1.2.2 Impact of NAS

The integration of Neural Architecture Search (NAS) into deep learning has significantly impacted both research and industry applications. One of the important contributions has been the optimization of architecture design. Traditionally, designing neural networks was a manual and time-consuming process that relied heavily on expert knowledge and iterative trial-and-error methods. NAS has revolutionized this process by automating the discovery of architectures, enabling the creation of models customized to specific tasks. This automation has not only streamlined the development process but also opened the door to more innovative and task-specific model designs that might have been overlooked by human experts.

Another key impact of NAS is the enhancement of model performance. NAS-generated architectures have consistently achieved State-of-the-art results across a variety of important benchmarks, including CIFAR-10, ImageNet, and COCO. These architectures often outperform handcrafted models in terms of both accuracy and efficiency, demonstrating the potential of NAS to push the boundaries of what deep learning models can achieve. This improvement in performance extends across a range of applications, from image classification to object detection and beyond, highlighting the versatility and effectiveness of NAS-driven designs.

Another transformative impact of NAS is the reduction of human intervention in the network design process. By automating the search for optimal architectures, NAS minimizes the need for expert knowledge, making it possible for even non-experts to achieve competitive results. This democratization of neural network design not only accelerates the pace of innovation but also broadens access to advanced deep learning techniques, allowing a wider range of practitioners to employ cutting-edge models in their respective fields.

1.2.3 Achievements of NAS

The achievements of Neural Architecture Search (NAS) are both extensive and continually expanding as the field progresses. One of the most notable accomplishments is NAS's ability to deliver State-of-the-art performance across various deep learning tasks. NAS-generated architectures have consistently outperformed manually designed models on standard datasets, like ImageNet. These successes demonstrate the effectiveness of NAS in discovering highly optimized architectures that push the boundaries of what neural networks can achieve in terms of accuracy and performance.

NAS has also brought about significant efficiency gains in the process of neural network design. Historically, designing effective models was a resource-intensive task, requiring considerable time and computational power. NAS has streamlined this process by incorporating techniques like weight sharing and early stopping, which drastically reduce the computational demands of searching for optimal architectures. These innovations ensure that the search process remains efficient while maintaining high levels of performance, allowing models to be developed faster and with fewer resources.

Another key achievement of NAS is the flexibility it introduces into neural network design. NAS allows for the exploration of a wide variety of architectural possibilities, including novel combinations of layers and connections that were previously unexplored. This flexibility has led to the discovery of unconventional architectures that challenge traditional design paradigms, further expanding the range of model designs available for various tasks.

Neural Architecture Search (NAS) has already found practical applications in several real-world scenarios, further highlighting its achievements. For instance, NAS has been employed by Google to develop EfficientNet, a family of models that achieve superior performance on image classification tasks. Additionally, NAS has been integrated into medical imaging, where it has helped optimize architectures for tasks like tumor detection and segmentation, leading to more accurate and faster diagnosis. These examples show how NAS advances both practical applications and theoretical innovations.

1.2.4 Future Directions

Despite the remarkable progress made in Neural Architecture Search (NAS), there are several promising avenues for future exploration. One key area is the development of improved search algorithms. Although current NAS methods have achieved impressive results, there is still a need for more efficient search strategies that can reduce computational costs without compromising performance. As NAS continues to evolve, researchers are expected to devise algorithms that streamline the search

process, making it faster and more resource-efficient, thus broadening its accessibility and practical application.

Neural Architecture Search also holds great potential for application in new domains beyond its traditional focus areas. Fields such as medical imaging, autonomous driving, and drug discovery present exciting opportunities for innovation. As NAS methodologies are adapted to these complex, high-stakes domains, the potential for groundbreaking advancements in technology and science grows. For instance, in medical imaging, NAS could help design models that accurately diagnose diseases with greater efficiency, while in autonomous driving, it could enhance the safety and reliability of vehicle perception systems.

A significant challenge that NAS continues to face is its high computational cost. While strides have been made to improve efficiency, the resource demands of architecture search remain substantial. Future research will likely focus on reducing this cost even further, possibly through the development of lighter, more efficient search techniques that require fewer computational resources. Achieving this would make NAS more accessible to researchers and developers who may not have access to extensive computing power.

Another promising direction is the incorporation of multi-objective optimization within NAS frameworks. Currently, most NAS systems primarily optimize for accuracy, but future research could explore balancing accuracy with other factors such as model size, energy consumption, and latency. This would be particularly important for deploying NAS-generated models on edge and mobile devices, where computational resources and energy efficiency are critical.

Additionally, hardware-aware NAS is poised to become a major focus in the future. As specialized hardware like Tensor Processing Units (TPUs) and Field-Programmable Gate Arrays (FPGAs) become more prevalent, NAS models will need to be co-optimized with specific hardware constraints in mind. This approach could lead to architectures that are fine-tuned for specific deployment environments, ensuring optimal performance and efficiency on various hardware platforms.

1.3 Deep Learning in Steganography

Steganography, the practice of hiding information within digital media, has been significantly transformed by deep learning techniques. Traditional methods relied on algorithms that manipulated image, audio, or video data to embed secret messages while maintaining imperceptibility. However, deep learning offers a more sophisticated approach, enabling the design of models that can automatically learn intricate features for secret message embedding and extraction. This evolution has led to improved capacity, security, and resistance to steganalysis. In this section, we explore the application of deep learning in steganography, highlighting key developments, impacts, achievements, and future directions.

1.3.1 Evolution of Deep Learning-Based Steganography

Deep learning's application to steganography has seen rapid advancements. Early work focused on using Convolutional Neural Networks (CNNs) to automate the feature extraction process. Generative Adversarial Networks (GANs) emerged as a powerful tool, where the generator creates stego images, and the discriminator acts as a steganalyzer, detecting secret information. Techniques like autoencoders have also been employed to generate cover images that are indistinguishable from the original ones while embedding secret data. Some key research milestones include the use of GANs to develop adversarial learning techniques for stego image generation, Variational Autoencoders (VAEs) for efficient cover generation with minimal distortion, and hybrid models that combine GANs and VAEs for higher security. The application of multi-stage deep learning architectures has helped balance embedding capacity and imperceptibility. Researchers have also introduced novel loss functions and training strategies to optimize the imperceptibility of stego images while ensuring robust data embedding.

1.3.2 Impact of Deep Learning-Based Steganography

The integration of deep learning into steganography has had a profound impact on both theoretical and practical applications. Key impacts include improved security, where deep learning models, particularly adversarial networks, have enhanced the robustness of stego images against modern steganalysis tools, making it more difficult to detect hidden messages. Additionally, these models have enabled higher payload capacities while maintaining visual fidelity, a significant advancement over traditional techniques. By automating feature extraction and learning optimal embedding patterns, deep learning reduces the need for manual intervention, which was prevalent in earlier approaches.

1.3.3 Achievements of Deep Learning-Based Steganography

Deep learning-based steganography has made significant strides in recent years, achieving remarkable success in both theoretical development and practical applications. One prominent achievement is the creation of robust models capable of generating stego images with high imperceptibility, making it difficult for steganalysis tools to detect hidden data.

Another notable achievement is the application of generative models like Generative Adversarial Networks (GANs) in steganography. The adversarial training can create more secure stego images that evade steganalysis. The use of GANs not only

enhanced the imperceptibility of stego images but also contributed to embedding larger payloads without significantly affecting image quality.

Additionally, the introduction of hybrid models, which combine autoencoders and adversarial networks, can be optimized for different steganographic tasks. This approach enhances the ability to balance imperceptibility and payload capacity while maintaining robustness against modern steganalysis tools.

These achievements showcase how deep learning-based steganography offers new possibilities for secure communication and information embedding.

1.3.4 Future Directions

The future of deep learning in steganography is promising, with several areas ripe for exploration. As deep learning architectures become more efficient, the focus will likely shift towards creating stego images with higher resolutions (e.g., 512 $\times$ 512 or larger) without sacrificing imperceptibility. Another exciting direction is the development of real-time steganography systems, allowing for the instantaneous embedding and extraction of messages in live video streams or interactive applications. While defensive models are improving, the future will also see advancements in deep learning-driven steganalysis, pushing researchers to develop even more robust and secure steganographic methods. Additionally, combining different types of media (e.g., image, audio, and text) in a single steganographic framework using deep learning could lead to richer and more complex hidden communication systems.

1.4 Deep Learning in Medical Applications

Deep learning has emerged as a transformative force in the medical field, offering innovative solutions to some of the most pressing challenges in healthcare. By incorporating the power of advanced algorithms and vast datasets, deep learning techniques have demonstrated remarkable capabilities in areas such as medical imaging, analysis of complex biological data, and treatment optimization. These technologies enable healthcare professionals to analyze complex medical data with unprecedented accuracy, enhancing diagnostic processes and personalizing patient care. From improving the detection of diseases in medical images to accelerating drug discovery and enabling real-time health monitoring, deep learning is reshaping the landscape of modern medicine. This section highlights the work done in this area, the impacts made, notable achievements, and future directions for deep learning applications in healthcare.

1.4.1 Evolution of Deep Learning in Medical Applications

Deep learning has made significant contributions to medical imaging, transforming how diseases are diagnosed and treated. In the field of image classification, Convolutional Neural Networks (CNNs) have proven especially effective. These models have been widely used to identify various types of cancers in histopathology images, detect diabetic retinopathy in retinal scans, and classify skin lesions in dermatology. The ability of CNNs to automatically learn and extract features from medical images has drastically improved the accuracy and speed of disease detection, leading to more effective diagnostic tools.

Object detection and segmentation are other critical areas where deep learning has been successfully applied. For example, deep learning models have been used to locate and segment tumors in brain MRI scans, which is essential for diagnosis and treatment planning. Similarly, these models have been employed to identify and outline organs in CT scans, facilitating more precise treatment planning. In ophthalmology, deep learning has been instrumental in detecting and segmenting blood vessels in retinal images, improving the diagnosis of vascular diseases and conditions like diabetic retinopathy.

Beyond classification and segmentation, deep learning techniques have also contributed to image reconstruction and enhancement. For instance, deep learning algorithms are used to denoise low-dose CT scans, making them safer for patients by reducing radiation exposure while maintaining image quality. Super-resolution techniques have been applied to enhance MRI images, providing clearer visualizations for better diagnosis. Additionally, deep learning has played a pivotal role in reconstructing PET images from limited data, improving the quality of imaging in situations where acquiring full datasets is challenging.

Image registration is another area where deep learning has shown great promise. Aligning medical images from different modalities or time points is crucial for accurate diagnosis and treatment. Deep learning models have been developed to align preoperative and intra-operative images, providing valuable guidance during surgeries. Similarly, these models have been used to register PET and CT images, improving the accuracy of multi-modality diagnostics and enhancing the overall quality of medical evaluations.

Finally, deep learning has been used to model and predict disease progression. Researchers have developed models that predict the progression of Alzheimer's disease using longitudinal MRI scans, offering insights into how the disease evolves over time. In oncology, deep learning models have been employed to simulate tumor growth and predict responses to treatment, helping doctors to tailor therapies more effectively. These advancements in disease progression modeling provide clinicians with valuable tools for proactive patient care and personalized treatment strategies. Furthermore, deep learning has revolutionized the analysis of biological sequence data, significantly advancing Protein Secondary Structure Prediction (PSSP) and pan-cancer classification. By utilizing models like Recurrent Neural Networks (RNNs), researchers have significantly enhanced the accuracy of these predictions, aiding

in understanding protein folding, stability, and interactions with other molecules. These advancements are crucial for drug discovery and the development of protein-based therapies, providing a deeper insight into the biological mechanisms underlying diseases. Similarly, deep learning models have excelled in analyzing vast genomic datasets, such as those from The Cancer Genome Atlas (TCGA), to identify shared molecular signatures across different cancer types. These models have significantly improved the precision of cancer diagnosis and enabled the development of more personalized treatment strategies, moving beyond traditional organ-based classification and advancing the field of precision oncology.

1.4.2 Impact of Deep Learning in Medical Applications

The integration of deep learning methodologies into medical imaging and the analysis of proteomic and genomic data has profoundly impacted healthcare, bringing significant improvements in various aspects of the field. One of the most notable advancements has been the improvement in diagnostic accuracy. Deep learning algorithms, particularly in areas like image classification and object detection, have demonstrated performance that rivals, and in some cases surpasses, that of human practitioners. This has led to earlier identification of medical conditions, a reduction in both false positives and false negatives, and more reliable and objective evaluations. These advancements enhance the quality of patient care by enabling more precise and timely diagnoses.

Another key impact of deep learning in medical imaging is the increased efficiency in diagnostic processes. Automation of traditionally labor-intensive tasks, such as image segmentation and analysis, has dramatically accelerated the pace of medical image interpretation. This has resulted in faster reporting times, allowing for quicker clinical decisions. Additionally, the reduced workload on radiologists enables them to focus on more complex cases, improving the overall operational flow within radiology departments and healthcare facilities.

Deep learning holds immense potential in processing proteomic and genomic data. In proteomics, it enables accurate prediction of protein structures, unraveling complex biological functions and accelerating drug discovery. In genomics, it empowers the classification of cancers by identifying shared molecular patterns across diverse cancer types, paving the way for personalized medicine and targeted therapies.

Deep learning also has the potential to enhance accessibility to high-quality diagnostics, particularly in underserved regions. In areas with limited access to specialized radiologists, deep learning models can facilitate large-scale screening programs, ensuring that individuals in remote or resource-limited environments receive timely diagnoses. Furthermore, these models can provide second opinions, increasing the availability of expert-level diagnostic evaluations without the need for physical consultations, thereby bridging the healthcare gap between urban and rural settings.

Personalized medicine is another area where deep learning is making a significant impact. By extracting detailed insights from medical imaging data, deep learning helps generate more precise prognoses and formulate individualized treatment strategies based on specific patient profiles. The ability to track therapeutic responses more effectively allows for adjustments in treatment plans in real time, leading to better outcomes. This personalized approach to healthcare is transforming how diseases are managed, offering treatments that are tailored to the unique needs of each patient.

Finally, deep learning has accelerated the pace of medical research by enabling faster and more comprehensive analysis of large datasets. The ability of these algorithms to identify novel biomarkers and imaging features associated with various diseases has opened new avenues for research and discovery. Moreover, deep learning is promoting the development of innovative imaging techniques and procedural standards, contributing to advancements in both diagnostic and therapeutic approaches. These research breakthroughs are shaping the future of medical imaging, pushing the boundaries of what is possible in healthcare.

1.4.3 Achievements of Deep Learning in Medical Applications

Several notable achievements underscore the transformative potential of deep learning in medical imaging. One such advancement is in breast cancer detection, where Google Health developed an AI system that has outperformed radiologists in breast cancer screening. This system has significantly reduced both false positives and false negatives, demonstrating a higher level of accuracy than human practitioners. Its ability to provide more reliable screening results marks a substantial step forward in early cancer detection and patient outcomes.

Another key achievement is in the screening for diabetic retinopathy, a condition that affects the eyes and can lead to blindness if left untreated. DeepMind, in collaboration with Google Health, developed a deep learning system capable of diagnosing diabetic retinopathy with the same level of accuracy as expert ophthalmologists. This system offers the potential for widespread, accessible screening, particularly in regions with limited access to specialized healthcare professionals.

In the realm of lung cancer detection, researchers at Northwestern University developed a Convolutional Neural Network (CNN) that surpassed radiologists in identifying lung cancer on low-dose chest CT scans. This deep learning model not only demonstrated superior accuracy but also had the potential to expedite early diagnosis, offering patients a better chance of effective treatment. Its success in detecting small, early-stage cancers that may be missed by human radiologists highlights the critical role of AI in improving cancer diagnostics.

The field of brain tumor segmentation has also seen remarkable progress, particularly through the Brain Tumor Segmentation (BraTS) challenge. Deep learning models participating in this challenge have continually improved in their ability to

segment brain tumors from MRI scans. These models are now approaching human-level performance, offering a more precise and consistent method of tumor detection, which is essential for treatment planning and prognosis in neuro-oncology.

During the COVID-19 pandemic, deep learning played a crucial role in the rapid development of models for detecting the virus from chest X-rays and CT scans. These models were instrumental in aiding healthcare professionals with quick triage and diagnosis, especially in overwhelmed healthcare systems. The ability of deep learning models to analyze imaging data quickly and accurately was vital in managing the pandemic, providing valuable insights into the progression of the disease and its impact on the lungs. In addition, Alphafold, developed by DeepMind, demonstrated near-experimental accuracy in predicting protein structures and played a crucial role in understanding the structural biology of SARS-CoV-2, aiding in the fight against COVID-19 by accelerating the discovery of viral protein interactions and potential therapeutic targets. These achievements collectively highlight the growing importance of deep learning in enhancing the accuracy, efficiency, and accessibility of medical diagnostics.

1.4.4 Future Directions

The future potential of deep learning in medical imaging is vast and holds promise across several key areas. One exciting direction is the integration of multi-modal data, combining imaging with genetic, clinical, and laboratory information to offer a holistic view of a patient's health. This could lead to earlier diagnoses, improved prognostic assessments, and more personalized treatment plans. Additionally, advancements in computational power, particularly through cloud computing and edge processing, could enable real-time image analysis, providing immediate feedback to clinicians at the point of care and speeding up the diagnostic process. Explainable AI (XAI) is another area of intense research, aimed at making deep learning models more transparent and trustworthy by using tools like heat maps to show how models arrive at specific diagnoses. In terms of disease detection, deep learning has the potential to identify conditions in their earliest stages, even before symptoms appear, such as in cancer screening using low-dose CT scans or biomarkers. Looking further ahead, autonomous diagnostic systems could operate independently of human oversight, particularly in resource-constrained environments or during emergencies. The integration of deep learning into personalized medicine also holds promise, as AI could analyze imaging alongside genomics and patient history to predict individual responses to treatments.

Finally, combining deep learning with robotic systems for image-guided surgeries could revolutionize surgical precision and safety, enabling more accurate intra-operative decision-making and improving patient outcomes.

In conclusion, deep learning is reshaping the medical field by offering more accurate, scalable, and personalized approaches to diagnostics and treatment. By

improving tasks such as protein structure prediction and gene expression classification, it is paving the way for more effective medical research and patient care.

1.5 Summary

This chapter presents an overview of deep learning's impact on three significant domains: Neural Architecture Search (NAS), steganography, and medical applications. It underscores the transformative potential of deep learning while addressing the advancements made, the challenges encountered, and the future possibilities in these fields.

The chapter begins by defining NAS, an automated approach that streamlines neural network design, moving away from manual configurations to a method that utilizes algorithms to explore a vast space of potential architectures. This automation has led to the discovery of efficient and high-performance models that outperform traditional designs, significantly advancing areas such as image recognition and natural language processing.

In the domain of steganography, the chapter highlights how deep learning has revolutionized secure data embedding techniques within digital media. Traditional methods faced limitations in imperceptibility and capacity; however, advancements like Generative Adversarial Networks (GANs) and autoencoders have enhanced robustness, security, and capacity, making hidden information increasingly difficult to detect.

The chapter also delves into the medical applications of deep learning, where it plays a crucial role in disease diagnosis, treatment optimization, and drug discovery. By analyzing vast datasets, deep learning models improve the accuracy of medical imaging interpretations and aid in personalized medicine initiatives.

Further, the chapter reviews the evolution of NAS, highlighting early methods like Grid and Random Search and their progression to more sophisticated strategies, including Evolutionary Algorithms, Reinforcement Learning, and Gradient-based approaches. It emphasizes the significant contributions of NAS, such as optimization of architecture design, enhanced performance metrics, and reduction of human intervention.

Looking towards the future, the chapter outlines several research avenues, including improving search algorithms, applying NAS to new domains, and developing techniques for multi-objective optimization and hardware-aware designs.

In summary, this chapter highlights the impact of deep learning in NAS, steganography, and medical applications and paves the way for understanding the evolving role of deep learning in shaping the future of technology and healthcare.

Bibliography

1. H. Liu, K. Simonyan, Y. Yang, DARTS: differentiable architecture search, in *7th International Conference on Learning Representations, ICLR 2019* (2019), pp. 1–13
2. P. Ren et al., A comprehensive survey of neural architecture search: challenges and solutions. ACM Comput. Surv. **54**(4), (2021). https://doi.org/10.1145/3447582
3. T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: a survey. J. Mach. Learn. Res. **20**, 1–21 (2019)
4. P. Liashchynskyi, P. Liashchynskyi, Grid search, random search, genetic algorithm: a big comparison for NAS, vol. 2017 (2019), pp. 1–11. http://arxiv.org/abs/1912.06059. (Online)
5. Y. Liu, Y. Sun, B. Xue, M. Zhang, G.G. Yen, K.C. Tan, A survey on evolutionary neural architecture search. IEEE Trans. Neural Netw. Learn. Syst. **34**(2), 550–570 (2023). https://doi.org/10.1109/TNNLS.2021.3100554
6. M. Wistuba, A. Rawat, T. Pedapati, A survey on neural architecture search (2019). http://arxiv.org/abs/1905.01392. (Online)
7. S. Santra, J.W. Hsieh, C.F. Lin, Gradient descent effects on differential neural architecture search: a survey. IEEE Access **9**, 89602–89618 (2021). https://doi.org/10.1109/ACCESS.2021.3090918
8. M.A. Wani, B. Sultan, Deep learning based image steganography: a review. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. (July), 1–26 (2022). https://doi.org/10.1002/widm.1481.
9. R. Zhang, S. Dong, J. Liu, Invisible steganography via generative adversarial networks. Multimed. Tools Appl. **78**(7), 8559–8575 (2019). https://doi.org/10.1007/s11042-018-6951-z
10. J. Tan, X. Liao, J. Liu, Y. Cao, H. Jiang, Channel attention image steganography with generative adversarial networks. IEEE Trans. Netw. Sci. Eng. **9**(2), 888–903 (2022). https://doi.org/10.1109/TNSE.2021.3139671
11. D. Hu, L. Wang, W. Jiang, S. Zheng, B. Li, A novel image steganography method via deep convolutional generative adversarial networks. IEEE Access **6**(c), 38303–38314 (2018). https://doi.org/10.1109/ACCESS.2018.2852771
12. N.M. Din et al., Breast cancer detection using deep learning: datasets, methods, and challenges ahead. Comput. Biol. Med. **149**, 106073 (2022). https://doi.org/10.1016/j.compbiomed.2022.106073
13. N.M. Dipu, S.A. Shohan, K.M. Salam, Deep learning based brain tumor detection and classification, in *2021 International Conference on Intelligent Technologies (CONIT)* (2021). https://doi.org/10.1109/conit51480.2021.9498384. (Preprint)
14. A.M. Ismael, A. Şengür, Deep learning approaches for COVID-19 detection based on chest X-ray images. Expert Syst. Appl. **164**, 114054 (2021). https://doi.org/10.1016/j.eswa.2020.114054
15. W. Saeed, C. Omlin, Explainable AI (XAI): a systematic meta-survey of current challenges and future opportunities. Knowl.-Based Syst. **263**, 110273 (2023). https://doi.org/10.1016/j.knosys.2023.110273
16. S. Das, M. Mishra, S. Majumder, Detection of diabetic retinopathy from retinal fundus images by using CNN model resnet-50. Deep. Learn. Biomed. Signal Med. Imaging 1–13 (2024). https://doi.org/10.1201/9781032635149-1
17. S. Ali, M.A. Wani, Recent trends in neural architecture search systems, in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, Nassau, Bahamas (2022), pp. 1783–1790. https://doi.org/10.1109/ICMLA55696.2022.00272
18. M. Tan, Q.V. Le, EfficientNet: rethinking model scaling for convolutional neural networks. http://arxiv.org/abs/1905.11946v5 (2019). (n. pag.)
19. S. Ali, M.A. Wani, Gradient-based neural architecture search: a comprehensive evaluation. Mach. Learn. Knowl. Extr. **5**, 1176–1194 (2023). https://doi.org/10.3390/make5030060

Chapter 2
Evolutionary Algorithm-Based Neural Architecture Search

2.1 Introduction

The increasing demand for specialized neural network architectures that cater to specific tasks has given rise to automated methods for architecture design, alleviating the need for manual, labor-intensive processes. Neural Architecture Search (NAS) has emerged as a key solution, enabling the discovery of optimized neural networks without human intervention. Among the various approaches to NAS, Evolutionary Algorithm-based NAS has proven to be particularly effective due to its ability to efficiently navigate the vast search space of neural architectures by employing biologically inspired optimization techniques.

Evolutionary Algorithms (EAs) are rooted in the principles of Darwinian natural selection. By evolving a population of candidate solutions—neural architectures—over multiple generations, EAs apply genetic operations such as mutation, crossover, and selection to gradually improve architecture performance. The strength of Evolutionary NAS lies in its ability to balance exploration and exploitation. Through mutation, the search can explore new areas of the architecture space, while crossover combines the strengths of high-performing architectures to refine solutions.

It is worth noting that the earliest NAS approaches were closely tied to genetic algorithms, which inspired many of the core concepts in Evolutionary NAS. These early Genetic Algorithm-based NAS methods demonstrated the feasibility of optimizing neural architectures through Evolutionary processes. However, before the rise of Evolutionary NAS, simpler techniques such as Grid Search and Random Search were commonly employed. In Grid Search, a predefined set of hyperparameters is exhaustively evaluated, but this method suffers from inefficiency due to its computational cost and lack of adaptability to dynamic search spaces. Random search, on the other hand, involves selecting hyperparameters at random and evaluating them. While more efficient than Grid Search in certain cases, Random Search lacks the structured guidance provided by Evolutionary or other NAS methods.

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_2

Evolutionary NAS represents a significant improvement over these earlier techniques by offering a flexible search process. Unlike Grid and Random search, Evolutionary NAS can adaptively refine architectures, using the best candidates from previous generations while also exploring new possibilities. This adaptability makes it a powerful tool for optimizing complex neural networks for a wide range of applications.

This chapter will explore the foundational principles of Evolutionary Algorithms, including key components like population dynamics, fitness evaluation, and genetic operators. Furthermore, it will detail how these principles are applied within NAS frameworks to evolve neural architectures. By the end of this chapter, readers will have a thorough understanding of how Evolutionary Algorithms power NAS, and their role in the future of neural network design.

2.2 Evolutionary Algorithm-Based NAS Methods

In Neural Architecture Search (NAS), Evolutionary Algorithms are frequently employed to navigate the vast search space of potential neural network architectures. The process of Evolutionary NAS begins with the definition of a search space, which encompasses the fundamental building blocks required to construct a neural network. For instance, in the case of a Convolutional Neural Network (CNN), the search space might include essential components such as convolutional layers, pooling layers, dense layers, and activation functions. These components can be combined in numerous ways to create different candidate architectures.

The Evolutionary Algorithm operates by treating each candidate architecture as an individual in a population. Initially, a population of simple neural networks is randomly generated, each representing a potential solution within the search space. From this population, the models that perform best—based on an evaluation metric such as accuracy—are selected as parents. These parent models undergo genetic operations, such as crossover (where parts of two architectures are swapped) and mutation (where a small change is introduced to an architecture), producing new architectures that are potentially more refined.

The newly generated architectures are then evaluated for performance. Over multiple generations, this process of selecting the best models, applying crossover and mutation, and evaluating performance is repeated. Through this Evolutionary process, the architecture becomes increasingly optimized, improving both its performance and efficiency compared to the initial models.

Below, we discuss the foundational concepts of Evolutionary NAS, exploring how key components—such as search space and genetic operations—work together to drive the optimization process.

2.2.1 Search Space

The search space in Evolutionary NAS forms the fundamental landscape that defines all potential neural network configurations available for exploration and optimization. It encompasses two key aspects: the operation space, which outlines the possible architecture component choices and hyperparameters, and the representation of these architecture components, which provides a structured way to encode and manipulate network designs. Together, these components enable the Evolutionary Algorithm to systematically explore and refine neural network architectures.

Two key types of search spaces that can be explored using Evolutionary Algorithms are the **chain-structured search space** and the **cell-based search space**. Each of these approaches offers unique ways to define and optimize the structure of neural networks.

A. **Chain-Structured Search Space**

In a chain-structured search space, the architecture is represented as a simple sequence of layers, typically organized in a feed-forward manner. The search process aims to evolve this sequence by altering the types and configurations of layers (e.g., convolutional, pooling, and dense layers), the number of filters or units, activation functions, and other hyperparameters. Evolutionary operations such as mutation and crossover are applied to modify the architecture's structure, thereby exploring different configurations and identifying the optimal chain of layers that yields the best performance.

For example, in a chain-structured CNN architecture:

- The input layer is followed by a series of convolutional layers.
- The final convolutional layer output is flattened and passed to dense layers for classification.
- The Evolutionary process searches for the best combination of convolutional filters, dense units, and activations.

This approach is intuitive and easy to implement, but it may lack the flexibility needed to design more complex architectures with repeated patterns or complex data flow.

An example of chain structured search space, in which a neural network configuration is encoded in a structured format, such as a Python dictionary, is shown in Table 2.1. This representation captures all essential hyperparameters, including the number of layers and their respective sizes, enabling a clear and consistent approach to defining architectures.

In this example, the network has two convolutional layers and one dense layer. The first convolutional layer uses 32 filters of size 3×3, while the second convolutional layer uses 64 filters of size 5×5. The dense layer has 128 units. This structured representation simplifies the implementation of NAS algorithms, allowing for efficient exploration and optimization of different network configurations. Figure 2.1 depicts a simplified flow of various steps in the Evolutionary Algorithm-based NAS,

Table 2.1 Sample architecture representation for chain-structured search space

Key	No. of conv_ layers	No. of flatten layers	No. of dense layers	Conv.#1 filter_ count	Conv.#2 filter_ count	Conv.#1 filter_ size	Conv.#2 filter_ size	Dense #1 unit	Activation function
Value	2	1	1	32	64	3	5	128	Relu

utilizing chain-structured search space. The various components are discussed in detail in later sections.

B. **Cell-Based Search Space**

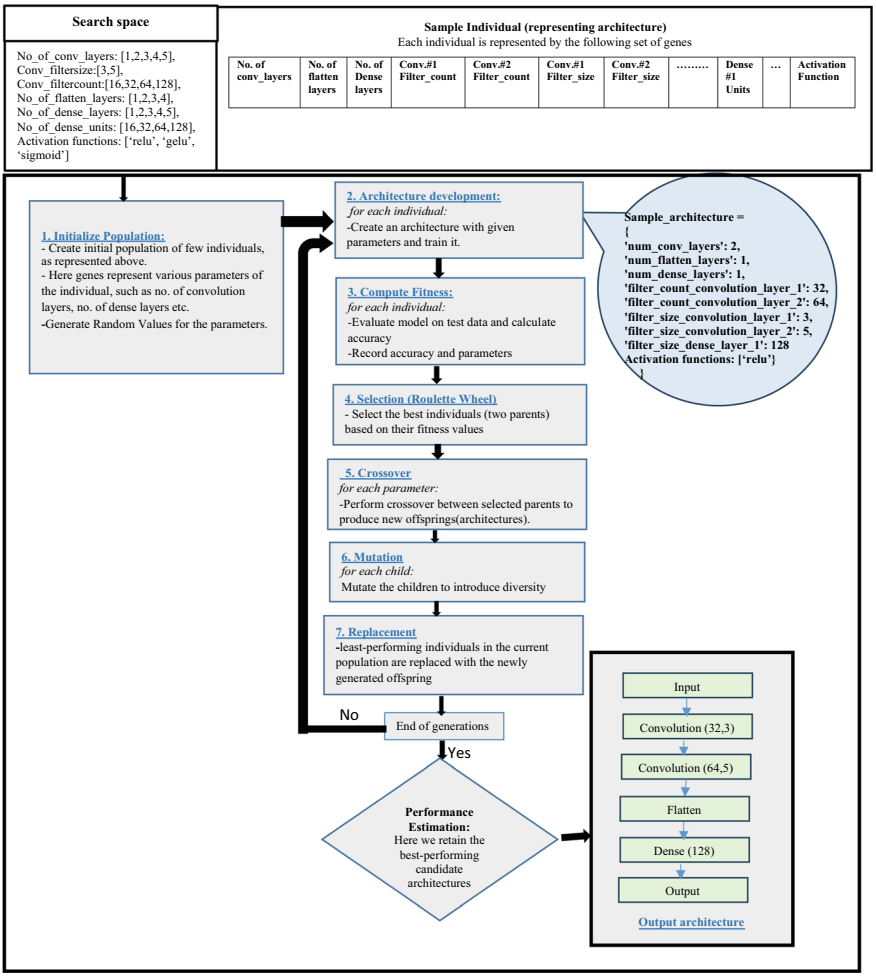


Fig. 2.1 Flowchart of simple chain-structured Evolutionary Algorithm-based Neural Architecture Search (EA-NAS)

The cell-based search space in Neural Architecture Search (NAS) offers a modular and flexible approach to designing deep learning models. In this framework, a neural network is built from multiple cells that are repeated throughout the architecture. Each cell functions as an independent unit, performing a specific set of operations such as convolutions, pooling, or skip connections. By focusing the search on optimizing the configuration of these cells, NAS can efficiently explore the design space and construct complex models with repeated, well-optimized components.

In an Evolutionary NAS, cells evolve through genetic operations like mutation and crossover, which modify the structure of the cell by altering paths between nodes or changing the operations along those paths. This process allows the Evolutionary Algorithm to optimize the cell structure incrementally. Once an optimized cell is discovered, it can be reused across various layers in the network, leading to sophisticated architectures that benefit from repeated patterns of operations. This approach significantly reduces the complexity of searching for optimal architectures, as it concentrates on refining smaller, reusable components.

To efficiently represent the structure of a cell, path-based one-hot encoding is often employed. For example, consider a NAS framework operating on a simple cell with four nodes, as shown in Fig. 2.2. Each node represents a feature map, while the edges connecting the nodes represent operations like convolutions or pooling. The goal is to represent the cell's structure using one-hot encoding, which captures the operation along each edge as a binary feature.

In this example, there are two types of operations: Operation 1 (denoted by green edges, e.g., a 3×3 convolution) and Operation 2 (denoted by yellow edges, e.g., a 5×5 convolution). One-hot encoding is used to represent the operation on each edge as a binary vector. For every edge between two nodes, a binary pair is assigned to indicate which operation is present. If Operation 1 is active on a given edge, the encoding would be [1, 0], meaning that Operation 1 is present, and Operation 2 is absent. Conversely, if Operation 2 is selected, the encoding would be [0, 1].

This encoding technique provides a compact and efficient way to represent the cell's structure, facilitating the manipulation and optimization of the architecture during the Evolutionary process. By encoding each operation as a binary feature, the algorithm can efficiently explore various combinations of operations and paths within the cell, ultimately leading to an optimized network architecture.

Figure 2.2 illustrates the flow of steps in an Evolutionary Algorithm-based NAS, utilizing the cell-based search space. The key components involved in this process—such as the search space, genetic operations, and evaluation strategies—are discussed in subsequent sections, providing a deeper understanding of how Evolutionary NAS optimizes neural architectures.

This approach exemplifies how Evolutionary methods, inspired by natural selection, can be applied to optimize neural network architectures, yielding efficient and high-performing models by evolving simple, modular cell structures.

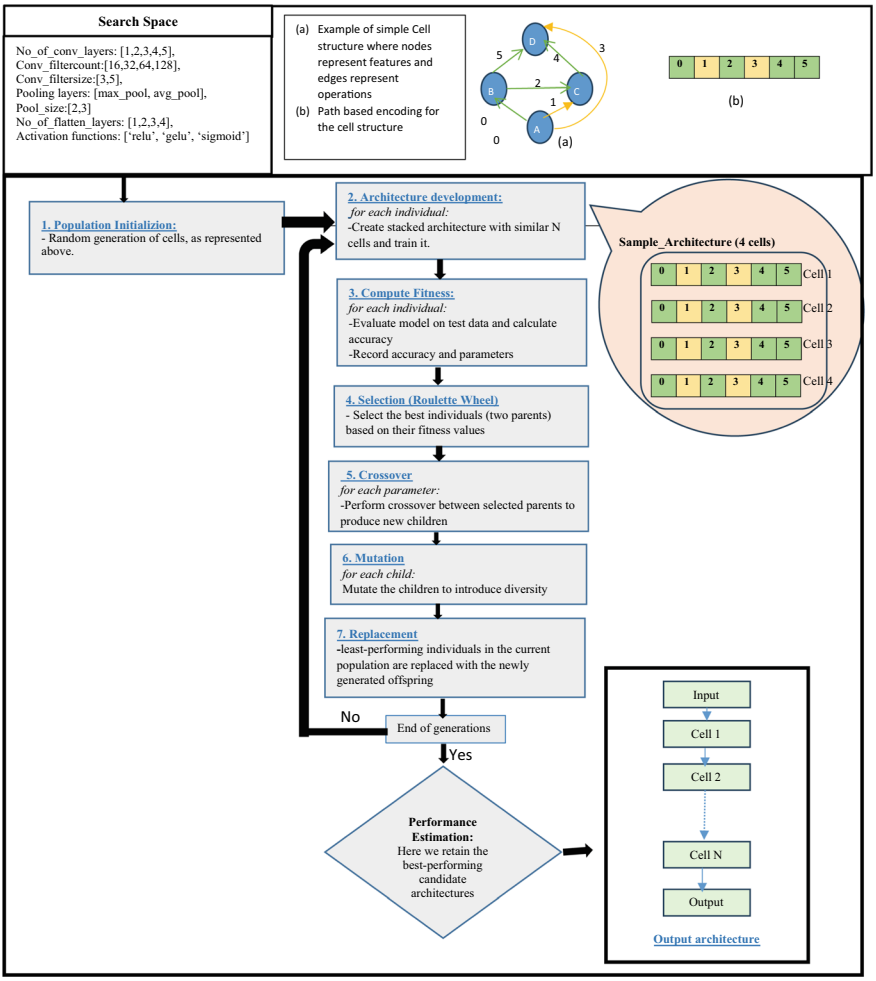


Fig. 2.2 Flowchart of simplified cell-based EA-NAS

2.2.2 Search Strategy

The search strategy in Evolutionary Neural Architecture Search (NAS) involves a series of methodical steps designed to explore and optimize neural network architectures efficiently. The Evolutionary process relies on principles of natural selection, using a population of candidate architectures that are progressively refined through iterative cycles of evaluation, selection, and genetic operations. These key steps, which form the backbone of the search strategy, are outlined below:

A. Population Initialization

In the Evolutionary Neural Architecture Search (NAS) framework, the population initialization phase marks the beginning of the search process. During this phase, an initial population of candidate architectures—each representing a potential neural network design—is generated. The population can consist of chain-structured networks or cell-based architectures, depending on the search space being utilized. Each candidate, or individual, within this population, is encoded using a set of genes that define essential architectural parameters such as the number of convolutional layers, dense layers, filter sizes, and activation functions.

These genes are randomly initialized within predefined bounds, ensuring diversity within the population and allowing for the exploration of a wide variety of network configurations. The initial population size is typically set to a reasonably large number, ensuring that the search begins from a broad and diverse set of candidate solutions. This random generation process results in architectures ranging from simple to highly complex designs, providing a rich foundation for subsequent Evolutionary operations.

The purpose of this diverse initialization is to ensure that the search explores various architectural designs early in the process. Each individual's performance is evaluated based on specific criteria, such as accuracy, computational efficiency, or a task-specific objective. The highest-performing architectures are then selected as "parents" for the next phase of the algorithm, where genetic operators such as crossover and mutation are applied. This allows the search to iteratively refine the architecture design, progressively leading to higher-performing neural networks. The random initialization of the population ensures that the search covers a broad spectrum of possibilities, increasing the likelihood of discovering optimal or near-optimal solutions as the Evolutionary process unfolds.

B. Architecture Development

In the architecture development phase of Evolutionary Neural Architecture Search (NAS), each individual within the population is transformed into a unique neural network architecture based on its encoded genetic parameters. These parameters, which define the network's structure and components—such as the number of convolutional layers, dense layers, filter sizes, and activation functions—act as the building blocks for each architecture. By decoding these genetic encodings, each individual is instantiated as a distinct neural network configuration, ensuring architectural diversity within the population. A simple example of architecture development in chain structure-based EA-NAS is shown in Fig. 2.3.

In the case of a cell-based search space, the architecture is created by stacking multiple cells together, as exemplified in Fig. 2.4. These cells function as modular building blocks, each defined by a set of operations—such as convolutions, pooling, or skip connections—and the specific connections between nodes within the cell. The configuration of each cell, including the operations performed along the edges and the connections between nodes, is determined by the genetic encoding provided by the Evolutionary Algorithm. By stacking N cells in sequence, the overall network gains both depth and complexity, allowing for more sophisticated feature extraction and data transformation.

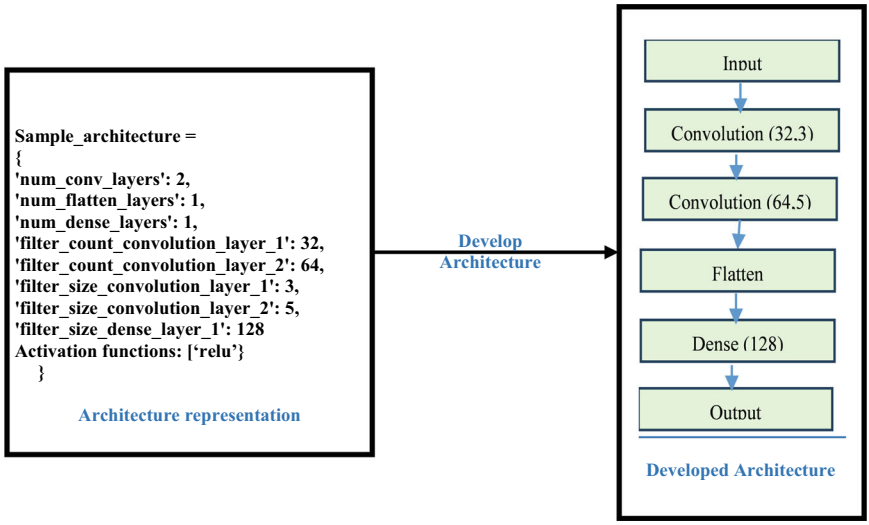


Fig. 2.3 Simple example of architecture development in chain structure-based EA-NAS

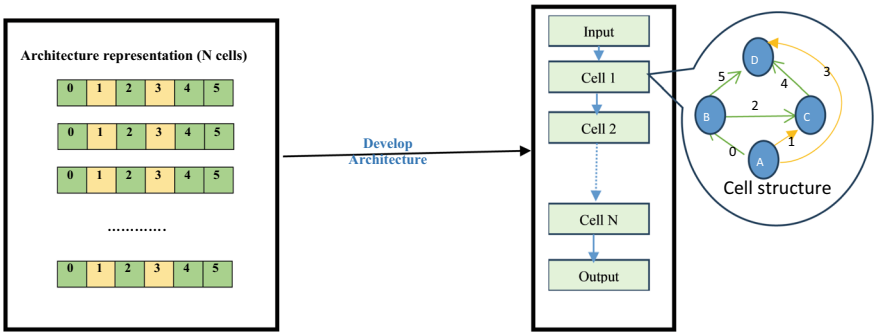


Fig. 2.4 Simple example of architecture development in cell-based EA-NAS

This cell-based approach offers a flexible framework for designing deep learning models. Once an optimized cell structure is discovered, it can be reused across the network, resulting in an efficient design process. The repetition of cells not only simplifies the architecture search but also constrains the search space, enabling a more targeted exploration of optimal network configurations. As a result, the developed architectures balance complexity and efficiency, with the potential to yield high-performing models suited to the specific task at hand.

C. Compute Fitness

Once the neural network architectures are constructed, their performance is evaluated through a fitness assessment using a designated dataset (see Fig. 2.5). The primary metric for determining the fitness of each individual architecture is accuracy, which

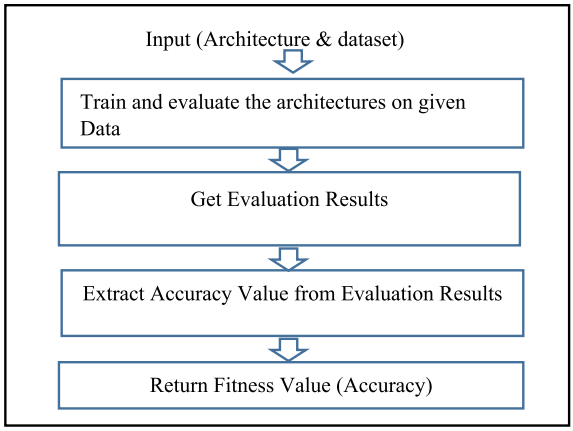


Fig. 2.5 Fitness computation in Evolutionary NAS

reflects the proportion of correct predictions made by the model on unseen data. This accuracy score provides a quantitative measure of how well the architecture generalizes to new inputs, serving as a direct indicator of its effectiveness for the given task.

During the evaluation process, each model's accuracy is calculated and recorded alongside its corresponding architectural parameters. This accuracy score acts as the fitness value in the Evolutionary Algorithm, guiding the selection of individuals for further processes. Models that achieve higher accuracy are considered fitter and are more likely to be selected for the next phase of evolution.

Following the fitness evaluation, the best-performing architectures will undergo genetic operations, such as crossover and mutation, to refine and optimize their design in subsequent generations.

D. **Parent Selection**

The next phase in the Evolutionary NAS process is selection, where individuals with the highest fitness values are chosen to become parents for the next generation. The Roulette Wheel Selection method is commonly used, which gives higher probabilities of selection to individuals with better fitness scores, while still allowing a chance for less fit individuals to be selected. This ensures diversity and helps avoid premature convergence to suboptimal solutions.

In the Roulette Wheel Selection process, the fitness value of each individual is normalized to reflect its proportion relative to the total fitness of the population. A "wheel" is imagined, where each individual occupies a segment proportional to its fitness. The algorithm then randomly "spins the wheel" to select two parents, with the likelihood of selection being proportional to the individual's fitness.

This method ensures that the best-performing individuals have a higher chance of being selected as parents while still maintaining diversity in the population. Once the two parent individuals are selected, they are used in the next stage to undergo genetic operations, such as crossover and mutation, to create offspring for the next generation.

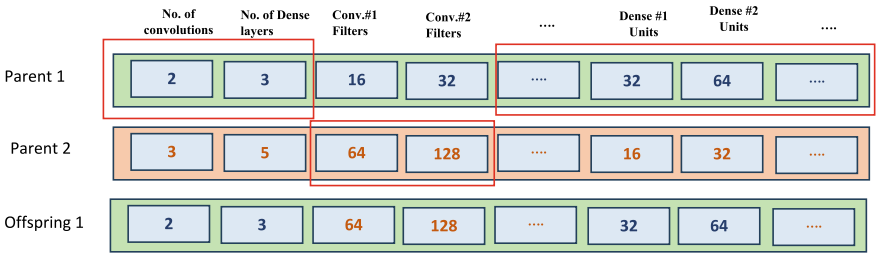


Fig. 2.6 Simple example of crossover in chain-structured architecture

E. **Genetic Operations**

Genetic operations in Evolutionary Algorithms and Neural Architecture Search (NAS) are fundamental processes that drive the exploration and optimization of neural network architectures. These operations, primarily crossover and mutation, play key roles in generating new solutions from existing ones, mimicking biological evolution.

Crossover involves combining elements of two parent architectures to create offspring (candidate architectures) with potentially improved characteristics. The process mirrors genetic recombination in biological reproduction and aims to explore novel architectural configurations. Once the parents were selected, the crossover method (shown in Fig. 2.6) is applied to generate new offspring. Each hyperparameter of the offspring is independently inherited from one of the two parents with equal probability.

In the cell-based crossover, operations between the cells of two parent architectures are interchanged to create offspring, as shown in Fig. 2.7. This involves swapping specific operations—such as convolutional layers, pooling layers, or activation functions—between corresponding cells in the parent architectures. By interchanging these operations, the offspring inherit a mix of features from both parents, enabling the exploration of novel architectural designs. The exchange of operations can occur at multiple levels within the cells, such as exchanging convolution types (e.g., depthwise vs. standard convolution) or altering kernel sizes, allowing for a diverse search of potential performance improvements. This method preserves the structural integrity of the cells while introducing variability in their functional components, enhancing the diversity of candidate architectures.

Mutation introduces further diversity by making small, random adjustments to an architecture. These changes might include varying the number of layers or altering filter sizes. The mutation probability—typically a small value such as 0.4—determines how often mutations occur. By applying mutations with a certain probability, the algorithm can explore different configurations and potentially discover novel and high-performing architectures that might not emerge through crossover alone.

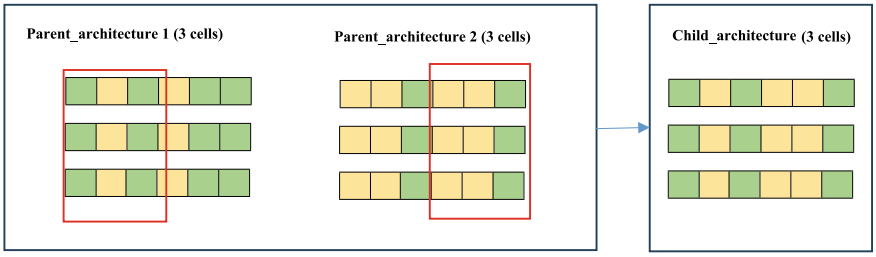


Fig. 2.7 Simple example of crossover in cell-based architecture

Specifically, in a point mutation, as illustrated in Figs. 2.8 and 2.9, a single hyperparameter of the architecture is randomly changed. In chain-structured architectures (Fig. 2.8), this might involve altering the depth or the type of layers, while in cell-based architectures (Fig. 2.9), the mutation can affect operations within individual cells, such as changing the convolution type, activation function, or other hyperparameters. After both crossover and mutation operations, the modified offspring are added to the new generation, ensuring continued exploration and refinement of the architecture space.

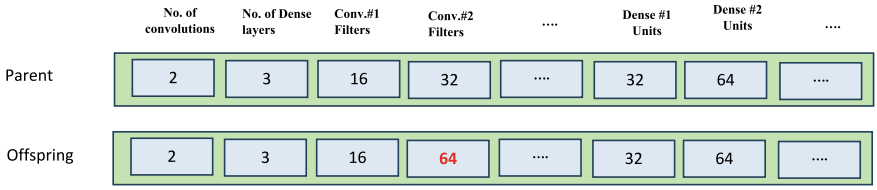


Fig. 2.8 Simple example of mutation in chain-structured architecture

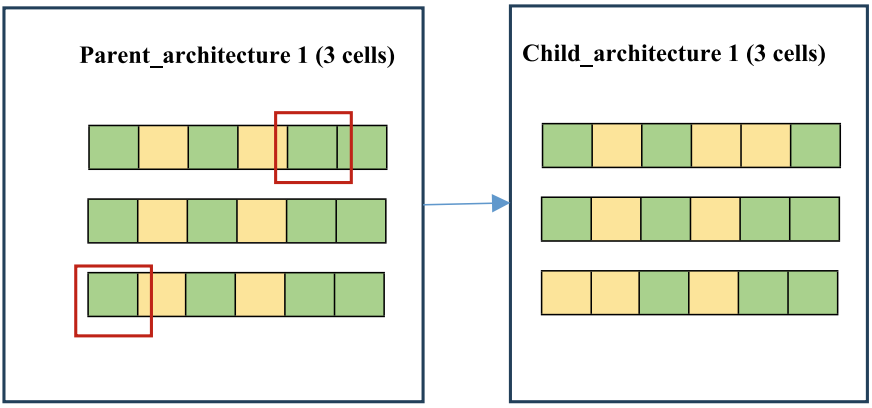


Fig. 2.9 Simple example of mutation in cell-based architecture

F. **Replacement**

After the selection and genetic operations (like crossover and mutation) generate new offspring, the next step is to replace the least-performing individuals in the current population with the newly generated offspring. The fitness of the individuals in the population is reassessed, and those with the lowest performance—typically those with the poorest accuracy or other relevant metrics—are removed. These least-performing individuals are replaced by the new offspring, ensuring that the population evolves by retaining high-performing architectures while introducing new potential solutions.

This replacement strategy ensures that the Evolutionary process maintains diversity and continually improves the population by discarding less effective models in favor of potentially stronger ones from the next generation.

G. **Termination Condition**

Steps B, C, D, E, and F of the Evolutionary NAS process continue iteratively until a predefined termination condition is met. Common termination criteria include achieving a specified accuracy threshold, reaching a maximum number of generations, or observing no significant improvement in fitness over a set number of iterations. Once the termination condition is satisfied, the algorithm concludes, and the best-performing architecture identified throughout the process is selected for deployment. This final architecture reflects the culmination of the Evolutionary search, optimized for the specific task of interest, such as image classification.

H. **Final Architecture**

The architecture with the highest fitness score tracked globally across all generations, is selected as the final output of the NAS process.

2.3 Results and Discussions

This section presents a comprehensive evaluation of the performance of various Evolutionary Neural Architecture Search (NAS) algorithms, on image classification datasets (CIFAR-10 and CIFAR-100), highlighting key advancements in neural architecture optimization.

A. ***Performance Comparison on CIFAR-10***

The performance of various Evolutionary Neural Architecture Search (NAS) algorithms on the CIFAR-10 dataset, as summarized in Table 2.2, reveals significant advancements in terms of accuracy, parameter efficiency, and computational costs. This section discusses these results, focusing on the trends in accuracy improvement, model complexity, and computational efficiency across different approaches.

Early methods such as LargeEvo (2017) achieved an error rate of 5.4%, marking a significant milestone in neural architecture search. Utilizing a chain-structured search space, LargeEvo demonstrated the effectiveness of Evolutionary Algorithms

Table 2.2 Performance comparison of Evolutionary NAS algorithms on CIFAR-10

References	Search space	Year	Error rate (%)	Parameters (Millions)	GPU days
LargeEvo [2]	Chain-structured	2017	5.4	5.4	2750
HierarchicalEvolution [5]	Cell-based	2017	3.63	–	300
AmoebaNet-A [4]	Cell-based	2018	3.34	3.2	3150
CNN-GA [7]	Cell-based	2018	3.22	2.9	35
LEMONADE [6]	Cell-based	2018	2.58	13.1	80
LEMONADE [6]	Cell-based	2018	4.57	0.5	80
NSGANet [9]	Cell-based	2019	4.67	0.2	27

in automating the design of neural networks. While this performance was commendable at the time, subsequent techniques have surpassed it, reflecting the rapid advancements in the field and the continual refinement of search strategies and architectures.

Hierarchical Evolution, introduced later the same year, mimics the modularized (cell-based) design pattern typically employed by human experts, enabling it to discover architectures that outperform many manually designed models for image classification. This approach reduced the error rate to 3.63%, demonstrating the efficacy of incorporating hierarchical structures into the architecture search process. Further refinements in Evolutionary methods, as seen in AmoebaNet-A (2018), brought the error rate down to 3.34%. This improvement reflects the modification of the tournament selection Evolutionary Algorithm by introducing an age property to favor the younger genotypes.

CNN-GA, also introduced in 2018, slightly improved on this with an error rate of 3.22%, highlighting the effectiveness of genetic algorithms specifically customized to convolutional neural networks. The important advancement in accuracy came from LEMONADE (2018), which achieved an error rate of 2.58%. This improvement can be attributed to its Lamarckian inheritance mechanism, allowing child networks to inherit and benefit from the predictive performance of their trained parents, enhancing the efficiency of the architecture search process. While LEMONADE's primary variant excelled in accuracy, a secondary version of the algorithm produced an error rate of 4.57%, using significantly fewer parameters (0.5 million compared to the original's 13.1 million). This result highlights the inherent trade-off between model complexity and accuracy in NAS. Reducing the number of parameters often leads to less accurate models, but in this case, the parameter-efficient variant of LEMONADE still performs competitively, making it suitable for applications where computational resources are limited.

The evolution of NAS methods has also focused on reducing model complexity, as reflected in the number of parameters required for each architecture. Early methods, such as LargeEvo and AmoebaNet-A, required substantial model sizes, with the latter using 3.2 million parameters. In contrast, CNN-GA managed to reduce this to 2.9

million parameters while maintaining strong accuracy, demonstrating that optimized genetic algorithms can lead to more efficient models without sacrificing performance.

NSGANet (2019) represents a significant step toward parameter efficiency, utilizing only 0.2 million parameters while achieving an error rate of 4.67%. This method underscores the growing emphasis on lightweight models that maintain competitive performance, a crucial development for deploying NAS-designed architectures in real-world applications where computational and memory resources are constrained.

Computational cost, often measured in GPU days, is a key factor in determining the practicality of NAS algorithms. LargeEvo, as one of the earliest approaches, required a staggering 2750 GPU days, which made it infeasible for widespread use. AmoebaNet-A, despite its improvements in accuracy, further increased this cost to 3150 GPU days, reflecting the high computational burden of early Evolutionary NAS techniques.

However, recent methods have made significant strides in reducing computational costs. CNN-GA, for instance, requires only 35 GPU days, while NSGANet reduces this further to 27 GPU days. These reductions are crucial for making NAS more accessible and feasible for broader adoption, as they lower the barrier to entry for organizations without vast computational resources.

B. ***Performance Comparison on CIFAR-100***

The performance of Evolutionary NAS algorithms on CIFAR-100 also shows a clear progression in accuracy, model efficiency, and computational cost. GeNet (2017) achieved an error rate of 29.03%, requiring 17 GPU days, but it set the stage for further advancements. LargeEvo (2017) reduced the error to 23% with 40.4 million parameters, though at the cost of 2750 GPU days, highlighting the trade-off between accuracy and computational resources.

By 2018, CNN-GA achieved a significant improvement with a 20.53% error rate using only 4.1 million parameters and requiring 40 GPU days, making it much more computationally feasible. The evolution of NAS methods culminated with NSGANet (2019), which achieved an error rate of 4.67% with just 0.2 million parameters, requiring only 27 GPU days, marking a major advancement in efficiency and performance.

Overall, the results indicate substantial progress in reducing error rates and computational demands, with NSGANet standing out as the most efficient method on CIFAR-100, demonstrating the growing potential of Evolutionary NAS for handling complex tasks with minimal resources (Table 2.3).

2.4 Future Research

The field of Evolutionary Neural Architecture Search (NAS) offers numerous opportunities for exploration and innovation. Future research could focus on enhancing genetic operations by investigating more sophisticated techniques, such as advanced

Table 2.3 Performance comparison of Evolutionary NAS algorithms on CIFAR-100

References	Search space	Year	Error rate (%)	Parameters (Millions)	GPU days
GeNet [10]	Cell-based	2017	29.03	–	17
LargeEvo [2]	Chain-structured	2017	23	40.4	2750
CNN-GA [7]	Cell-based	2018	20.53	4.1	40
NSGANet [9]	Cell-based	2019	4.67	0.2	27

crossover methods and multi-objective optimization. Incorporating additional objectives, like energy efficiency, would enable a more comprehensive evaluation of architectures. Additionally, expanding the application of Evolutionary NAS to emerging domains, including autonomous systems, healthcare, and edge computing, could lead to significant advancements. Customizing architectures to address the specific requirements of these fields may yield innovative solutions and improved performance. Overall, these directions highlight the potential for Evolutionary NAS to evolve and adapt, driving further breakthroughs in neural network design.

2.5 Summary

This chapter provided a detailed exploration of Evolutionary Algorithm-based Neural Architecture Search (NAS) for image classification. We outlined the fundamental concepts behind Evolutionary NAS, illustrating how natural selection principles—such as mutation, crossover, and selection—can be employed to evolve neural network architectures across generations. This iterative process enables the efficient exploration of vast search spaces, allowing for the automatic discovery of high-performing architectures without the need for human intervention.

We discussed key components of Evolutionary NAS, including the representation of candidate architectures, genetic operations that foster diversity, and fitness evaluation mechanisms that guide the search process toward optimal solutions. Furthermore, we presented a performance comparison of various Evolutionary NAS methods, highlighting advancements in accuracy, model complexity, and computational efficiency. Finally, we considered future research directions, emphasizing the need for further improvements in balancing predictive performance with computational costs to enhance the applicability of Evolutionary NAS in real-world scenarios.

Bibliography

1. B. Pfeifer, A. Holzinger, M.G. Schimek, *Robust Random Forest-Based All-Relevant Feature Ranks for Trustworthy AI the Creative Commons Attribution Non-Commercial License 4.0* (*CC BY-NC 4.0*) (2022). https://doi.org/10.3233/SHTI220418
2. E. Real, S. Moore, A. Selle, S. Saxena, Y.L. Suematsu, J. Tan, Q.V. Le, A. Kurakin, Large-scale evolution of image classifiers, in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70 (2017), pp. 2902–2911. JMLR.org
3. B. Zoph, G. Brain, V. Vasudevan, J. Shlens, Q.V. Le, G. Brain, *Learning Transferable Architectures for Scalable Image Recognition* (2018). https://doi.org/10.1109/CVPR.2018.00907
4. E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33 (2019), pp. 4780–4789
5. H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu, Hierarchical representations for efficient architecture search. http://arxiv.org/abs/1711.00436 (2017)
6. T. Elsken, J.H. Metzen, F. Hutter, Efficient multi-objective neural architecture search via Lamarckian evolution. http://arxiv.org/abs/1804.09081 (2018)
7. Y. Sun, B. Xue, M. Zhang, G.G. Yen, J. Lv, Automatically designing CNN architectures using the genetic algorithm for image classification. IEEE Trans. Cybern. **50**(9), 3840–3854 (2020)
8. S. Ali, M.A. Wani, Gradient-based neural architecture search: a comprehensive evaluation. Mach. Learn. Knowl. Extr. **5**, 1176–1194 (2023). https://doi.org/10.3390/make5030060
9. Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, V. N. Boddeti, Multi-criterion Evolutionary design of deep convolutional neural networks. http://arxiv.org/abs/1912.013 (2019)
10. L. Xie, A. Yuille, Genetic CNN, in *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 1379–1388
11. S. Ali, M.A. Wani, Recent trends in neural architecture search systems, in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA), Nassau, Bahamas* (2022), pp. 1783–1790. https://doi.org/10.1109/ICMLA55696.2022.00272

Chapter 3
Gradient-Based Neural Architecture Search

3.1 Introduction

The rapid advancement of deep learning has revolutionized numerous fields, from computer vision to natural language processing. However, the effectiveness of deep learning models heavily relies on the choice of their architectures. Traditionally, designing these architectures has been a labor-intensive and expert-driven process. To address this challenge, Neural Architecture Search (NAS) has emerged as a promising solution, aiming to automate the architecture design process.

Within the realm of NAS, various techniques have been explored, including Evolutionary Algorithms and reinforcement learning. While these methods have shown promise, they are often computationally expensive and time-consuming. In contrast, Gradient-based NAS techniques have gained attention for efficiently exploring the search space.

Among these Gradient-based approaches, Differentiable Architecture Search (DARTS) stands out for its innovative formulation of the architecture search problem as a continuous optimization task. By allowing for the simultaneous optimization of model parameters and architecture through Gradient descent, DARTS significantly reduces the computational costs associated with traditional NAS methods. This unique framework facilitates exploration of the architecture search space, enabling rapid convergence to high-performance models.

In this chapter, we will provide a comprehensive overview of Gradient-based NAS, focusing specifically on DARTS, a basic method in this category. We will begin by discussing the fundamental concepts of Gradient-based NAS, transitioning into a detailed examination of the DARTS methodology. By the end of this chapter, readers will gain a clearer understanding of how Gradient-based techniques, particularly DARTS, are reshaping the landscape of neural network design.

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_3

3.2 Gradient-Based NAS Methods

Gradient-based Neural Architecture Search (NAS) methods offer a powerful approach to optimizing neural architecture designs by incorporating Gradient optimization techniques. These methods focus on selecting the best architectures from a predefined search space through a process known as bi-level optimization. This involves simultaneously learning both the operation strength parameters and the weights of the neural network. The operation strength parameters determine the contributions of different operations within the network, while the weights are the traditional parameters of the network layers.

In Gradient-based NAS, the cell-based approach is predominantly used, where the search process revolves around a modular unit known as a "cell." Each cell is a small unit of a neural network that can be configured in various ways using different operations like convolutions, pooling, and activation functions. During the search phase, the NAS algorithm optimizes the architecture of this cell by learning which operations should be included and how they should be connected. Once the optimal cell architecture is identified, it is used as a building block to construct the final, larger neural network model.

The Gradient-based NAS approach operates with two types of cells: normal cells and reduction cells. Normal cells generate a feature map that preserves the spatial dimensions of the input, using convolution and pooling operations with a stride of 1. In contrast, reduction cells produce feature maps with reduced spatial dimensions, employing operations like convolution and pooling with a stride of 2.

Differentiable Architecture Search (DARTS) stands as a widely recognized NAS method for automation of designing neural networks. The core premise of DARTS lies in its capacity to seek out good neural architectures by considering architecture as a variable (continuous), enabling optimization through Gradient-based approaches.

Below, we present an overview of Gradient-based NAS. We will explore the foundational aspects of DARTS, focusing on how it effectively searches for optimal cell architectures. The key foundational aspects of DARTS include.

3.2.1 Search Space

The search space in DARTS is a central component that defines the possible architectures the method can explore during the optimization process. It consists of a collection of candidate operations and their arrangement within a modular framework known as cells.

A. Cell Representation

In DARTS, the architecture is built from stacked cells, where each cell is represented as a Directed Acyclic Graph (DAG) structure as shown in Fig. 3.1. This structure allows for the flow of information from one operation to another, enabling complex

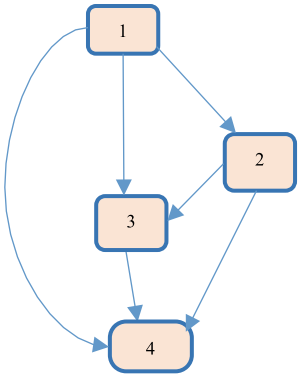


Fig. 3.1 DAG representing cell structure in DARTS

interactions between the operations. A DAG comprises N nodes $x^{(i)}$, representing a collection of feature maps, and edges (k, l) representing various operations $o^{(k,l)}$ applied to these feature mappings.

In the DARTS cell, there are four nodes, with every cell having two inputs and one output. The inputs for the current cell are provided by the outputs of the two cells that precede it. To obtain the cell's output, a concatenation operation is implemented on all the middle nodes within the cell. The value of a middle node is computed by considering the information from all the nodes that precede it

$$x^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right) \tag{3.1}$$

This approach allows for efficient exploration of the search space, as the optimization focuses on a smaller, more manageable unit rather than the entire network. Gradient-based NAS methods have demonstrated the ability to find highly effective cell architectures with reduced computational resources compared to traditional search methods.

B. Candidate Operations

DARTS explores a diverse set of operations within a Convolutional Neural Network (CNN) cell, encompassing seven potential operations listed in Table 3.1. Each operation represents a different way to process data within the network structure. Importantly, there is also a zero operation which signifies no connection between nodes, providing flexibility in architectural configurations.

Initially, these operations are placed between each pair of nodes within the cell, with their associated weights set to zero. As the search progresses, DARTS dynamically learns the strengths and weights for each operation through a Gradient-based optimization process. This iterative approach allows DARTS to adaptively adjust the contributions of each operation within the cell, aiming to maximize the performance of the neural network architecture on the given task.

Table 3.1 Operation space

Type of operation	Size of kernel
Dilated separable convolutions	(3 × 3 and 5 × 5)
Separable convolutions	(3 × 3 and 5 × 5)
Identity	–
Max pooling	3 × 3
Average pooling	3 × 3

3.2.2 Base Network for Search

In DARTS, the base network utilized during the architecture search phase is a relatively simple structure consisting of eight cells, as shown in Fig. 3.2. Within this configuration, the third and sixth cells serve as reduction cells, which are specifically designed to down-sample the feature maps, effectively managing the spatial dimensions. The remaining cells are designated as normal cells, focusing on feature extraction through various operations.

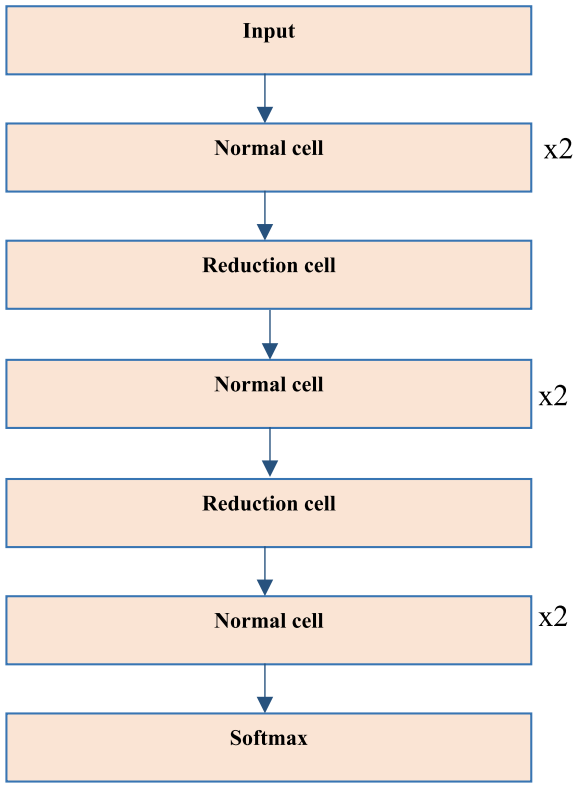


Fig. 3.2 Base network used in DARTS for cell search. The notation (x2) indicates that two cells are stacked

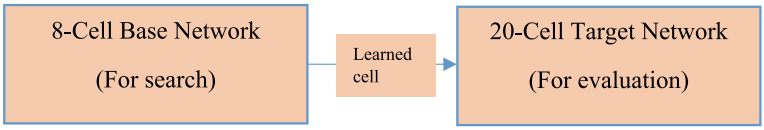


Fig. 3.3 Difference in architecture depths during search and evaluation phases

While this base network provides a foundational framework for exploring different cell configurations, it is important to note that the evaluation of the discovered cell architecture occurs within a deeper network that includes 20 cells, as illustrated in Fig. 3.3. This deeper network evaluation allows for a more comprehensive assessment of the architecture's performance across a wider range of complexities and input variations.

One of the critical challenges in DARTS arises from the inherent differences in behavior between shallow and deep networks. The configurations identified during the architecture search phase may not necessarily yield optimal results when evaluated within the deeper network. This discrepancy is commonly referred to as the: depth gap." The depth gap highlights the potential pitfalls of relying solely on shallow architectures for guiding the search process, as they may not capture the full range of interactions and dependencies present in deeper architectures.

3.2.3 Search Strategy

The Gradient-based search strategy in DARTS comprises two key steps: Continuous Relaxation and Bi-level Optimization. Continuous relaxation transforms discrete architectural decisions into a continuous space, allowing Gradients to flow smoothly through architecture parameters during optimization. This transformation enables the integration of architecture search directly into the training process, significantly enhancing the overall search efficiency.

Bi-level optimization further refines this approach by simultaneously handling the search for optimal architectures and model training. This framework allows for the iterative refinement of both the architecture and its weights. Below, we explain these steps in detail:

A. **Continuous Relaxation**

In DARTS-based methods, a key aspect is the placement of mixed operations on the edges connecting nodes within a cell. In this approach, each edge, denoted as $o^{(i,j)}$ in the directed acyclic graph represents a combination (linear) of all the candidate operations. Every operation is assigned a weight, and these weights, collectively represented as α', sum up to 1.

As a result, every edge embodies a weighted combination of all operations within the search space. The variable α' can be fine-tuned using Gradient descent. The

goal is to find the optimal cell topologies by learning a collection of variables (α') that specify how the operations should be combined for a provided set of nodes. The categorical decision-making process for selecting operations is converted into a Softmax operation encompassing all feasible operations in DARTS, as described in Eq. (3.2).

$$\overline{o}^{(i,j)}(x) = \sum_{o \in O} \frac{\exp\left(\alpha'^{(i,j)}\right)}{\sum_{o' \in O} \exp\left(\alpha_{o'}{}'^{(i,j)}\right)} o(x) \tag{3.2}$$

where $\overline{o}^{(i,j)}(x)$ depicts blended operations applied to each edge.

Figure 3.4 visually illustrates the continuous relaxation within a cell. In these methods, the complexity of seeking a network architecture is simplified by focusing on acquiring a set of parameters, denoted as, $\alpha' = \left\{\alpha'^{(i,j)}\right\}$, which depict the strengths of different operations positioned along the edges connecting pairs of nodes within the cell. Following the completion of the search process, the discrete neural architecture is constructed by replacing the blended operations between nodes with the operation choices that are most probable.

B. Two-Level Optimization

Following the continuous relaxation, the subsequent step involves two-level optimization, which seeks to simultaneously acquire both α' (operation strength parameters) and the weights ($\hat{w}$) associated with every operation in the network (Fig. 3.5). The L_{train} (training loss) and L_{val} (validation loss) are influenced by the network

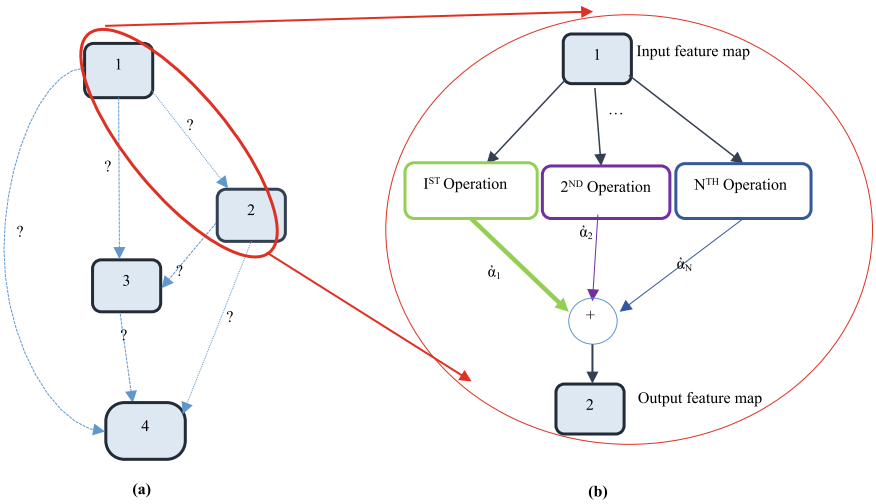


Fig. 3.4 Continuous relaxation of the search space in DARTS cell. **a** Visualization of the DAG corresponding to cell structure **b** Shows mixed operation. (The count of operations positioned on the edges linking every pair of nodes is indicated by N.)

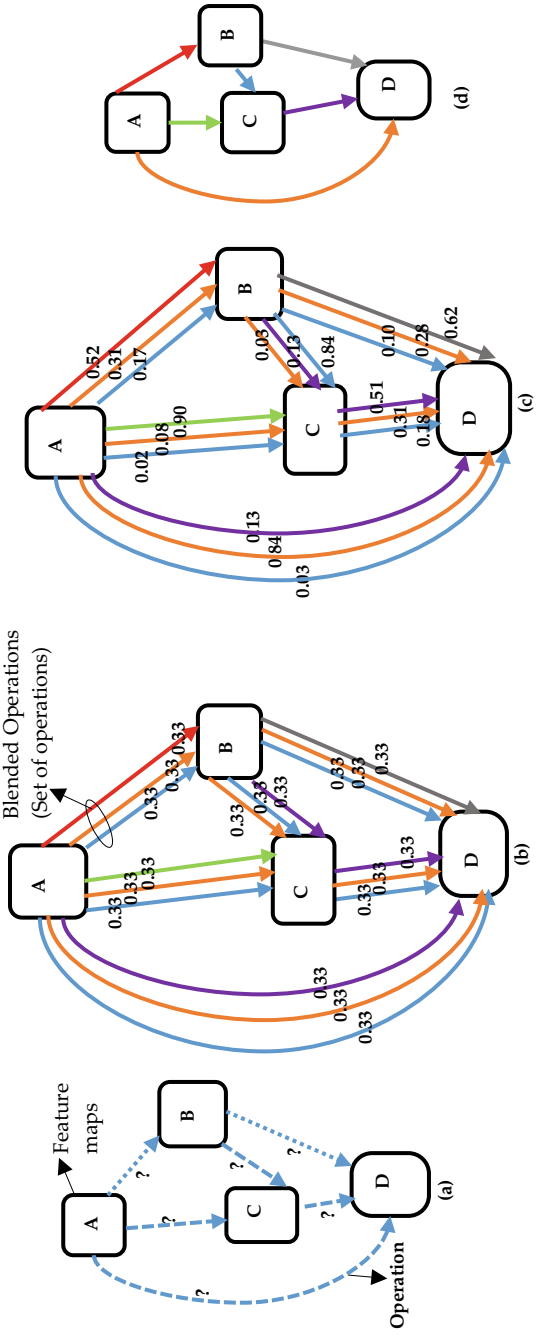


Fig. 3.5 DARTS Summary **a** The operations at the edges are initially unspecified. **b** Continuous relaxation of the search space **c** Resolution of a two-level optimization problem to enhance mixing probabilities and network weights simultaneously. **d** Utilization of acquired mixing probabilities to deduce the final architecture

configuration as well as the weights ($\hat{w}$) of network. In the context of architecture search, the objective is to obtain α'^{*} which reduces $L_{val}\left(\hat{w}^{*}, \alpha'^{*}\right)$, where the optimal weights $\hat{w}^{*}$ for the given network are found by reducing the training loss: $\hat{w}^{*} = argmin\ \hat{w}L_{train}\left(\hat{w}, \alpha'^{*}\right)$. This challenge can be formulated as a two-level optimization problem, as denoted by Eqs. (3.3 and 3.4) below:

$$\min_{\alpha'} \quad L_{val}\left(\hat{w}\left(\alpha'\right), \alpha'\right) \tag{3.3}$$

$$\text{s.t.}\hat{w}^{*}\left(\alpha'\right) = argmin_{\hat{w}}L_{train}\left(\hat{w}, \alpha'\right) \tag{3.4}$$

The optimization problem is framed as finding the optimal α' values to minimize the validation loss, assuming that the weights are already fine-tuned on the training data. Nevertheless, Eq. (3.3) introduces a computational difficulty. Achieving optimal weights necessitates training the network to minimize the training loss, which involves frequent adjustments to the weights as the network's design parameter $\left(\alpha'\right)$ evolves. This iterative process renders training infeasible and demands substantial computational time and resources. To tackle the challenge presented by the inner optimization step, the authors proposed a simple approximation technique to compute the Gradient, involving

$$\nabla\alpha' L_{val}\left(\hat{w}\left(\alpha'\right), \alpha'\right) \tag{3.5}$$

$$\approx \nabla\alpha' L_{val}\left(\hat{w} - \xi\nabla w L_{train}\left(\hat{w}, \alpha'\right), \alpha'\right) \tag{3.6}$$

where $\hat{w}$ represents the existing weights and ξ represents the learning rate. Instead of pursuing full convergence through continuous training for the inner optimization (Eq. 3.3), the approach aims to approximate $\hat{w}^{*}\left(\alpha'\right)$ by adjusting $\hat{w}$ in one training iteration.

The formula to update the weights ($\hat{w}$) based on the training loss is given as follows:

$$\hat{w}^{*} = \hat{w} - \xi\nabla\hat{w}L_{train}\left(\hat{w}, \alpha'\right) \tag{3.7}$$

Additionally, the formula for updating the operation strength parameter ($\alpha\prime$) guided by the validation loss is expressed as follows:

$$\alpha'^{*} = \alpha' - \nabla\alpha' L_{val}\left(\hat{w} - \xi\nabla\hat{w}L_{train}\left(\hat{w}, \alpha'\right), \alpha'\right) \tag{3.8}$$

In these equations, $\nabla\hat{w}L_{train}$ represents the training loss Gradient concerning the weights ($\hat{w}$) and $\nabla\alpha\prime L_{val}$ represents the validation loss Gradient concerning the hyperparameter, α'. The step size of the update is determined by the learning rate ξ.

3.2.4 Obtaining the Optimal Neural Architecture in DARTS

In DARTS, the process of obtaining the best neural architecture begins with the exploration of a continuously relaxed architecture search space. Instead of making early commitments to discrete architectural choices, DARTS employs a weighted combination of possible operations. This flexibility allows the optimization process to utilize gradient descent effectively, enabling the search for optimal configurations without being constrained by binary decisions.

After the optimization phase, the next step is to convert this relaxed representation into a discrete cell architecture. This is achieved by selecting the top-ranked operations based on their learned weights. Specifically, DARTS retains the operations that have the highest probabilities, ensuring that the most promising configurations are incorporated into the final model.

This technique allows for a more informed and data-driven approach to neural architecture design, ultimately leading to architectures that are not only efficient but also well-suited to the tasks they are intended to perform. By utilizing the strengths of both continuous relaxation and discrete selection, DARTS effectively bridges the gap between theoretical exploration and practical implementation, resulting in high-performing neural networks tailored for diverse applications.

3.2.5 Final Cell Architectures

In this section, we present the final cell architectures (both normal and reduction), learned using the DARTS framework. Each DARTS cell has two inputs, c_{k-1} and c_{k-2}, which represent the output and input of the previous cell, respectively. The architecture is designed to produce a single output, c_{k}. Figure 3.6 shows the normal cell learned through DARTS. Normal cells are designed for feature extraction and consist of various operations, such as convolutions and pooling, optimized to capture essential patterns in the data. These configurations enable the network to effectively process input while maintaining computational efficiency.

Reduction cells, on the other hand, are responsible for down-sampling the feature maps. They help manage the spatial dimensions of the data as it passes through the network, allowing for deeper and more abstract feature representations. Together, these learned normal and reduction cells form the building blocks of the target architecture, ensuring robust performance across various tasks. Figure 3.7 shows the reduction cell architecture learned through DARTS.

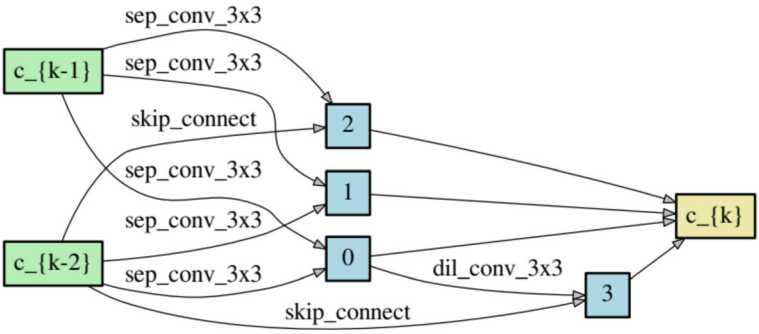


Fig. 3.6 Normal cell learned through DARTS

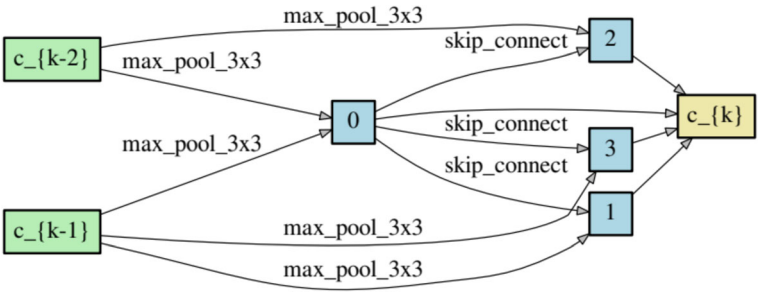


Fig. 3.7 Reduction cell learned through DARTS

3.2.6 Training and Evaluation in Target Network

In the DARTS framework, once the cell architectures are learned, both normal and reduction cells are utilized to construct a deeper architecture, typically comprising 20 cells, as shown in Fig. 3.8. This 20-cell deep architecture is then employed for the evaluation of the learned cells, allowing for an assessment of their performance on various tasks.

The decision to use a shallower network during the architecture search process is primarily motivated by the need to save computational resources. By limiting the depth of the network during the search phase, the process becomes more efficient, enabling faster experimentation and iteration. This approach helps in rapidly identifying promising architectures without incurring the high computational costs associated with training deep networks.

Once optimal cell architectures are established, they are integrated into the deeper model for thorough evaluation, ensuring that the discovered configurations are both effective and practical for deployment. The deeper target network is then trained from scratch and evaluated for performance.

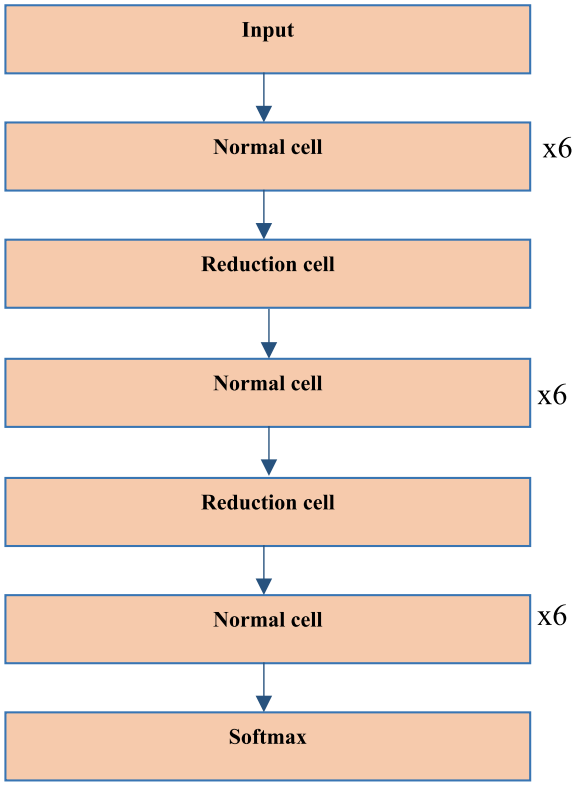


Fig. 3.8 Target network used in DARTS for cell evaluation. The notation (x6) indicates that six cells are stacked

3.3 Results and Discussion

In this section, we present the results of the DARTS method and compare its performance with few established NAS methods, including Reinforcement Learning-based methods and Evolutionary Algorithms. Our evaluation is based on various benchmarks, focusing on accuracy, efficiency, and the quality of the discovered architectures. Table 3.2 shows the performance of various Neural Architecture Search (NAS) methods on the CIFAR-10 dataset.

Evolutionary Algorithms have proven to be effective in the domain of NAS on image classification tasks. NSGANETv1 showcased both accuracy and efficiency, achieving an impressive 97.98% accuracy within 27 days. OSNAS followed suit with a noteworthy accuracy of 97.44% in 4 days. These methods demonstrate the capacity of Evolutionary approaches to explore the architecture space and uncover high-performing solutions, though with varying computational demands.

Reinforcement Learning techniques have also exhibited promising results in NAS endeavors. MetaQNN attained an accuracy of 93.08% with 11.2 million parameters, requiring a substantial 100 GPU days, on the CIFAR-10 dataset. BlockQNN demonstrated a remarkable accuracy of 97.42%, utilizing reinforcement learning techniques

Table 3.2 Comparative evaluation of different NAS methods on the CIFAR-10 dataset

Reference	Search method	Accuracy (%)	Parameters (Millions)	Total GPU days
MetaQNN	Reinforcement learning	93.08	11.2	100
BlockQNN	Reinforcement learning	97.42	3.3	96
NSGANETv1	Evolutionary Algorithm	97.98	2.9	27
OSNAS	Evolutionary Algorithm	97.44	3.3	4
DARTS	Gradient optimization	97.24	3.3	4

within a span of 96 GPU days. These results highlight the potential of reinforcement learning to guide the search process effectively, leading to competitive architectures.

Gradient optimization-based methods, emphasizing computational efficiency, have displayed strong performance on the datasets for the classification of images like CIFAR-10. DARTS, as an exemplar, reached an accuracy of 97.24% with 3.3 million parameters, all in just 4 days of GPU time. This method underscores the advantage of rapid exploration through their efficient search process.

In summary, the performance comparison of various NAS methods on the CIFAR-10 dataset reveals distinct trends among different search paradigms. Evolutionary Algorithms have showcased competitive accuracies, with methods like NSGANETv1 and OSNAS demonstrating notable efficiency. Reinforcement learning approaches, exemplified by MetaQNN and BlockQNN, have proven effective in achieving high accuracies, albeit with varying computational investments. Gradient optimization techniques, as demonstrated by DARTS, emphasize computational efficiency and rapid exploration, achieving remarkable accuracies in short periods. The choice of NAS method depends on a trade-off between accuracy and computational resources, with each approach catering to different preferences and priorities.

3.4 Future Directions

Future research directions for Gradient-based NAS should focus on addressing the challenges associated with the determination of optimal cell structures. Developing methods that can effectively navigate the complexities of cell design will enhance the search process. Additionally, overcoming the depth gap encountered in DARTS during the transition from architecture search to evaluation is crucial; this may involve innovating techniques that ensure smoother transitions and better performance assessments. Moreover, refining the operations utilized in DARTS to minimize the impact of irrelevant characteristics will be essential for improving the efficiency and accuracy of the search process. On the other hand, the strengths of cell-based

search spaces should be leveraged further. Future studies could explore novel configurations and combinations of candidate operations within cells to maximize flexibility in design. Additionally, investigating ways to customize stacked cells could lead to more adaptive and efficient neural architectures, ultimately contributing to the advancement of automated design methodologies in deep learning.

3.5 Summary

This chapter provided a comprehensive overview of Gradient-based Neural Architecture Search (NAS). The foundational concepts of this approach were discussed. The details of how it automates the task of neural network architecture design that offers solutions to the limitations of manual processes were presented. By focusing on DARTS, we illustrated its effectiveness in optimizing neural architectures, particularly for image classification tasks. The experimental comparisons highlighted DARTS's performance, showcasing its advantages in terms of accuracy and computational efficiency. The chapter has equipped readers with a solid understanding of Gradient-based NAS, specifically how DARTS operates, its mechanisms, and its potential to advance the field of deep learning.

Bibliography

1. B. Pfeifer, A. Holzinger, M.G. Schimek, *Robust Random Forest-Based All-Relevant Feature Ranks for Trustworthy AI the Creative Commons Attribution Non-Commercial License 4.0* (*CC BY-NC 4.0*) (2022). https://doi.org/10.3233/SHTI220418
2. B. Zoph, Q.V. Le, G. Brain, *Neural Architecture Search with Reinforcement Learning*
3. B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning. https://bowenbaker.github.io/metaqnn/. (Online)
4. E. Real et al., *Large-Scale Evolution of Image Classifiers* (2017)
5. B. Zoph, G. Brain, V. Vasudevan, J. Shlens, Q.V. Le, G. Brain, *Learning Transferable Architectures for Scalable Image Recognition* (2018). https://doi.org/10.1109/CVPR.2018.00907
6. E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search. www.aaai.org. (Online)
7. J. Mun, S. Ha, J. Lee, DE-DARTS: neural architecture search with dynamic exploration. ICT Express **9**(3), 379–384 (2023). https://doi.org/10.1016/j.icte.2022.04.005
8. H. Liu, K. Simonyan, Y. Yang, DARTS: differentiable architecture search, in International Conference on Learning Representations, ICLR 2019 (2019), pp. 1–13
9. X. Chen, L. Xie, J. Wu, Q. Tian, Progressive differentiable architecture search: bridging the depth gap between search and evaluation, in *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 1294–1303. https://doi.org/10.1109/ICCV.2019.00138
10. X. Chu, Fair DARTS: eliminating unfair advantages in differentiable architecture search. https://github.com/xiaomi-automl/FairDARTS. (Online)
11. K. Nakai, T. Matsubara, K. Uehara, *Att-DARTS: Differentiable Neural Architecture Search for Attention*

12. C.-H. Hsu et al., MONAS: multi-objective neural architecture search. www.tensorflow.org/tutorials/deep. (Online)
13. Z. Zhong et al., *BlockQNN: Efficient Block-wise Neural Network Architecture Generation*
14. C.J.C.H. Watkins, Learning from delayed rewards, pp. 1–234. Ph.D. Thesis (1989)
15. V. Mnih et al., Human-level control through deep reinforcement learning. Nature **518** (2015). https://doi.org/10.1038/nature14236
16. L.-J. Lin. Reinforcement Learning for Robots using Neural Networks. Carnegie Mellon University, USA (1992).
17. M. Längkvist, L. Karlsson, A. Loutfi, Inception-v4, Inception-ResNet and the impact of residual connections on learning. Pattern Recognit. Lett. **42**(1), 11–24 (2014). http://arxiv.org/abs/1512.00567. (Online)
18. X. Chen, L. Xie, J. Wu, Q. Tian, Imagenet classification with deep convolutional neural networks. https://github.com/. (Online)
19. G. Huang, S. Liu, L. Van Der Maaten, K.Q. Weinberger, *CondenseNet: An Efficient DenseNet using Learned Group Convolutions*
20. G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 217 (2017), pp. 2261–2269. https://doi.org/10.1109/CVPR.2017.243
21. Y. LeCun, Y. Bengio, G. Hinton, G. Deep learning. Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539
22. J. Peters, S. Schaal, Policy Gradient methods for robotics, in *IEEE International Conference on Intelligent Robots and Systems* (2006), pp. 2219–2225. https://doi.org/10.1109/IROS.2006.282564
23. Z. Lu et al., Multi-objective Evolutionary design of deep convolutional neural networks for image classification. https://github.com/mikelzc1990/nsganetv1. (Online)
24. H. Zhang, Y. Jin, K. Hao, Evolutionary search for complete neural network architectures with partial weight sharing. IEEE Trans. Evol. Comput. **26**(5), (2022). https://doi.org/10.1109/TEVC.2022.3140855
25. J. Hu, Squeeze-and-Excitation_Networks_CVPR_2018_paper.pdf, CVPR (2018), pp. 7132–7141. http://openaccess.thecvf.com/content_cvpr_2018/html/Hu_Squeeze-and-Excitation_Networks_CVPR_2018_paper.html. (Online)
26. F. Juefei-Xu, V.N. Boddeti, M. Savvides, *Local Binary Convolutional Neural Networks* (2017). https://doi.org/10.1109/CVPR.2017.456
27. L. Chen, D. Alahakoon, NeuroEvolution of augmenting topologies with learning for data classification, in *2006 International Conference on Information and Automation* (2006). https://doi.org/10.1109/ICINFA.2006.374100
28. X. Chen, L. Xie, J. Wu, Q. Tian, Progressive differentiable architecture search: bridging the depth gap between search and evaluation. https://github.com/. (Online)
29. H. Cai, L. Zhu, S. Han, Proxylessnas: direct neural architecture search on target task and hardware. https://github.com/MIT-HAN-LAB/ProxylessNAS. (Online)
30. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* (2015). https://doi.org/10.1038/nature14539
31. S. Ali, M.A. Wani, Recent trends in neural architecture search systems, in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA), Nassau, Bahamas* (2022), pp. 1783–1790. https://doi.org/10.1109/ICMLA55696.2022.00272
32. S. Ali, M.A. Wani, Gradient-based neural architecture search: a comprehensive evaluation. Mach. Learn. Knowl. Extr. **5**, 1176–1194 (2023). https://doi.org/10.3390/make5030060

Chapter 4
Efficient Training of Deep Learning Architectures

4.1 Introduction

The development of deep learning models involves two key stages: architecture generation and training. Traditionally, creating neural architectures involves a manual process requiring extensive adjustments and deep expertise. Recently, however, there has been a significant shift with the advent of automated methods for architecture generation, such as Neural Architecture Search (NAS). NAS represents a major advancement by automating the architecture generation process, which has led to considerable improvements in model development.

While substantial advancements have been made in architecture generation—from manual to automated methods—less focus has been given to the training phase of deep learning model development. Traditional training techniques have dominated this stage. To address this gap, a novel training method based on a coarse-to-fine-tuning approach is introduced. This method starts with coarse training of the target architecture, allowing the neural network to learn general features applicable to various cases but with lower accuracy. In the subsequent fine-tuning stage, this initial broad understanding is refined to enhance task-specific accuracy. The fine-tuning process incorporates two techniques: Simple Selective Freezing (SSF) and Progression-Based Selective Freezing (PSF). These techniques selectively freeze certain layers during fine-tuning to better control adaptation and improve overall classification accuracy.

To evaluate the effectiveness of the new training method, experiments using a 20-cell deep architecture generated through a Gradient-based architecture search were conducted. The results on the CIFAR-10 dataset demonstrate that training the architecture with the new coarse-to-fine-tuning approach yields improved accuracy. Additionally, experiments were performed to assess the effectiveness of this approach on the same architecture using the CIFAR-100 and MNIST datasets for evaluating performance across different datasets.

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_4

In conclusion, this chapter presents a new approach to improve model training through a novel coarse-to-fine-tuning method applied to a deep architecture generated via manual or automated methods. By examining the effectiveness of the new approach across multiple datasets, the aim is to highlight its potential to enhance model performance and accuracy. These insights will contribute to advancing training methodologies and their applicability in diverse deep learning tasks.

4.2 Issues with the Traditional Training Approach

The process of deep learning involves two main stages: architecture generation and training (Fig. 4.1). In the architecture generation stage, the structure of the neural network is established within a predefined search space, using either manual or automated methods. Following this, the defined architecture undergoes training on the specific task it is intended to perform.

In traditional deep learning training methods, neural network weights are initialized randomly and then optimized using Gradient-based techniques like stochastic gradient descent (SGD). During this process, the network iteratively adjusts its parameters to minimize a predefined loss function, computing gradients and updating weights over multiple epochs. Training continues until the model converges or achieves an acceptable level of performance. However, this approach can sometimes lead to suboptimal results. This issue arises when adjustments in earlier layers alter the weight distribution throughout the network, which can negatively impact learning in subsequent layers and result in less-than-ideal solutions.

Training the entire network simultaneously can cause changes in earlier layers to affect the training of later layers, potentially impairing their ability to learn meaningful representations. This issue is especially pronounced in deep networks, where the cumulative effects across multiple layers can hinder the network's ability to converge to an optimal solution. Addressing this challenge, a new training approach designed to enhance the accuracy of deep learning models is developed, which is detailed in the following section.

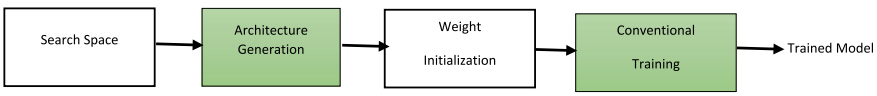


Fig. 4.1 Two-stage process of model development with conventional training in deep learning

4.3 Efficient Training Via Coarse-to-Fine-Tuning

While significant attention has been given to architecture generation in deep learning, the training phase has received comparatively less focus. To bridge this gap a coarse-to-fine-tuning approach is introduced, which incorporates selective freezing to refine the training process and address this imbalance.

4.3.1 Coarse Training

Initially, the target network is trained to capture general features that apply across a range of scenarios, aiding in the classification of various classes but with reduced accuracy. The network begins with random weights and is trained through multiple epochs, adjusting parameters iteratively to minimize the loss function. This coarse training phase follows traditional deep learning methods but has a different objective: rather than aiming to achieve the final model, it focuses on learning broad, general features. This foundational step ensures that the network establishes a solid base by identifying generalizable patterns in the data, which will be refined in subsequent stages.

4.3.2 Selective Freezing

The new method, known as selective freezing, provides a simple yet effective solution to mitigate the impact of changes in certain layers on the training of other layers within the network. Selectively freezing a portion of the network's weights during training, can reduce the influence of adjustments in these frozen layers on subsequent layers. This approach allows for more focused parameter tuning and leads to improved learning outcomes.

Selective freezing supports coarse-to-fine-tuning, which starts with training the network to learn coarse features and then refines the model's performance through focused fine-tuning. Figure 4.2 illustrates the block diagram of the coarse-to-fine-tuning approach with selective freezing.

Two methods are presented for implementing selective freezing: Simple Selective Freezing and Progression-Based Selective Freezing. Both methods facilitate incremental weight learning, allowing the model to improve its accuracy through iterative

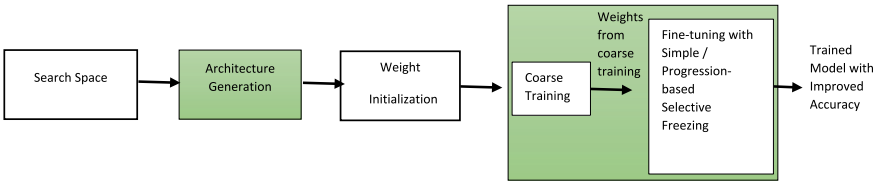


Fig. 4.2 Two-stage model development process with the new coarse-to-fine-tuning approach: Utilizing simple and progression-based selective freezing during fine-tuning

refinement. The following sections detail these two approaches to selective freezing during fine-tuning.

A. Simple Selective Freezing

In a network with depth N, the Simple Selective Freezing (SSF) technique divides the weights ω learned during coarse training into two parts: the weights ω_1 associated with the first $N/2$ layers (or cells) are selectively frozen, while the weights ω_2 of the remaining $N/2$ layers are optimized during fine-tuning. The weights ω associated with the entire network that is obtained during the coarse training can be represented using concatenation notation as follows:

$$\omega = [\omega_1, \omega_2] \tag{4.1}$$

The frozen layers, representing the initial *N/2* layers, retain essential general features learned during coarse training. While the unfrozen layers, closer to the output, are fine-tuned to adapt to specific task objectives or requirements.

The weight update process in the SSF technique is shown by Eqs. (4.2 and 4.3).

$$\omega_1(p+1) = \omega_1(p) \tag{4.2}$$

$$\omega_2(p+1) = \omega_2(p) - \xi \cdot \nabla_{\omega_2(p)} L_{train}(\omega(p)) \tag{4.3}$$

Here, $\omega(p)$ represents the overall weights of the network, ξ is the learning rate parameter, and $\nabla_{\omega_2(p)} L_{train}(\omega(p))$ represents the gradient of the training loss function for updating the weights of unfrozen layers. $\omega_1(p)$ and $\omega_1(p+1)$ denote the weights of the frozen segment before and after fine-tuning, respectively. Similarly, $\omega_2(p)$ and $\omega_2(p+1)$ denote the weights of the unfrozen segment before and after fine-tuning, respectively. The illustration of simple selective freezing on an "*N*-layer" network is given in Fig. 4.3.

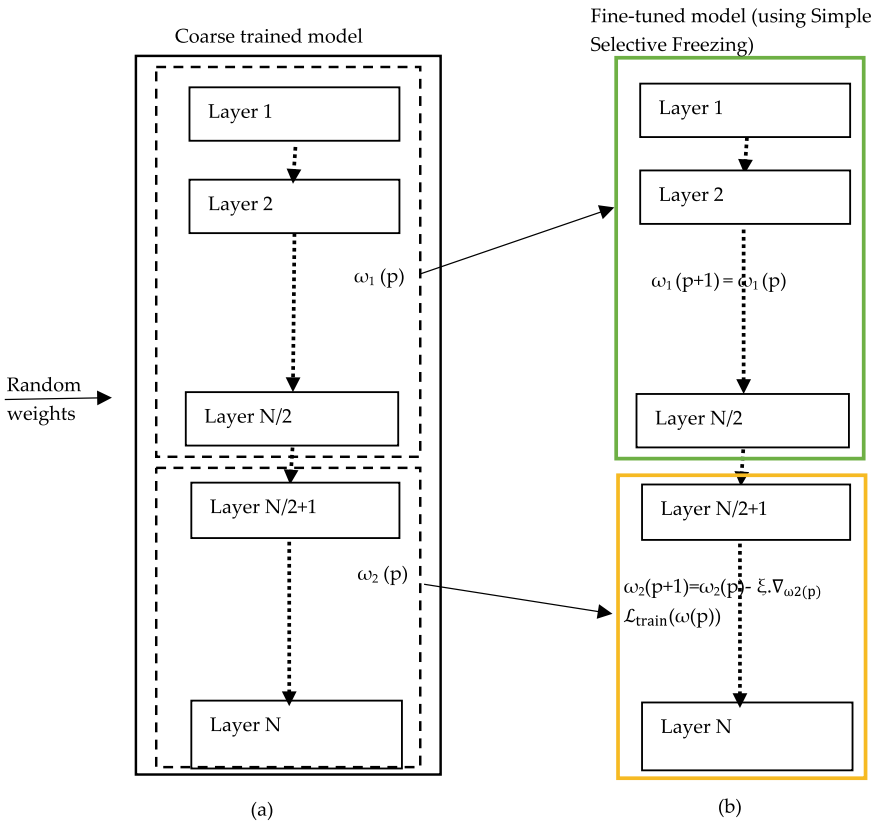


Fig. 4.3 Illustration of simple selective freezing on an "N-layer" network. **a** Depicts the intermediate model obtained after the coarse training phase with weights ω, which consists of two parts i.e., $\omega_1(p)$ and $\omega_2(p)$. **b** Shows the final model, after selective freezing, with the first half of the weights remaining unchanged (i.e., $\omega_1(p+1) = \omega_1(p)$) and the second part of weights is updated to $\omega_2(p+1)$. Note that the frozen part of the network is shown in the solid green block and the updating part is shown in the solid yellow block

The steps of training the target network with the SSF technique are summarized below in Algorithm 1.

Algorithm 1 Coarse Training Integrated with SSF Technique

I. *Consider a Target Network of N layers (or cells) to be trained.*

II. *Initialize the weight parameters (ω) with zero values and set the loss function to cross-entropy.*

III. *While not converged:* ***(Coarse training phase)***

Update weights ω of the network by descending the gradient of the training loss function($\mathcal{L}_{train}$) using the learning rate ξ

$$\omega(p+1) = \omega(p) - \xi.\nabla_{\omega(p)}\,\mathcal{L}_{train}(\omega(p))$$

IV. *Divide the network into two segments each consisting of N/2 layers (or cells), with weights ω_1 and ω_2 associated with the two segments.*

V. *Freeze the weights ω_1 associated with the first segment.*

VI. *While not converged:* ***(Fine tuning phase)***

Update weights ω_2 of the second segment by descending the gradient of the training loss function($\mathcal{L}_{train}$) using the learning rate ξ

$$\omega_2(p+1) = \omega_2(p) - \xi.\nabla_{\omega 2(p)}\,\mathcal{L}_{train}(\omega(p))$$

B. **Progression-Based Selective Freezing**

The Progression-based Selective Freezing (PSF) technique involves iteratively freezing different segments of the network and fine-tuning the remaining segments. It starts with the weights ω obtained from coarse training. These weights are divided into three sets ω_1, ω_2, and ω_3. The first "*l*" layers with weights ω_1 are unfrozen, the next "*m*" layers with weights ω_2 are frozen, and the remaining "*N-l-m*" layers with weights ω_3 are unfrozen. The unfrozen weights are updated during each iteration. The value of m remains the same (i.e., K) during all iterations, but the value of l changes as the location of the frozen segment moves down by m layers after each iteration. The value of l is set to 0 during the first iteration, its value progressively increases by m during each iteration and its value becomes N-$2K$ (*or* N-$2m$) during the last iteration.

The equations for selectively freezing and updating the weights of the different segments during $(p+1)$th iteration of PSF are given below by Eqs. (4.4, 4.5, and 4.6):

$$\omega_1(p+1) = \omega_1(p) - \xi \cdot \nabla_{\omega_1(p)} L_{train}(\omega(p)) \tag{4.4}$$

$$\omega_2(p+1) = \omega_2(p) \tag{4.5}$$

$$\omega_3(p+1) = \omega_3(p) - \xi \cdot \nabla_{\omega_3(p)} L_{train}(\omega(p)) \tag{4.6}$$

Here, $\omega(p)$ represents the overall weights of the network, ξ is the learning rate parameter, $\nabla_{\omega_1(p)} L_{train}(\omega(p))$ and $\nabla_{\omega_3(p)} L_{train}(\omega(p))$ represent the gradient of the training loss functions in the $(p+1)$th iteration. $\omega_1(p)$ and $\omega_1(p+1)$ denote the weights of the first (unfrozen) segment before and after fine-tuning, respectively. Similarly, $(\omega_2(p), \omega_2(p+1))$, and $(\omega_3(p), \omega_3(p+1))$ denote the weights of the second (frozen)

segment and third (unfrozen) segment before and after fine-tuning, respectively. Note that, in the first iteration, there are no layers in the first segment. In each successive iteration, the first segment grows, the second segment shifts downwards and the third segment shrinks. By freezing specific layers in each iteration, the training process is stabilized by preventing previously learned parameters (of the frozen segment) from excessively influencing the remaining layers. This balanced approach ensures that the unfrozen layers can adapt more effectively to the task at hand without being overwhelmed by the frozen ones. Figure 4.4 illustrates the PSF technique for an N-layer network.

The steps for training the target network with PSF are outlined below in Algorithm 2.

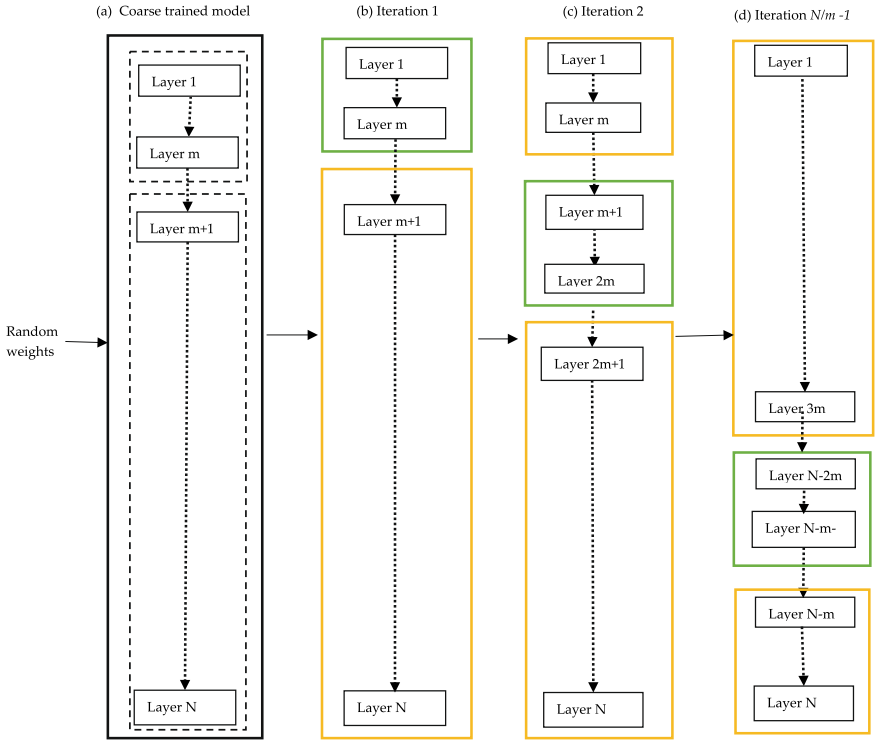


Fig. 4.4 Illustration of progression-based selective freezing. **a** The weights of the intermediate model obtained after the coarse training. **b** During the Ist iteration, the weights of the first $m(=K)$ layers are frozen while the weights of the remaining layers are updated. **c** During the 2nd iteration, the weights of the second set of m layers are frozen while the weights of the remaining layers are updated. **d** During last iteration, the weights of the last but one m layers are frozen while the weights of the remaining layers are updated. Note that frozen part of network is shown in solid green block and updated part is shown in solid yellow block

Algorithm 2 Coarse Training Integrated with PSF Technique

I. *Consider the target network of N layers (or cells) to be trained. Let m = K be the number of layers to be frozen during fine-tuning.*

II. *Initialize the weight parameters (ω) with zero values and set the loss function to cross-entropy.*

III. *While not converged:* ***(Coarse training phase)***

Update weights ω of the network by descending the gradient of the training loss function ($\mathcal{L}_{train}$) using the learning rate ξ

$$\omega(p+1) = \omega(p) - \xi.\nabla_{\omega(p)}\,\mathcal{L}_{train}(\omega(p))$$

IV. *Divide the network layers into three segments having 'l', 'm', and 'n' number of layers respectively, with ω_1, ω_2, and ω_3 representing the weights of each segment.*

V. ***Do:*** ***(Fine-tuning phase)***

l=0 for the first iteration, l = l+K otherwise, starting at 1st position
m = K starting at (l+1)th position,
n = (N-l-m). starting at (l+m+1)th position
Freeze the weights (ω_2) for the second segment of the network
While not converged
Update weights of the first segment (ω_1) and last segment (ω_3) by descending the gradient of the training loss function($\mathcal{L}_{train}$) using the learning rate ξ

$$\omega_1(p+1) = \omega_1(p) - \xi.\nabla_{\omega 1(p)}\,\mathcal{L}_{train}(\omega(p))$$

$$\omega_3(p+1) = \omega_3(p) - \xi.\nabla_{\omega 3(p)}\,\mathcal{L}_{train}(\omega(p))$$

While (l < N-2K)

The proposed approach is versatile and can be applied to various deep learning architectures. To illustrate its effectiveness, we specifically implement it in a 20-layer architecture derived using DARTS. Traditionally, neural networks are trained using standard training methods. To assess the new approach, the performance of the target network trained with both conventional methods and our proposed technique is compared. The results indicate that the new method consistently yields models with improved accuracy. The following section provides a detailed discussion of the application of the selective freezing training technique to the target architecture.

4.4 Experiments

The performance of the selective freezing training approach is evaluated in the context of image classification tasks. Several well-known image classification datasets, namely MNIST, CIFAR-10, and CIFAR-100 were used. An architecture search was conducted specifically on the CIFAR-10 dataset. The target architecture created was trained across all three datasets. The transferability feature of using one dataset for creating a target architecture and training it with the proposed approach across different datasets is also discussed.

4.4.1 Datasets

The CIFAR-10 is a widely used benchmark dataset in computer vision research, consisting of 60,000 color images in 10 different classes, with 6,000 images in every class. The dataset is divided into training and test sets, with 50,000 images for training

purposes and 10,000 images for testing. Each image in the dataset has a dimension of 32 × 32 pixels. The CIFAR-10 dataset is publicly accessible at (https://www.cs.toronto.edu/~kriz/CIFAR.html).

The CIFAR-100 is another widely used benchmark dataset consisting of 60,000 color images, similar to CIFAR-10, but is divided into 100 different classes instead of 10. Each class has 600 images, with 500 images in the training set and 100 images in the test set. The images are of size 32 × 32 pixels, just like in CIFAR-10. The CIFAR-100 dataset poses a more challenging classification task due to its increased number of classes, making it valuable for evaluating the robustness and generalization capabilities of machine learning models. The dataset is accessible at (https://www.cs.toronto.edu/~kriz/CIFAR.html).

The MNIST dataset is a well-known dataset used for image classification applications. It is a collection of grayscale handwritten digits. The dataset has 60,000 images for training and 10,000 images for testing.

4.4.2 Results and Discussion

The results of the new coarse training integrated with selective freezing techniques for a target network are discussed below and are compared with the results of the traditional training method.

A. Results of Coarse Training Integrated with SSF Technique

The impact of using different data proportions for training the target network using the new training method is explored. The target model is trained with 60%, 80%, and the entire training dataset for evaluating the new coarse training integrated with the SSF technique.

To gain a comprehensive understanding of the effectiveness of the new training approach, parallel experiments were conducted to compare it with the traditional training method. Both approaches were subjected to an equal number of epochs, corresponding to a given proportion of the dataset.

The results on CIFAR-10, presented in Table 4.1, clearly demonstrate the SSF technique's effectiveness in improving the target model's accuracy. When utilizing traditional training methods, the model attained accuracies of 85.12%, 89.09%, and 96.11% for 60%, 80%, and 100% of the training data, respectively. These results were achieved across 400 epochs for 60% of the data, 500 epochs for 80% of the data, and 600 epochs for the entire dataset. In contrast, the coarser training integrated with the SSF technique produced 94.10, 95.59, and 96.22% accuracy for the corresponding data proportions training over the same epoch counts.

Table 4.1 Performance comparison of coarse training integrated with SSF versus traditional training on CIFAR-10

Training method	Percentage of data used for training (%)	Parameters (Millions)			Accuracy (%)
		Parameters		Average parameters	
Traditional training	60	3.3		3.3	85.12
	80	3.3		3.3	89.09
	100	3.3		3.3	96.11
Coarse-to-fine-tuning with SSF		Coarse training (P1)	Fine-tuning training (P2)	[(P1 + P2)/ 2]	
	60	3.3	1.65	2.5	94.10
	80	3.3	1.65	2.5	95.59
	100	3.3	1.65	2.5	96.22

These findings highlight the improvements offered by the coarse training integrated with the SSF technique, demonstrating its potential as a valuable tool for deep learning model optimization.

B. Cross-Dataset Transferability Results of Coarse Training Integrated with the SSF technique

The learned cell obtained by employing the new coarse training integrated with the SSF technique on the CIFAR-10 dataset was used to evaluate its performance on CIFAR-100 and MNIST datasets, to check the versatility of the new approach. The results from the CIFAR-100 dataset are given in Table 4.2. With a training data proportion of 60%, the new method improved accuracy from 74.29 to 75.56% over a total of 400 epochs. At 80% of training data, accuracy with the new method was 79.16% compared to 77.54% using traditional training, spanning 500 epochs. With complete training data, accuracy reached 80.35% with the new training method, compared to 79.86% with the traditional training, across 600 epochs.

The experimental results for the MNIST dataset are shown in Table 4.3. Using the traditional training method, accuracies of 88.19, 94.09, and 94.99% were achieved for 60, 80, and 100% of the training data, respectively, with epochs set at 50, 60, and 70. In contrast, the new coarse training integrated with the SSF approach, involving 40, 50, and 60 epochs of coarse training followed by 10 epochs of fine-tuning for the respective data proportions, significantly surpassed traditional training results. The new method demonstrated enhanced accuracies of 99.51%, 99.55%, and 99.67%, respectively, for the same data proportions.

The results in Tables 4.2 and 4.3 indicate that the new method not only enhances performance compared to traditional training but is also highly transferable, where an architecture learned on one dataset can be trained on other datasets. For instance, when a network with cells learned on CIFAR-10 was trained using the new approach on CIFAR-100, the accuracy increased by about 1.7% and 2.1% with the 60% and

Table 4.2 Transferability performance comparison of coarse training integrated with SSF versus traditional training on CIFAR-100

Training method	Percentage of data used for training (%)	Parameters (Millions)			Accuracy (%)
		Parameters		Average parameters	
Traditional training	60	3.4		3.4	74.29
	80	3.4		3.4	77.54
	100	3.4		3.4	79.86
Coarse-to-fine-tuning with SSF		Coarse training (P1)	Fine-tuning training (P2)	[(P1 + P2)/ 2]	
	60	3.4	**1.7**	**2.6**	75.56
	80	3.4	**1.7**	**2.6**	79.16
	100	3.4	**1.7**	**2.6**	**80.35**

Table 4.3 Transferability performance comparison of coarse training integrated with SSF versus traditional training on CIFAR-100

Training method	Percentage of data used for training (%)	Parameters (Millions)			Accuracy (%)
		Parameters		Average parameters	
Traditional training	60	3.3		3.3	88.19
	80	3.3		3.3	94.09
	100	3.3		3.3	94.99
Coarse-to-fine-tuning with SSF		Coarse training (P1)	Fine-tuning training (P2)	[(P1 + P2)/ 2]	
	60	3.3	**1.65**	**≈2.5**	99.51
	80	3.3	**1.65**	**≈2.5**	99.55
	100	3.3	**1.65**	**≈2.5**	**99.67**

80% data proportions, respectively, compared to the traditional method. Similarly, on the MNIST dataset, the accuracy increased by around 12.9%, 5.3%, and 4.9% with the cell learned on CIFAR-10 using 60%, 80%, and 100% of the data, respectively. The accuracy gains across different proportions of these datasets confirm that a target architecture learned on one dataset can be effectively trained on other datasets, showing the robustness and versatility of the new training approach.

Moreover, the new training method demonstrates efficacy across varying data percentages, showing improvements in both portions of datasets and complete datasets. Particularly notable are its improvements in scenarios with limited data percentages, attributed to its selective freezing techniques designed to enhance generalization in data-constrained environments. This adaptability stems from its ability to extract maximum information from smaller datasets. Conversely, while the

method's performance gains may appear relatively smaller on complete datasets, this is primarily due to the inherent complexity of the task, where marginal improvements are harder to achieve regardless of the training method employed. Thus, the new method's versatility in optimizing model performance across different data scenarios underscores its potential for improving model performance for deep learning applications.

C. Results of Coarse Training Integrated with the PSF Technique

The coarse training integrated with the Progression-Based Selective Freezing (PSF) technique builds upon the SSF technique where freezing 50% of the weights yielded notable improvements. The PSF technique is applied to the already learned weights obtained during coarse training. The location of the frozen segment of the network progressively changes as training progresses. Three distinctive variants of PSF are examined, strategically freezing 1/4, 1/5, and 1/10th of the weights. The impact of different freezing proportions on the performance of the trained models, on the CIFAR-10 dataset, is examined.

Table 4.4 presents experimental results showing the impact of progressively freezing 1/4th of the weights on the performance of the trained model. In the initial iteration, where the first 1/4th of the weights (already learned during coarse training) were frozen, an accuracy of 96.11% was noted. The subsequent iterations indicated improved accuracies: 96.22%, and finally, 96.27%. The new approach involved two main phases: coarse training and fine-tuning. During coarse training, the model builds its foundational knowledge for 500 epochs. Following this, in each round of fine-tuning, the model improves its performance further for 30 epochs. Moreover, during fine-tuning only 2.5 million parameters are updated, in addition to the 3.3 million parameters updated during coarse training. The average number of parameters updated in the coarse and fine-tuning phases is approximately 2.9 million.

Table 4.5 illustrates the impact of freezing 1/5th of the weights and modifying the remaining ones, giving insights into improvement in accuracy and associated computational cost. In the initial iteration, where the first 1/5th of the weights (already

Table 4.4 Performance metrics of coarse training integrated with 1/4th weight freezing (Variant1) of PSF on CIFAR-10 dataset

Variant	Iteration	Parameters (Millions)			Accuracy (%)
		Coarse training parameters (P1)	Fine-tuning training parameters (P2)	Average parameter [(P1 + P2)/2]	
Coarse-to-fine-tuning with 1/4th weights frozen (Variant1) in PSF	1st 1/4th frozen	3.3	2.5	2.9	96.11
	2nd 1/4th frozen	3.3	2.5	2.9	96.22
	3rd 1/4th frozen	3.3	2.5	2.9	96.27

learned during initial training) is frozen, an accuracy of 96.25% is noted. Subsequent iterations displayed a consistent upward trend, reaching a value of 96.29%. Each iteration involved 30 epochs and the average parameter count updated is approximately 2.95 million.

Table 4.6 presents the accuracy results for the variant involving the freezing of 1/10th of the weights while modifying the remaining ones. As the initial 1/10th of the weights are frozen, a consistent improvement in accuracy is observed. Accuracy rates enhanced from 96.14% with the first set of weights frozen to 96.40% when the last set of weights was frozen. This gradual enhancement underscores the impact of the PSF technique in optimizing the trained model's performance. The number of epochs was set to 30 for each iteration of fine-tuning phase. These findings emphasize the effectiveness of this approach in achieving improved accuracy with the trained model.

The proposed concept of selective freezing offers a more comprehensive means of stabilizing and controlling the training process, which is essential for improving the performance of a trained model. This improvement exceeds what was achieved solely through traditional training. It stabilizes the overall model training, leading to enhanced performance in the given task or application. Thus, the new method's versatility in optimizing model performance across different data scenarios underscores its potential for improving model performance for deep learning applications.

Table 4.5 Performance metrics of coarse training integrated with 1/5th weight freezing (Variant2) of PSF on CIFAR-10 dataset

Variant	Iteration	Parameters (Millions)			Accuracy (%)
		Coarse training parameters (P1)	Fine-tuning training parameters (P2)	Average parameters [(P1 + P2)/2]	
Coarse-to-fine-tuning with 1/5th weights frozen (Variant2) in PSF	1st 1/5th frozen	3.3	2.6	2.95	96.25
	2nd 1/5th frozen	3.3	2.6	2.95	96.27
	3rd 1/5th frozen	3.3	2.6	2.95	96.27
	4th 1/5th frozen	3.3	2.6	2.95	96.29

Table 4.6 Performance metrics of coarse training integrated with 1/10th weight freezing (Variant 3) of PSF on CIFAR-10 dataset

Variant	Iteration	Parameters (Millions)			Accuracy (%)
		Coarse training parameters (P1)	Fine-tuning training parameters (P2)	Average parameters [(P1 + P2)/2]	
Coarse-to-fine-tuning with 1/10th weights frozen (Variant3) in PSF	1st 1/10th frozen	3.3	2.97	3.14	96.14
	2nd 1/10th frozen	3.3	2.97	3.14	96.21
	3rd 1/10th frozen	3.3	2.97	3.14	96.26
	4th 1/10th frozen	3.3	2.97	3.14	96.32
	5th 1/10th frozen	3.3	2.97	3.14	96.33
	6th 1/10th frozen	3.3	2.97	3.14	96.35
	7th 1/10th frozen	3.3	2.97	3.14	96.37
	8th 1/10th frozen	3.3	2.97	3.14	96.39
	9th 1/10th frozen	3.3	2.97	3.14	**96.40**

4.5 Future Directions

Future work can explore several key areas to enhance and validate the coarse-to-fine-tuning approach. One focus can be on integrating this approach with a wider variety of manually designed and automated architectures to assess its versatility and effectiveness across different network designs.

Evaluating the scalability of the approach for large models and datasets, as well as its computational demands, can be crucial for practical applications in resource-constrained environments. Finally, comparing the coarse-to-fine-tuning approach with other training techniques, such as weight-sharing methods, can provide insights into its relative strengths and potential areas for improvement. Through these efforts, one can refine and extend the approach, establishing its value and applicability in deep learning model training.

4.6 Summary

In summary, a novel training technique based on a coarse-to-fine-tuning approach was introduced, aimed at improving the accuracy of deep learning models. The approach begins with coarse training of the target model and then incorporates either of the two selective freezing techniques, SSF and PSF. These techniques offer flexible fine-tuning strategies aimed at refining the model developed during the coarse training phase. Utilizing these techniques, good improvements in accuracy compared to the traditional training method were achieved.

Bibliography

1. L. Li, A. Talwalkar, Random search and reproducibility for neural architecture search, in *Proceedings of PMLR* (2020), pp. 367–377
2. Z. Zhong et al., BlockQNN: efficient block-wise neural network architecture generation. IEEE Trans. Pattern Anal. Mach. Intell. **43**(7), 2314–2328 (2021). https://doi.org/10.1109/TPAMI.2020.2969193
3. B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, in *Proceedings of the 5th International Conference on Learning Representations (ICLR), Toulon, France, April 24–26* (2017)
4. B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, in *Proceedings of the 5th International Conference on Learning Representations (ICLR) 2017, Toulon, France, April 24–26* (2017)
5. Y. Gu, Y. Cheng, C.L.P. Chen, X. Wang, Proximal policy optimization with policy feedback. IEEE Trans. Syst. Man Cybern. Syst. **52**(7), 4600–4610 (2022). https://doi.org/10.1109/TSMC.2021.3098451
6. B. Zoph, G. Brain, V. Vasudevan, J. Shlens, Q.V. Le, G. Brain, Learning transferable architectures for scalable image recognition, in *Proceedings of the CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*) (2018), pp. 8697–8787. https://doi.org/10.1109/CVPR.2018.00907
7. E. Real, Large-scale evolution of image classifiers, in *Proceedings of the 34th International Conference on Machine Learning* (*ICML*), vol. 70 (2017), pp. 2902–2911
8. E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Aging evolution for image classifier architecture search, in *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), pp. 4780–4789
9. H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu, Hierarchical representations for efficient architecture search, in *International Conference on Learning Representations* (2018). http://arxiv.org/abs/1711.00436. (Online)
10. D.E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in *Foundations of Genetic Algorithms*, vol. 1 (1991), pp. 69–93. Elsevier. https://doi.org/10.1016/B978-0-08-050684-5.50008-2. ISSN 1081-6593, ISBN 9780080506845
11. T. Elsken, J. H. Metzen, F. Hutter, Efficient multiobjective neural architecture search via Lamarckian evolution, in *International Conference on Learning Representations* (2019). http://arxiv.org/abs/1804.09081. (Online)
12. S. Ali, M.A. Wani, Gradient-based neural architecture search: a comprehensive evaluation. Mach. Learn. Knowl. Extr. **5**(3), 1176–1194 (2023). https://doi.org/10.3390/make5030060
13. H. Liu, K. Simonyan, Y. Yang, DARTS: differentiable architecture search, in 7*th International Conference of the Learning Represent* (*ICLR*) (2019), pp. 1–13
14. E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33 (2019), pp. 4780–4789

15. X. Chen, L. Xie, J. Wu, Q. Tian, Progressive differentiable architecture search: bridging the depth gap between search and evaluation, in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 1294–1303. https://github.com/. (Online)
16. K. Nakai, T. Matsubara, K. Uehara, Att-DARTS: differentiable neural architecture search for attention, in *Proceedings of the IEEE International Joint Conference on Neural Networks* (*IJCNN*) (2020), pp. 1–8
17. G. Raskutti, M.J. Wainwright, Early stopping and non-parametric regression: an optimal data-dependent stopping rule. J. Mach. Learn. Res. **15**, 335–366 (2014)
18. W. Roth, F. Pernkopf, S. Member, Bayesian neural networks with weight sharing using Dirichlet processes. IEEE Trans. Pattern Anal. Mach. Intell. **42**(1), 246–252 (2020). https://doi.org/10.1109/TPAMI.2018.2884905
19. S. Ali, M. A. Wani, Recent trends in neural architecture search systems, in 2022 21*st IEEE International Conference on Machine Learning and Applications* (*ICMLA*)*, Nassau, Bahamas* (2022), pp. 1783–1790. https://doi.org/10.1109/ICMLA55696.2022.00272

Chapter 5
Generative Adversarial Networks in Image Steganography

5.1 Introduction

Generative Adversarial Networks (GANs) have gained considerable attention in image steganography mainly because these networks can encode and decode secret information using digital images efficiently. Leveraging the framework of GANs, researchers have devised novel techniques to embed and extract secret data seamlessly using images, offering a robust solution for secure communication and data concealment. This chapter discusses various GAN architectures proposed by researchers that have been used in image steganography.

5.2 Vanilla Generative Adversarial Networks

Vanilla GAN, also known as the original GAN model, serves as the basic network on which subsequent GAN architectures are built. Vanilla GAN has a straightforward architecture consisting of a generator and a discriminator network that engages in a competitive training process. The generator aims to produce realistic synthetic data samples, while the discriminator strives to differentiate between real and synthetic samples. Vanilla GAN is simple and easy to implement, making it an ideal starting point for exploring its use in various applications. Despite its simplicity, Vanilla GAN demonstrates remarkable capabilities in generating diverse and realistic data distributions, laying the groundwork for advancing its application in various domains, including image generation and steganography.

The block diagram of the vanilla GAN is given in Fig. 5.1. Multilayer Perceptron (MLP) is commonly used as the basic building block for both the generator and discriminator networks in Vanilla GAN. MLPs are the simplest form of neural networks, consisting of multiple layers of neurons with fully connected connections between each layer.

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_5

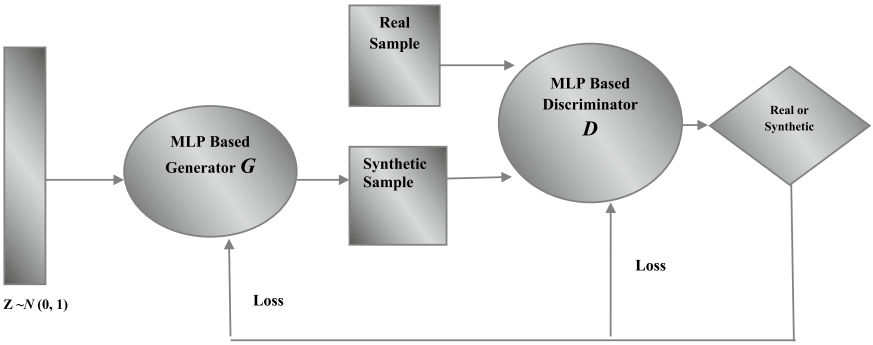


Fig. 5.1 Vanilla GAN framework

One analogy characterizes GAN as a thief (the Generator) who forges currency and a detective (the Discriminator) who tries to apprehend him. The detective must become more efficient at spotting fake currency as it becomes more realistic-looking, and vice versa.

The word "generative" refers to the model's primary goal of producing new synthetic data. Depending on the training dataset used, a GAN can learn to make different types of synthetic data.

The word "adversarial" refers to the competitive, game-like dynamics between the Generator and Discriminator, the two models that make up the GAN framework. The two networks are constantly competing to outsmart one another: the Discriminator becomes more effective in differentiating real samples from synthetic ones, while the Generator becomes more efficient in producing synthetic data similar to the real data.

5.2.1 Generator Network

The architecture of the generator network is given in Fig. 5.2. The generator uses a random noise, usually represented as a simple N-dimensional Gaussian random vector, to produce synthetic data that closely resembles the distribution of real data. To achieve this, the generator typically comprises a series of fully connected layers, each followed by activation functions. As the noise data moves through these layers, it undergoes a process of up-sampling, gradually transforming it into high-resolution output images.

In essence, the generator functions as a network of interconnected layers, with each layer playing a specific role in shaping the output. This network architecture ensures that the generated synthetic data aligns with the desired characteristics of the real data distribution. Once the data passes through all the layers, it reaches the output layer, where necessary adjustments are made to ensure compatibility with the discriminator network. Ultimately, the goal of the generator is to produce synthetic

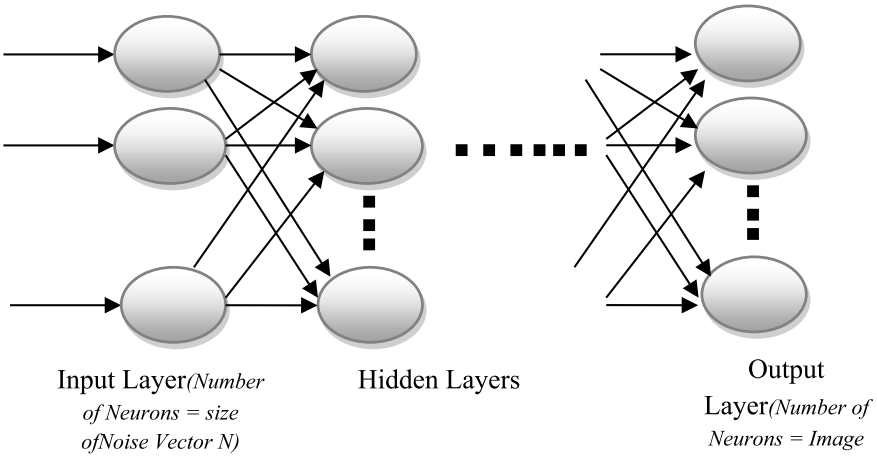


Fig. 5.2 Generator model

samples that can effectively deceive the discriminator into not differentiating between the real data and the synthetic data.

5.2.2 Discriminator Network

The discriminator typically consists of a series of fully connected layers, each followed by activation functions, enabling it to extract relevant features from the input images (see Fig. 5.3).

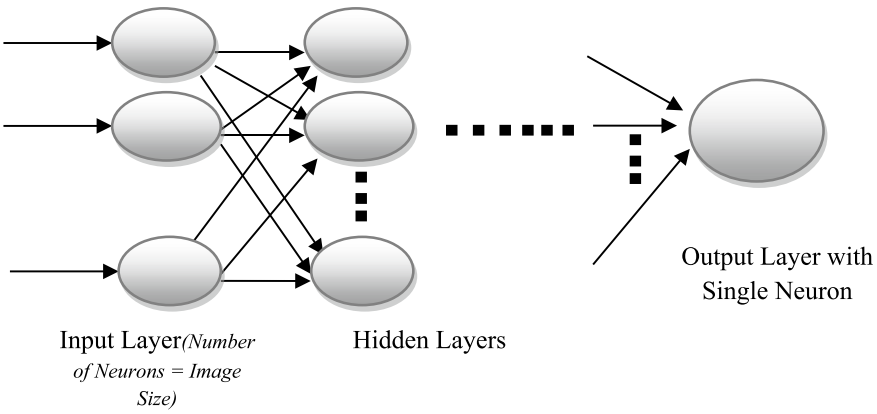


Fig. 5.3 Discriminator model

When processing data, the discriminator network analyzes both real and synthetic samples, extracting features that differentiate between them. These features are then used to make binary classification decisions, determining whether each input sample is real or synthetic. As the data moves through the network, it undergoes a process of down-sampling, gradually condensing the information to a binary classification output.

The discriminator operates as a network of interconnected layers, with each layer contributing to the discrimination process. The network architecture is designed to effectively distinguish between real and synthetic data samples, providing feedback to the generator for improving its ability to generate realistic data.

5.2.3 Loss Function

The loss function for Vanilla GAN involves two components.

5.2.3.1 Discriminator Loss

The discriminator minimizes the difference between the probability distributions of real and synthetic samples. This is typically achieved using binary cross-entropy loss, where the discriminator endeavors to maximize the probability of correctly classifying real and synthetic samples.

5.2.3.2 Generator Loss

Conversely, the generator seeks to maximize the probability of the discriminator incorrectly classifying its generated samples as real. Thus, it aims to minimize the discriminator's ability to distinguish between real and synthetic samples.

Formally, the Vanilla GAN loss function can be represented as

$$\frac{Min}{G}\frac{Max}{D}V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}\big[\log(1 - D(G(z)))\big] \quad (5.1)$$

In this equation, G represents the generator network, D represents the discriminator network, x denotes the real data samples, z denotes the random noise input to the generator, $p_{data}(x)$ represents the distribution of real data, and $p_z(z)$ represents the distribution of the noise vector z. The first term corresponds to the discriminator's loss, while the second term corresponds to the generator's loss.

This loss function makes the generator and discriminator compete with each other, leading to the convergence of both networks towards an equilibrium where the generator produces realistic samples and the discriminator struggles to differentiate between real and synthetic samples effectively.

5.2.4 GAN Training

The training algorithm of GAN is given below.

Training Algorithm of GAN

For each training iteration do

1. ***Training the Discriminator:***
    1. *Select a Random sample x from the Training Dataset.*
    2. *Input a Random Noise vector z to the generator network. The generator produces a synthetic sample x*.*
    3. *Pass x and x* to the discriminator to predict real and synthetic sample*
    4. *Calculate the classification errors and back-propagate the total error to update the trainable parameters (θ_d) of the discriminator, aiming to reduce the classification errors.*
2. ***Training the Generator:***
    1. *Input a Random Noise vector z to the generator network. The generator produces a synthetic sample x*.*
    2. *Pass x* to the discriminator to classify x*.*
    3. *Calculate the classification error, back-propagate the error, and update the trainable parameters for the generator to increase the discriminator's error.*

5.3 Deep Convolutional Generative Adversarial Networks

Deep Convolutional Generative Adversarial Networks (DCGANs) represent a pivotal advancement in the field of generative models. Figure 5.4 depicts the basic framework of DCGAN. It consists of CNN-based generator and discriminator networks.

The DCGAN uses convolution operations with integer strides in the discriminator and convolution operations with fractional strides in the generator. The convolution operations with stride size 2 or more reduce the spatial dimensions of the feature maps. Similarly, convolution operations with fractional strides up-sample the input feature maps. By utilizing convolutions with integer strides and fractional strides,

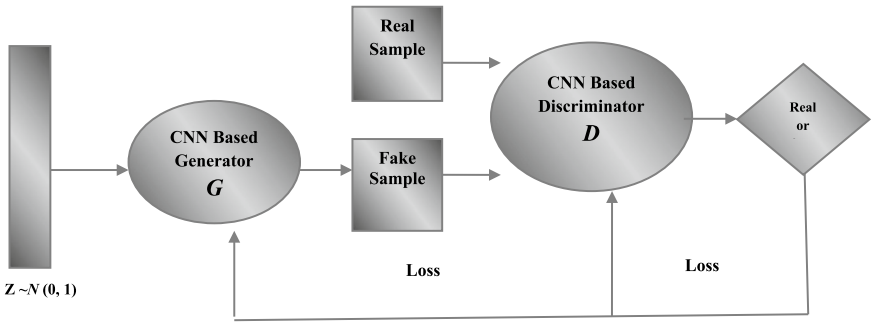


Fig. 5.4 DCGAN framework

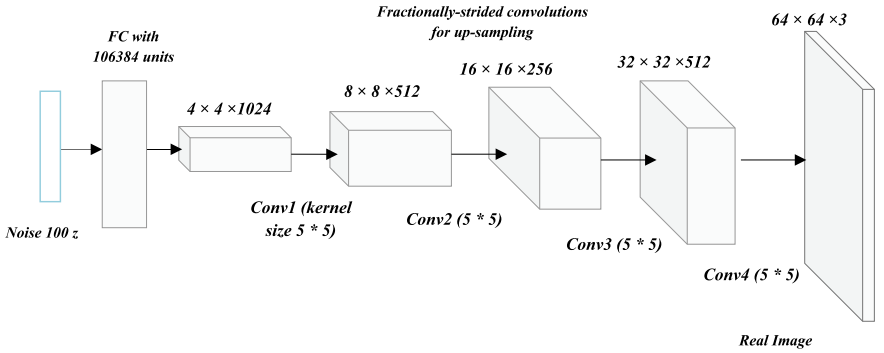


Fig. 5.5 DCGAN generator

DCGAN ensures the transmission of crucial information across layers, promoting the production of coherent and detailed synthetic samples.

5.3.1 Generator Network

The DCGAN generator network is depicted in Fig. 5.5. The initial phase involves initializing a random noise vector, typically denoted as z, with dimensions [100, 1]. This vector is then passed through a fully connected layer, yielding a vector of random numbers with dimensions [16384, 1], achieved through linear projection. Subsequently, in the reshaping step, the outputs from the previous phase are transformed into a feature map of size $4 \times 4 \times 1024$, which is fed into the network's first convolution layer using fractional strides.

The generator network comprises four convolution layers that use fractional strides, each designed to up-sample the input feature maps, where feature maps progress from size 4–8 to 16–32. The architectural choices ensure that the generator progressively refines the random noise input, generating increasingly detailed and coherent images as it traverses through the network layers.

5.3.2 Discriminator Network

The DCGAN discriminator network functions as a binary classification network, tasked with discriminating between real and synthetic images. It operates by receiving a $3 \times 64 \times 64$ input image and processing it through a series of convolution layers (included with batch normalization and Leaky ReLU activations) as shown in Fig. 5.6. These layers are crucial for extracting meaningful feature maps from the input image, and these feature maps undergo down-sampling as they progress through the network.

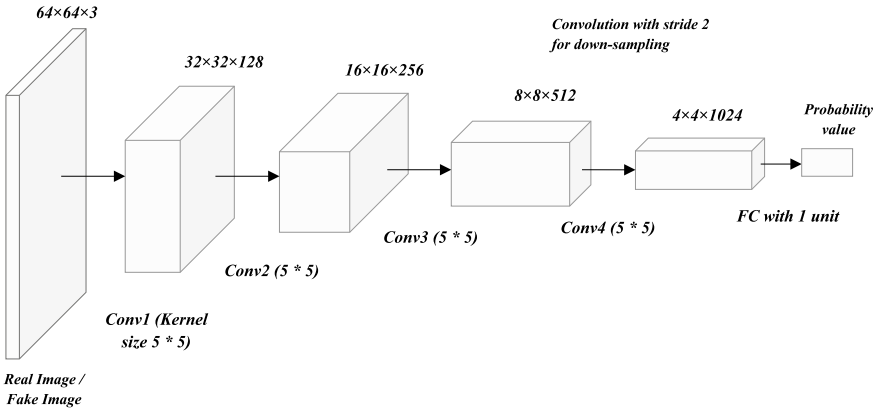


Fig. 5.6 DCGAN discriminator

The discriminator's objective is to produce a scalar probability indicating the likelihood that the input image is real rather than synthetic. This probability is obtained by applying a sigmoid activation function to the output of the network.

5.4 Wasserstein Generative Adversarial Networks (WGAN)

The Wasserstein Generative Adversarial Network (WGAN) uses a critic instead of estimating the probability in the discriminator to determine if the generated samples are real or synthetic. This critic, as depicted in Fig. 5.7, utilizes the Wasserstein distance metric to evaluate the quality of generated samples in a more nuanced manner and provides gradient signals that are more informative and stable compared to the binary classification task of traditional discriminators.

By minimizing the Wasserstein distance between the distribution of generated samples and the distribution of real data, WGAN aims to overcome limitations such as mode collapse and instability commonly observed in traditional GAN training. This leads to more stable training and facilitates the generation of diverse and high-quality samples across different modes of data distribution.

Furthermore, WGAN training is characterized by its robustness to variations in model architecture and hyperparameter settings. Unlike traditional GANs, which are highly sensitive to factors like network depth, activation functions, and learning rates, WGANs exhibit greater resilience, making them more suitable for practical applications where fine-tuning complex architectures can be challenging.

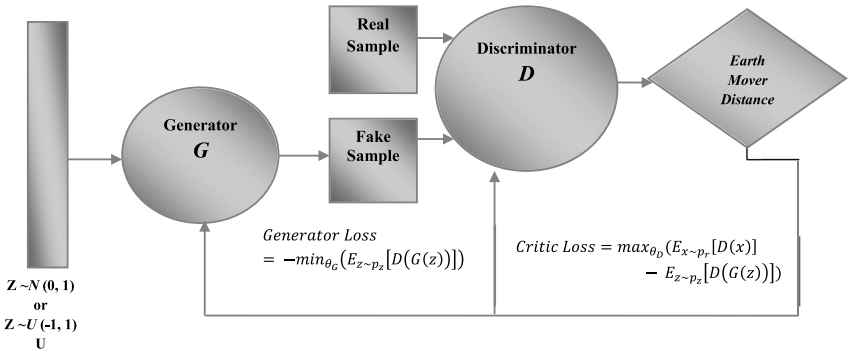


Fig. 5.7 WGAN

5.4.1 Wasserstein Distance

The loss function used in WGANs is based on the Wasserstein distance, also known as the Earth Mover's Distance (EMD). Unlike traditional GANs, which use the binary cross-entropy loss to train the discriminator and generator, WGANs optimize the Wasserstein distance between the distributions of real and generated data. Wasserstein distance is directly tied to the fidelity of generated samples to the true data distribution.

The Wasserstein distance measures the minimum amount of work (or "distance") required to transform one probability distribution into another. In the context of WGANs, this distance is computed between the distribution of real data (x) and the distribution of generated data (z). The generator aims to minimize this distance, effectively bringing the generated distribution closer to the real data distribution.

5.4.2 Discriminator Loss Function

The loss function aims to maximize the difference between the critic's output scores for real and generated samples. This difference quantifies how well the critic can distinguish between real and synthetic data. The discriminator loss function in WGANs is expressed as

$$\textit{Wassertein Critic Loss} = \max_{\theta_D}\left(E_{x\sim p_r}[D(x)] - E_{z\sim p_z}[D(G(z))]\right) \tag{5.2}$$

5.4.3 Generator Loss

The generator loss function involves minimizing the Wasserstein distance but with a flipped sign. It aims to minimize the discrepancy between the critic's scores for real and generated samples. The generator loss function in WGANs is expressed as

$$Generator\ Loss = -\min_{\theta_G}\left(E_{z\sim p_z}[D(G(z))]\right) \tag{5.3}$$

This loss function drives the training of WGANs, leading to the refinement of both the generator and discriminator networks over successive iterations.

5.5 Auxiliary Classifier Generative Adversarial Networks (ACGAN)

Auxiliary Classifier Generative Adversarial Networks (ACGANs) represent an extension of traditional GANs, incorporating additional conditioning information to guide the generation process. In ACGANs, both the generator and discriminator networks are conditioned on auxiliary information, such as class labels or other attributes associated with the data. This conditioning enables more fine-grained control over the generated samples, allowing users to specify desired attributes or characteristics of the generated outputs.

5.5.1 ACGAN Architecture

The architecture of an ACGAN consists of two main components: the generator and the discriminator. As illustrated in Fig. 5.8, the generator takes random noise (z) as input, along with the desired conditioning information (c), and learns to generate samples that not only resemble real data but also exhibit the specified attributes. This is typically achieved through a series of convolutional and up-sampling layers, where the noise vectors and auxiliary information are combined and transformed into high-dimensional representations, ultimately producing realistic images.

The discriminator in an ACGAN serves a dual role. In addition to discriminating between real and generated samples, it is also responsible for predicting the auxiliary information associated with the samples. The discriminator receives both real and generated samples, along with their respective auxiliary conditioning information, and learns to classify them into their corresponding classes or attribute categories. This dual-task architecture encourages the discriminator to learn a more informative representation of the data, enabling it to accurately predict the auxiliary information while effectively distinguishing between real and synthetic samples.

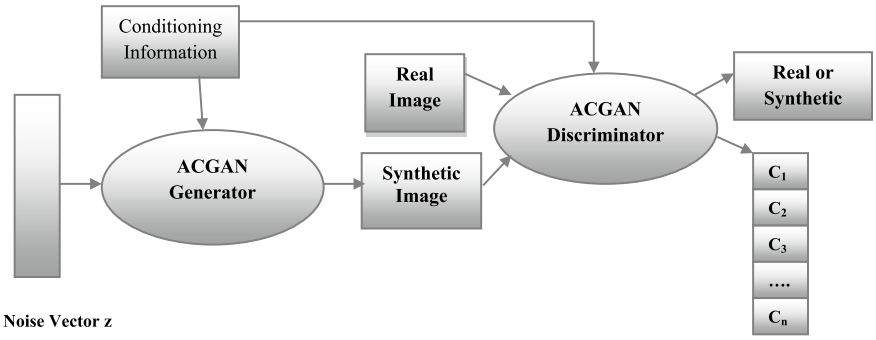


Fig. 5.8 Workflow of ACGAN

Overall, ACGANs offer a powerful framework for conditional image generation, allowing for the generation of diverse and controllable outputs across various domains. By leveraging conditioning information, ACGANs enable users to specify desired attributes or characteristics of the generated samples, making them particularly useful for tasks such as image synthesis, and data augmentation.

5.5.2 Generator Architecture

The ACGAN generator is a neural network designed to create realistic images conditioned on class labels. Its architecture typically involves a sequence of layers that progressively upscale a lower-dimensional noise vector, concatenated with an embedded class label, into a high-dimensional image.

Starting with a fully connected layer that reshapes the input into an initial low-resolution feature map, the network then applies several transposed convolutional layers (or deconvolutions) interspersed with batch normalization and ReLU activations (Fig. 5.9). This series of up-sampling steps incrementally increases the spatial resolution while refining the generated image details. The final layer employs a tanh activation function to produce the output image, ensuring pixel values are within the desired range, typically $[-1, 1]$. The incorporation of class labels at multiple stages allows the generator to produce images that align with the specified class, enhancing the diversity and quality of the generated outputs.

5.5.3 Discriminator Architecture

The ACGAN discriminator network is designed to evaluate the authenticity and class of input images, as depicted in Fig. 5.10. The network comprises several convolutional layers that progressively down-sample the input image, extracting hierarchical

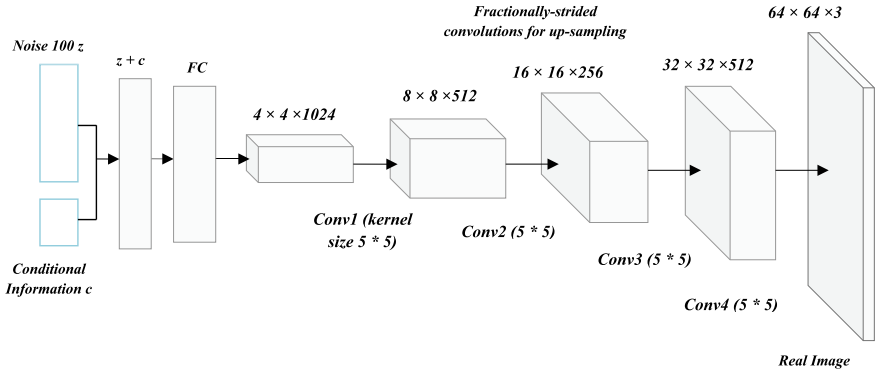


Fig. 5.9 Architecture of ACGAN generator

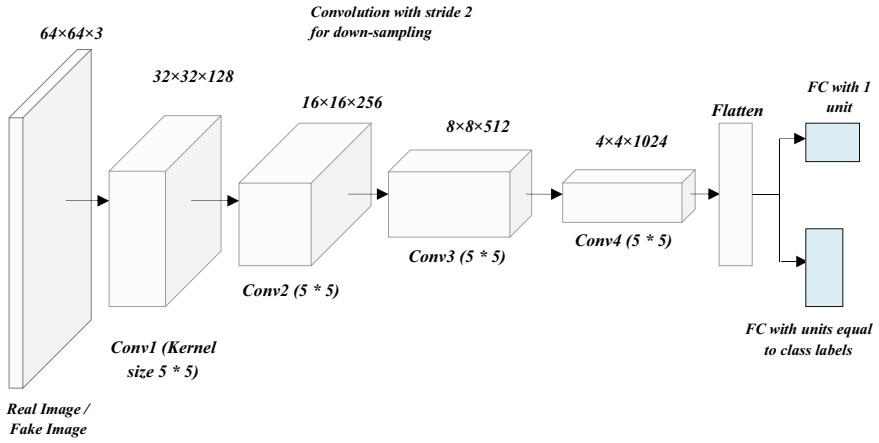


Fig. 5.10 Architecture of ACGAN discriminator

features. Each convolutional layer is typically followed by batch normalization and leaky ReLU activation to enhance training stability and convergence. The discriminator has two output branches: one for real/synthetic image classification and another for predicting the class label. This dual-output structure enables the discriminator to simultaneously perform adversarial training and supervised classification, thereby guiding the generator to produce high-quality images that are both realistic and class-consistent.

5.5.4 Loss Function

The ACGAN loss functions for both the generator and discriminator are tailored to optimize two main objectives: generating realistic images and predicting auxiliary information, such as class labels or attributes, associated with the samples. These loss functions play a crucial role in guiding the training process and ensuring that the generator produces high-quality images aligned with the specified attributes.

5.5.4.1 Generator Loss Function

The ACGAN generator loss function is composed of two components: the adversarial loss and the auxiliary classification loss. The adversarial loss, similar to traditional GANs, encourages the generator to produce samples that are indistinguishable from real data, as perceived by the discriminator. Formally, the generator's loss function can be expressed as

$$\textit{Generator Adversarial Loss} = -E_{z\sim p_z}\left[\log D(G(z, c))\right] \tag{5.4}$$

The generator aims to minimize this loss, effectively fooling the discriminator into believing that the generated samples are real.

The auxiliary classification loss encourages the generator to produce samples that match the specified attributes or classes. Formally, the auxiliary classification loss for the generator can be expressed as

$$\textit{Generator Auxiliary Loss} = -E_{z\sim p_z, c\sim p_c}\left[\log Q((c|G(z, c)))\right] \tag{5.5}$$

It predicts the auxiliary information c for the given generated samples. The generator seeks to minimize this loss, ensuring that the generated samples are aligned with the specified attributes.

5.5.4.2 Discriminator Loss Functions

The ACGAN loss function also consists of two components: the adversarial loss and the auxiliary classification loss. The adversarial loss guides the discriminator to effectively distinguish between real and generated samples, while the auxiliary classification loss encourages the discriminator to accurately predict the auxiliary information associated with the samples.

Formally, the discriminator's loss function can be expressed as

$$\begin{aligned}\textit{Discriminator Adversarial Loss} = &- E_{x\sim p_{data}(x)}\left[\log D(x, c)\right] \\ &- E_{z\sim p_z}\left[\log(1 - D(G(z, c)))\right]\end{aligned} \tag{5.6}$$

$$\begin{aligned} \textit{Discriminator Auxiliary Loss} = & - E_{x\sim p_{data}(x),c\sim p(c)}\big[\log Q((c|x))\big] \\ & - E_{z\sim p(z),c\sim p(c)}\big[\log(1 - Q((c|G(z,c))))\big] \end{aligned} \quad (5.7)$$

The discriminator aims to minimize the adversarial loss while maximizing the auxiliary classification loss.

By jointly optimizing these loss functions, ACGAN facilitates the generation of diverse and controllable outputs while ensuring alignment with specified attributes or characteristics. The adversarial and auxiliary classification components of the loss functions guide the training process, leading to the synthesis of high-quality images that not only resemble real data but also exhibit the desired attributes or classes.

5.6 Challenges and Future Direction

Generative Adversarial Networks (GANs) have demonstrated significant advancements in the generation of realistic data, spanning images, audio, and text. However, alongside these strides, they present a set of challenges and opportunities for future research.

One notable challenge is mode collapse, a phenomenon where the generator of a GAN learns to generate only a limited subset of outputs, failing to encompass the entire target distribution. Additionally, GANs are notorious for their training instability, characterized by issues such as oscillations, vanishing gradients, and divergence, making them challenging to train effectively.

Another challenge lies in evaluating the performance of GANs. While metrics like Inception Score and Fréchet Inception Distance (FID) have been proposed, they may not always align with human judgment, adding complexity to assessing GAN-generated outputs. Moreover, the generation of high-resolution images poses a persistent challenge due to constraints in memory and computational resources.

Furthermore, GANs often lack interpretability, hindering the understanding of why they produce specific outputs. This limitation underscores the need for improved methods to interpret and comprehend the generated samples, enhancing the utility and trustworthiness of GAN-generated data.

Addressing these challenges presents avenues for future research, including refining training techniques to mitigate instability, devising more robust evaluation metrics, overcoming limitations in generating high-resolution content, and enhancing interpretability to foster a deeper understanding of GAN-generated data. Through concerted efforts in these areas, GANs can realize their full potential in generating diverse, realistic data across various domains.

5.7 Summary

This chapter focused on GAN, the prevailing deep learning architecture widely employed in several applications such as medical, remote sensing, computer vision, natural language processing, and steganography. It delved into the discussion of various Generative Adversarial Network (GAN) models, including Vanilla GAN, Deep Convolutional GAN (DCGAN), Wasserstein GAN (WGAN), and Auxiliary Classifier GAN (ACGAN). As researchers continue to refine GAN-based approaches to image steganography, the potential for enhancing data security and privacy remains promising, paving the way for new advancements and applications in the field.

Bibliography

1. A. Radford, L. Metz, Unsupervised representation learning with deep convolutional generative adversarial networks (2016). https://arxiv.org/abs/1511.06434v2
2. M. Mirza, S. Osindero, Conditional generative adversarial nets (2014). https://arxiv.org/abs/1411.1784v1
3. A. Odena, Semi-supervised learning with generative adversarial networks (2016). https://arxiv.org/abs/1606.01583v2
4. D. Hu, L. Wang, W. Jiang, S. Zheng, B. Li, A Novel image stegnography method via deep convolutional generative adversarial networks, in *IEEE Transactions on Information Forensics and Security*
5. I.J. Goodfellow et al., Generative adversarial nets (2014). https://arxiv.org/abs/1406.2661v1
6. D.-Y. Lin et al., Synthesizing remote sensing images by conditional adversarial networks, in *IGARSS* (2017). IEEE. 978-1-5090-4951-6/17/$31.00 ©2017
7. Jaiwen et al., *An End-to-End Generative Adversarial Network for Crowd Counting Under Complicated Scenes*
8. M. Arjovsky, S. Chintala, L. Bottou, Wasserstein GAN (2017). http://arxiv.org/abs/1701.07875. (Online)
9. M.M. Liu, M.Q. Zhang, J. Liu, P.X. Gao, Y.N. Zhang, Coverless information hiding based on generative adversarial networks. Yingyong Kexue Xuebao/J. Appl. Sci. **36**(2), 371–382 (2018). https://doi.org/10.3969/j.issn.0255-8297.2018.02.015
10. B. Sultan, M.A. Wani, A new framework for analyzing color models with generative adversarial networks for improved steganography. Multimed. Tools Appl. (2023). https://doi.org/10.1007/s11042-023-14348-7
11. M.A. Wani, B. Sultan, Deep learning based image steganography: a review. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 1–26 (2022). https://doi.org/10.1002/widm.1481

Chapter 6
Deep Learning Architectures for Image Steganography

6.1 Introduction

The two information hiding techniques of cryptography and steganography are widely used for communicating secret information securely. Cryptography uses a key and an encryption algorithm to convert the secret information into a form called cipher data that is difficult to comprehend. This cipher data is converted back to the original format at the receiver end by using a decryption algorithm. Another way of hiding secret information securely is through steganography, which embeds the secret data in a cover medium. Thus, in cryptography, it is known that communication of secret information is taking place whereas in steganography it is not known if a cover medium has secret information embedded in it or not.

The secret information can be text, an image, audio, or video. Similarly, the cover medium can be text, an image, audio, or video. The general layout of a typical steganography process is depicted in Fig. 6.1.

The steganographic encoder receives the cover object C (image, text, audio, or video) and the secret message m (image, text, audio, or video) to produce a stego object (S) using key k. The steganographic encoder applies the encoding algorithm or function (f) onto the cover C and m to generate a stego object.

$$S = f(C, m, k)$$

This stego object is transmitted through a communication channel to the receiver, which acts as a steganographic decoder and extracts decoded secret message (m') from the stego object using key k. The steganographic decoder applies the decoding algorithm or function (f') onto the stego object S to obtain the decoded secret message m'. The decoder is said to be successful if it can extract m' such that $m = m'$.

$$m' = f'(S, k)$$

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_6

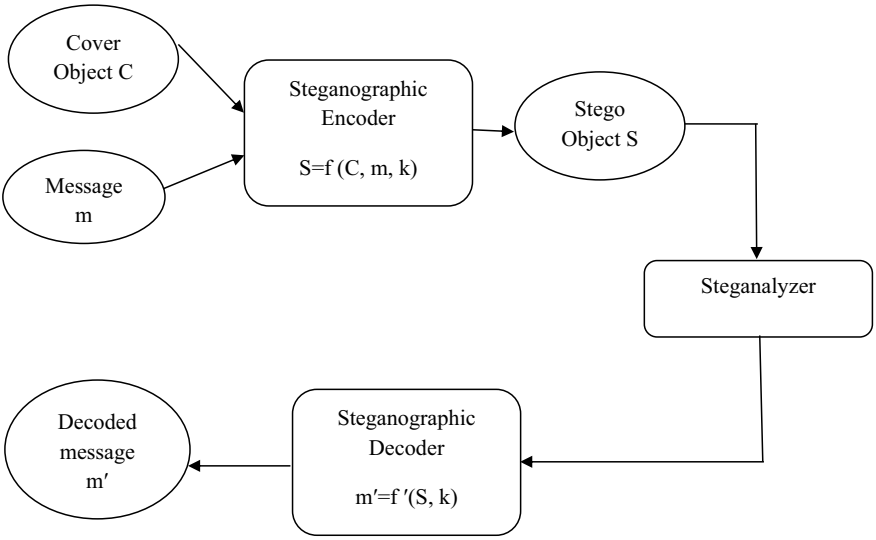


Fig. 6.1 General layout of a typical steganography process

During transmission of the stego object from the encoder to the decoder, the steganalyzer analyses the stego object. This process of intrusion into the communication channel to find out whether the secret communication is taking place by the intruder is known as steganalysis. The security of a steganographic system can be increased by making the probability distribution $P_{C'}$ of the stego object similar to the probability distribution P_C of the cover object. Thus

$$D(P_{C'}, P_C) < \in$$

where D is the distance measure.

Section 6.2 of this chapter describes evaluation criteria and metrics for image steganographic techniques. Deep Learning based steganography techniques are described in Sect. 6.3. Performance comparison is discussed in Sect. 6.4. Section 6.5 presents future research directions. The conclusion is finally presented in Sect. 6.6.

6.2 Evaluation of Image Steganographic Techniques

Selecting an appropriate evaluation mechanism for analyzing how well a steganographic technique is performing is important. This section describes a mechanism that is used to evaluate steganographic techniques.

6.2.1 Criteria for Evaluating Steganography Techniques

The quality of a steganographic technique is characterized by three key parameters: security, hiding capacity, and invisibility (or distortion measure). The steganographic techniques reviewed here are evaluated using these parameters. A definition and metric of these parameters are presented below.

(a) **Security**: Security measures resistance offered by a steganography technique to steganalysis test. A steganographic technique is said to be secure if it can hide the secret data inside the image in such a way that steganalysis is not able to detect the presence of secret data inside it. Some popular steganalysis tools are Xu's Net (Xu et al. 2016), Ye's Net (Ye et al. 2017) SRM (Fridrich and Kodovsky 2012), and ATS (Lerch-Hostalot and Megías 2016).
(b) **Hiding Capacity**: Hiding capacity measures the amount of secret information that can be embedded inside an image. With the increase in embedding capacity, the stego image quality decreases; hence the two are inversely proportional to each other.
(c) **Invisibility**: Invisibility measures similarity between the cover image and the stego image. The goal of steganographic techniques is to embed the secret data such that it results in stego images that are indistinguishable from the cover images, i.e., embedded data is invisible.

6.2.2 Evaluation Metrics

The metrics of the three parameters: (i) Security, (ii) Hiding Capacity, and (iii) Invisibility for evaluating the performance of a steganographic technique are given below.

(a) **Security**: Steganalysis test determines whether a given image is embedded with secret data or not; it labels a given image as a cover or stego. A high error rate implies a secure steganographic technique and is computed using the following: number of true positives (TP), number of false negatives (FN), number of true negatives (TN), and number of false positives (FP). The accuracy and error rate are defined below:

$$Accuracy = (TP + TN)/(TP + FP + FN + TN)$$
$$Error\ Rate = (1 - Accuracy) * 100$$

(b) **Hiding Capacity**: Hiding capacity is the number of secret data bits that can be effectively embedded inside the cover image. It is computed as the ratio of maximum hiding capacity to the image size.

$$\text{Relative Capacity} = \frac{\text{Absolute Capacity}}{\text{Image Size}}$$

Maximum hiding capacity or absolute capacity gives the number of secret data bits hidden inside an image and relative capacity gives the maximum number of bits hidden per pixel. Relative capacity is also referred to as bit rate, and is measured in bits per pixel (bpp) or bytes per pixel (bpp).

(c) **Extraction Accuracy**: Extraction accuracy is obtained by dividing the number of secret bits that are accurately decoded by the extractor model to the number of secret bits concealed inside the cover image. With more steganography capacity, the extractor network's performance declines.

$$Extraction\ Accuracy = \frac{No.of\ \ bits\ successfully\ decoded}{No.of\ \ bits\ Embedded}$$

(d) **Invisibility (or distortion measurement)**: Various metrics can be used to measure the distortion introduced in a stego image as a result of secret information embedding. These include Mean Square Error (MSE), Root Mean Square Error (RMSE), Correlation, Quality Index, Peak Signal-to-Noise Ratio (PSNR), Weighted PSNR (WPSNR), Structural SIMilarity (SSIM), Manhattan Distance, Euclidean Distance and Kullback–Leibler Divergence (K-L divergence). PSNR and SSIM are the two commonly used metrics for distortion measurement in steganography. A high value of PSNR and SSIM denote better image quality and lesser distortion and vice versa.

PSNR is defined as the ratio of the maximum power to the power of noise that influences the quality of an image

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{(\mathrm{Max_I})^2}{\mathrm{MSE}}\right)$$

Here $\mathrm{Max_I}$ indicates the maximum possible intensity levels (pixel values) in an image, and MSE is the mean square error that is given as:

$$\mathrm{MSE} = \frac{1}{\mathrm{mn}}\sum_{\mathrm{i}=0}^{\mathrm{m}-1}\sum_{\mathrm{j}=0}^{\mathrm{n}-1}\left(\mathrm{C(i,j)} - \mathrm{C'(i,j)}\right)^2$$

where C(i, j) represents the cover image pixel and C′(i, j) represents the stego image pixel.

m and n represent the number of rows and columns respectively in the images.

SSIM measures the similarity between two images and is given as:

$$\mathrm{SSIM}\left(\mathrm{C}, \mathrm{C'}\right) = \frac{(2\mu_{\mathrm{c}}\mu_{\mathrm{c'}} + \mathrm{C1})(2\sigma_{\mathrm{cc'}} + \mathrm{C2})}{(\mu_{\mathrm{c}}^2 + \mu_{\mathrm{c'}}^2 + \mathrm{C1})\left(\sigma_{\mathrm{c}}^2 + \sigma_{\mathrm{c'}}^2 + \mathrm{C2}\right)}$$

Here μ_c denotes the mean intensity of cover image (C), and $\mu_{c'}$ denotes the mean intensity of the stego image (C′); σ_c^2 and $\sigma_{c'}^2$ represent the variance of C and C′ respectively; $\sigma_{cc'}$ gives the covariance between C and C′; C1 and C2 are two parameters for stabilizing weak denominator divisions and are given by $C1 = (k_1\ L)^2$ and $C2 = (k_2\ L)^2$, L indicates the dynamic range of pixels values (typically this is 2 # bits per pixel-1) and k_1 and k_2 are set to 0.01 and 0.03 respectively as default values.

6.3 Deep Learning-Based Steganography Techniques

This section discusses the important points of deep learning-based steganography techniques. The workflow diagrams of these techniques are discussed. The strong and weak features are discussed briefly at the end of this section.

6.3.1 Cover Generation Techniques

Cover generation techniques generate secure cover images for steganography. These artificial cover images are generated by GAN models like SGAN, and WGAN. The models described in SGAN Volkhonskiy et al. (2020), Zi et al., (2019) and Shi et al. (2018) have been developed for generating secure covers for steganography, and a comparison of the performance of these models is given in Table 6.2. The workflow diagram of the SGAN model for image steganography is shown in Fig. 6.2.

The model consists of a DCGAN generator, a DCGAN discriminator, and a CNN-based steganalyzer. DCGAN generator receives random noise as input and generates an image from this noise. The discriminator receives generated and real images alternatively and its role is to distinguish between real and generated images. A secret message is embedded in the generated cover image using a traditional steganography

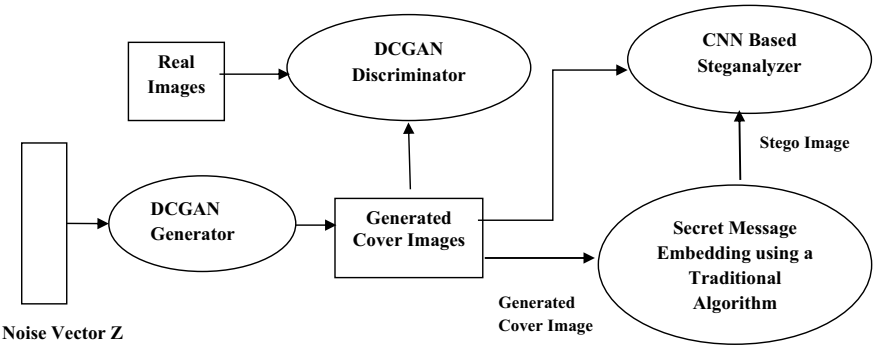


Fig. 6.2 Workflow diagram of SGAN

algorithm like LSB. In the original DCGAN, only the discriminator is used to perform classification, but in SGAN, steganalyzer is also used to perform binary classification. Steganalyzer's role is to distinguish between the generated cover image and the stego image. During the early stages of training, the generator produces images that do not resemble real images. As the training progresses, the generator produces better and better images with the help of the feedback received from the discriminator and the steganalyzer. The trained generator produces cover images that cannot be differentiated from the real images or stego images. This results in secure cover images for steganography. The generator is trained against the discriminator and the steganalyzer simultaneously using the loss function given below:

$$\begin{aligned} L \; = & \alpha\left(E_{X\sim p(X)}\left[\log D(X)\right] + E_{Z\sim p(z)}\left[\log(1 - \; D(G(Z)))\right]\right) \\ & + (1-\alpha)E_{Z\sim p(z)}\left[\log S(\text{Stego}(G(Z))) + \; \log(1 - S(G(Z)))\right] \end{aligned}$$

where α is a parameter that is used to control the trade-off between the realism of covers and security of covers.

Z is a noise vector.

X is a real image.

E (.) is the expectation value.

Stego (.) is the output of a traditional embedding algorithm.

S (.) is the output of the steganalyzer network.

G (.) is the output of the generator network (generated cover image).

D (.) is the output of the discriminator network.

6.3.2 Distortion Learning Techniques

Distortion learning techniques use high pass filtered cover images in distortion generators to produce distortion maps for steganography. A pre-trained simulator is used to embed secret messages in the distortion map. The distortion maps are generated by GAN-based distortion learning models like ASDL-GAN (Tang et al. 2017), UT-SCA-GAN (Yang et al. 2018), and UT-6-HPF-GAN (Yang et al. 2020). A comparison of the performance is presented in Table 6.3. The workflow diagram of the ASDL-GAN model for image steganography is shown in Fig. 6.3.

The distortion-based steganography algorithms use a high pass filtered version of the cover image to generate residual maps. On similar lines, GAN-based distortion learning models use high pass filtered versions of cover images for producing distortion maps and in steganalyzer. This ensures consistency between the generator and steganalyzer at the input level. As depicted in Fig. 6.4, the generator network receives the high pass filtered cover image as an input and generates the distortion map. The

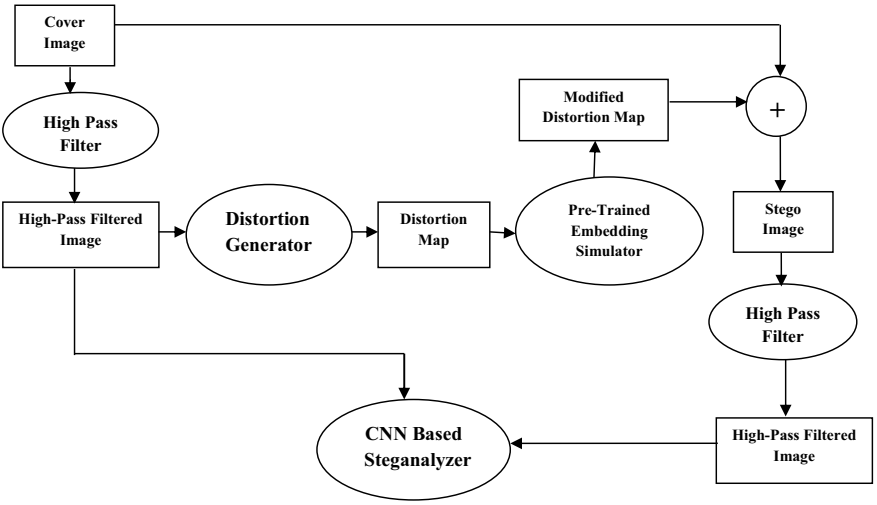


Fig. 6.3 Workflow of ASDL GAN

distortion map is fed into a small network called Ternary Embedding Simulator (TES) that is pre-trained and acts as an activation function for the generation of modified distortion map m. The modified distortion map m and the cover image are added together to form the stego image S. The role of the steganalyzer is to distinguish between generated stego images and the cover images. The two networks compete with each other to achieve the task. TES is pre-trained and uses the loss function given below:

$$l_{TES} = \frac{1}{H}\frac{1}{W}\sum_{i=1}^{H}\sum_{j=1}^{W}\left(m_{i,j} - m'_{i,j}\right)^2$$

where $m_{i,j}$ is the modified distortion map and $m'_{i,j}$ is the ground truth

The steganalyzer is trained using the function:

$$l_D = -\sum_{i=1}^{2} y'_i \log(y_i)$$

where y'_i represents ground truth and y_i represents the output of the steganalyzer.

The generator is trained using the loss function given below:

$$l_G = \alpha l_1 + \beta l_2$$

where α and β are the weightage given to l_1 and l_2 respectively

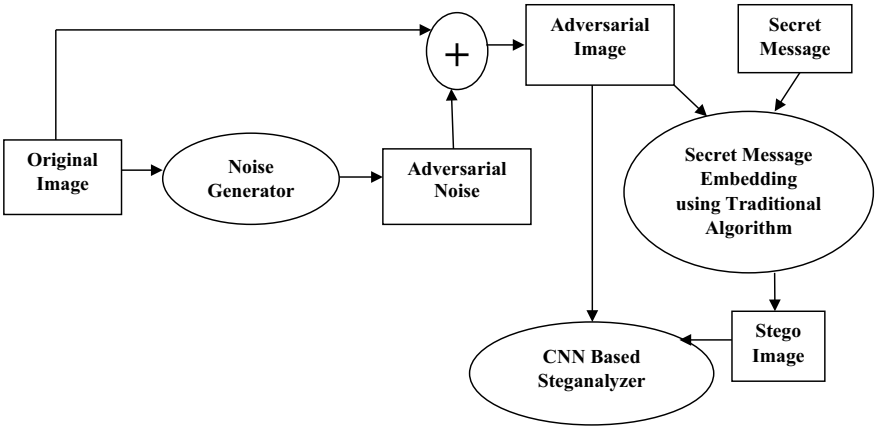


Fig. 6.4 Workflow of adversarial embedding technique

l_1 is given as $l_1 = -l_D$ and $l_2 = (\text{Capacity} - H * W * q)^2$

q represents the payload of messages that are carried by the stego image.

6.3.3 Adversarial Image Embedding Techniques

Adversarial image embedding techniques utilize adversarial examples for steganography for improved security. The approaches described in (Zhang et al. 2018; Zhou et al. 2020; Ma et al. 2019; Tang et al. 2019) use an adversarial image for the steganography process, and a comparison of performance is given in Table 6.4. An adversarial example is an image that is obtained by adding adversarial noise to an image. Figure 6.4 shows the workflow of the adversarial image embedding technique described in (Zhou et al. 2020).

As depicted in Fig. 6.4, the original image is fed into the generator network to produce the adversarial noise. The generated adversarial noise is added to the original image to produce the adversarial image. The secret data is then embedded in the adversarial image using the traditional embedding algorithm like WOW and UNIWARD. To improve the security of steganography, CNN based steganalyzer is used that receives adversarial image and stego image alternatively to train the generator so that adversarial and stego images cannot be differentiated. The steganalyzer outputs a normalized two-dimensional vector. This two-dimensional vector [a, b] determines the final classification result. If a > b, the image is more likely to be a cover image, and if a < b, the image is more likely to be a stego image. The distance d = (a–b) is used to decide if the image is a cover image or a stego image. The loss function for training the generator network is given as:

$$L_G = \lambda \left\| X - X' \right\|_2^2 + f(X')$$
$$f(X') = \text{abs}(Z(X') - Z(\text{Stego}(X'))$$

where X represents the original image and X′ is the adversarial image.

λ is the weight parameter of $\left\| X - X' \right\|_2^2$ (L2 function).

Stego (.) is the output of a traditional embedding algorithm.

Z (X′) is the distance d = (a–b) when input is a cover adversarial image.

Z (Stego(X′)) is the distance d = (a–b) when input is stego image.

Abs (.) is absolute value.

The steganalyzer is trained using the cross-entropy loss function given below

$$L_S = E[\log(S(X'))] + E[\log(1 - S(\text{Stego}(X')))]$$

where X′ represents the adversarial image.

Stego (X′) is the stego image produced after embedding secret data in the adversarial image.

S (.) represents the output of the steganalyzer network.

6.3.4 GAN Embedding Techniques

GAN embedding techniques for image steganography are unlike traditional algorithms as these techniques are free from any human intervention. A model representing this technique is trained using the adversarial training of GAN to embed a secret message in a cover image. The techniques described in (Abadi and Andersen 2016), GSIVAT (Hayes and Danezis 2017; Shi et al. 2019; Liu et al. 2018; Yedroudj et al. 2020; Yang et al. 2019; Fu et al. 2020) are GAN embedding techniques. A comparison of the performance of such techniques is given in Tables 6.5, 6.9, and 6.11. The GAN embedding GSIVAT model workflow is shown in Fig. 6.5.

A secret message (M) and a cover image (C) are fed to the generator network to generate the stego image (C′) as shown in Fig. 6.5. The extractor network receives the stego image generated and recovers the secret message. The discriminator network serves as a steganalyzer and receives both the cover image and the stego image alternatively and provides feedback for training the generator. The generator is trained so that the steganalyzer cannot differentiate between the cover image and the stego image. The output value of the steganalyzer that is close to 0 indicates that the input message is a stego image and the value that is close to 1 indicates that the input image is a cover image. Initially, the generator produces noisy images from which

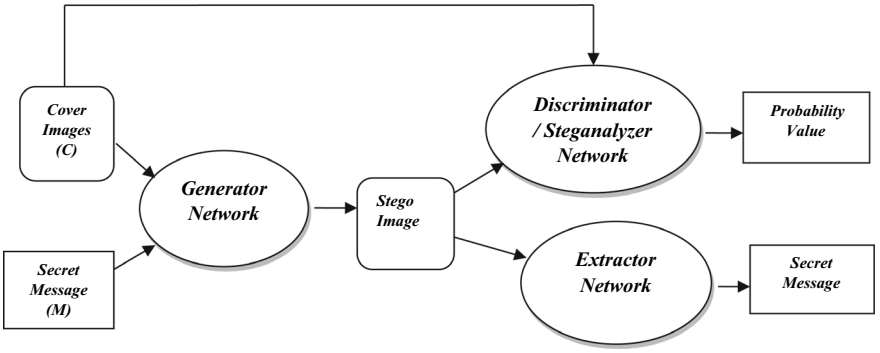


Fig. 6.5 Workflow of GSIVAT model

the extractor is not able to extract the secret message successfully. As the training progresses, all the networks improve their performances. With the trained generator, the extractor can successfully recover the secret message from the stego image. The extractor network is trained by minimizing the loss function shown below:

$$\begin{aligned} L_E(\theta_G, \theta_E, M, C) &= d\left(M, M'\right) \\ &= d\left(M, B\left(\theta_E, C'\right)\right) \\ &= d(M, B(\theta_E, A(\theta_G, C, M)) \end{aligned}$$

where θ_G and θ_E denote the parameters of the generator and extractor networks.

M and M' denote the original secret message and the recovered secret message.

C and C' denote the cover and stego images.

$d(M, M')$ is the Euclidean distance between M and M'.

The steganalyzer uses the Sigmoid cross-entropy loss function and is given as:

$$L_S\left(\theta_S, C, C'\right) = -y(\log(E(\theta_S, X))) - (1 - y)(\log(1 - E(\theta_S, X)))$$

where θ_S denotes the parameters of the steganalyzer network.

$y = 0$ indicates stego image i.e., $X = C'$, and $y = 1$ denotes cover image i.e., $X = C$.

The generator loss function is the weighted sum of the reconstruction loss, the extractor loss and the steganalyzer loss and is given as:

$$L_G(\theta_G, C, M) = \lambda_G \, d\left(C, C'\right) + \lambda_E L_E + \lambda_S L_S$$

where λ_G, λ_E, and λ_S are the weights assigned to the three losses.

6.3.5 *Embedding Less Techniques*

Embedding less steganography techniques produces a stego image without embedding a secret message in a cover image. Here the model learns how to generate the stego image directly from the noise and secret message. The approaches described in (Wang et al. 2018; Liu et al. 2018; Hu et al. 2018; Zhang et al. 2019; Ke et al. 2017) are embedding less approaches. A performance comparison of these approaches is given in Table 6.6. The workflow of the embedding less steganography model described in (Hu et al. 2018) is shown below in Fig. 6.6.

Figure 6.6a shows the stego image generation process. The segments S_i of secret information are mapped to the noise vectors Z_i by employing a mapping function.

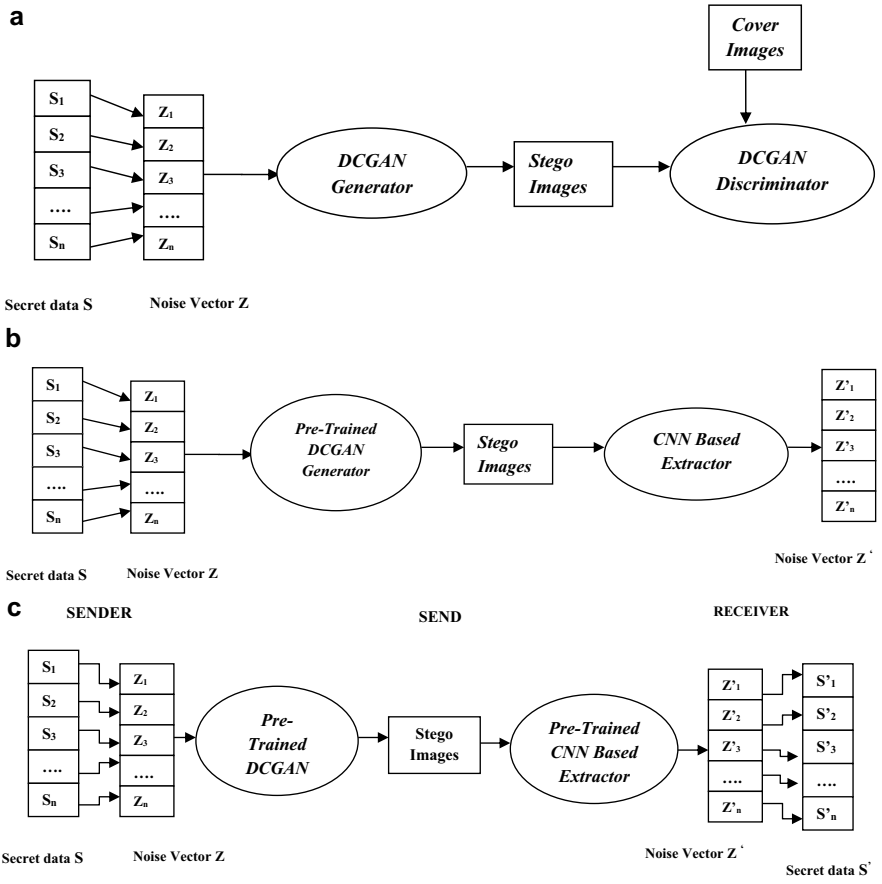


Fig. 6.6 **a** Training stego image generation, **b** Training secret message extractor, **c** Workflow of embedding less steganography model

These transformed noise vectors are then fed into the DCGAN generator for the generation of stego images. The trained generator is obtained after DCGAN converges using the loss function given below:

$$\frac{\min}{G}\frac{\max}{D}V(G, D) = E_{X\sim p(x)}\left[\log D(X)\right] + E_{Z\sim p(z)}\left[1 - \log(1 - D(G(Z)))\right]$$

where X represents the cover image.

Z is the mapped noise vector.

E denotes expectation.

G(Z) is the stego image generated by generator G.

Figure 6.6b displays the training of the extractor network. The extractor network receives the stego image generated by the trained generator and outputs the noise vector. The extractor is trained by using the difference between the original noise vector and the output of the extractor as the recovery accuracy measure. The extractor network training loss function is given as:

$$L_E = \sum_{i-1}^{n}\left(Z - E(\text{Stego})^2\right) = \sum_{i=1}^{n} Z - E(G(Z))^2$$

where Z is the noise vector, E(Stego) is the noise vector recovered by the extractor network. This noise vector is then used to recover the secret message bits using reverse mapping rules.

Figure 6.6c displays how the communication takes place. The sender uses the trained generator and the receiver uses the trained extractor. At the sender side, the secret message S is divided into segments S_i and then each segment S_i is mapped into noise vector Z_i, This mapping is fed to the trained generator which generates stego images. The extractor at the receiver side extracts the noise vector from the stego image. The secret data is recovered from the Noise vector as per the reverse mapping rules.

6.3.6 Category Label Techniques

Category label image steganography techniques use the conditional GAN model for the generation of stego images. The category labels and noise are fed to the generator network for the generation of stego images. The labels act as a driver for the stego image generation. The techniques described in (Zhang et al. 2020; Liu et al. 2018) are category label techniques. A comparison of the performance of these techniques

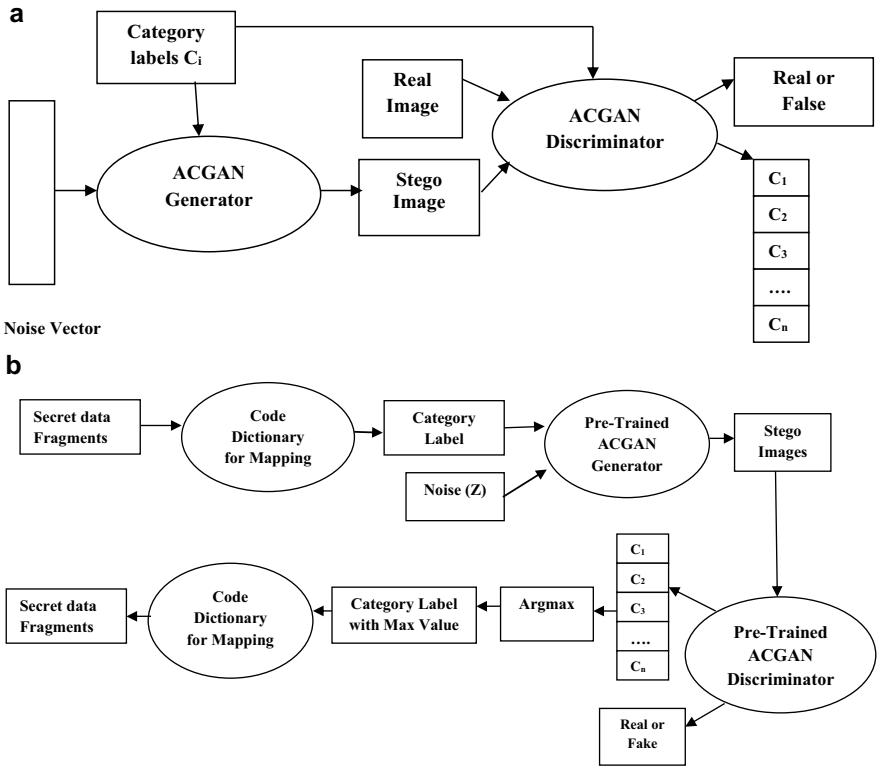


Fig. 6.7 **a** ACGAN model, **b** Workflow of a Noise-Label steganography model

is given in Table 6.7. Figure 6.7a and b show the workflow of the category label image steganography technique described in Liu et al. (2018).

The ACGAN generator is trained using a noise vector and the category label as two inputs as shown in Fig. 6.7a. The ACGAN discriminator has two outputs: one is the class probability value of the input image and the second is the category label of the input image. ACGAN is trained using a two-fold objective function which is given as:

$$L_G = E\left[\log P(S = \text{real}\,|X_{\text{real}})\right] + E\left[\log P\left(S = \text{Stego}|X_{\text{stego}}\right)\right]$$

$$L_C = E\left[\log P(C = C\,|X_{\text{real}})\right] + E\left[\log P(C = C|X_{\text{fake}})\right]$$

The generator is trained to maximize $L_C - L_G$ while the discriminator is trained to maximize $L_C + L_G$.

E (.) is the expectation value.

P (.) is the conditional probability.

The text information that needs to be secretly communicated is first encrypted using a code dictionary as shown in Fig. 6.7b. The mapped label and noise Z are used as input to train the ACGAN generator for the generation of the stego image. The relationship between the labels and secret data is created using a mapping function. The trained ACGAN discriminator receives the stego image at the receiver end and produces a category label as output. The category label is decoded using the same code dictionary to extract the secret message.

6.4 Performance Comparison of Image Steganographic Techniques

The performance of image steganographic techniques is compared in this section. The results reported by researchers on benchmark data sets CelebA, Bossbase, PASCAL-VOC12, CIFAR-100, ImageNet, and USC-SIPI have been used to evaluate the performance of various images steganography techniques. A summary of the datasets that have been used for comparing the performance is presented first.

6.4.1 Datasets Used for Comparison

The commonly used datasets in image steganography are briefly summarized below.

(a) CelebA (*CelebA Dataset*, n.d.) is a large–scale dataset of celebrity faces. It consists of 202599 face images of various celebrities across the globe with 10177 distinctive identities. Each image has 40 attribute annotations and five landmark locations.

(b) Bossbase (*Bossbase Dataset*, n.d.) is the benchmark dataset for steganography. It consists of 10,000 grayscale images of diverse scenes. The dataset has images of buildings, animals, landscapes, etc. Each image in the dataset is of size 512 $*$ 512.

(c) ImageNet (*ImageNet Object Localization Challenge | Kaggle*, n.d.) is a large-scale dataset of annotated images. It consists of approximately 14 million images of 21 thousand categories. Each category in the dataset consists of hundreds of images.

(d) PASCAL-VOC12 (*PASCAL VOC 2012 | Kaggle*, n.d.) consists of images of 20 different object classes such as dogs, cats, sheep, persons, sofas, plants, etc. There is a bounding box annotation, object class annotation, and pixel-level segmentation annotation for all the images in the dataset.

(e) CIFAR-100 (*CIFAR-10 and CIFAR-100 Datasets*, n.d.) is a tiny color image dataset of 100 different object categories such as butterfly, mountain, lion, mouse, etc. It consists of 50000 labeled images having 500 images of each

class and 10000 unlabeled images for testing. Each image is of size $32 * 32 * 3$.

(f) USC-SIPI (*SIPI Image Database*, n.d.) is a digital image database that consists of a collection of images such as brodatz and mosaic textures, high-altitude aerial images, mandrill, peppers, moving heads, moving vehicles, fly-overs, etc. The database is split into volumes having different image sizes of $256 * 256$, $512 * 512$, or $1024 * 1024$. All images are 8 bits/pixel for black and white images, and 24 bits/pixel for color images.

6.4.2 *Security Performance Comparison*

This section gives the security comparison of different steganography techniques. The security of a steganography technique is determined by the error rate of the steganalysis test performed by using a steganalyzer. The dataset used for the steganalysis test is split into two parts: the training dataset and the testing dataset. The steganalyzer is trained using the images in the training dataset and corresponding stego images. It is tested by using the testing dataset and the corresponding stego images. The error rate of the steganalysis test determines the security of a steganography technique. A higher error rate indicates better security and vice versa.

The security comparison of content-based steganography techniques such as WOW, HUGO, S-UNIWARD, and LSB is shown in Table 6.1. The steganalyzer employed in these methods is SRM except for (Lu et al. 2021) which uses RLCM-100D steganalyzer. All these methods employ the Bossbase dataset. The embedding capacity is set to 0.4 or 0.01 for the steganalysis test. From Table 6.1, it can be seen that content-based steganography methods have an error rate in the range of 10–25 which is not considered to be high.

The security performance of steganography approaches that use different cover generation techniques is performed. These techniques generate artificial cover images

Table 6.1 Security comparison of content-based traditional steganography techniques

References	Dataset	Embedding technique	Image size	Steganalyzer used	Relative capacity (bits per pixel)	Error of steganalyzer (%)
Pevný et al. (2010)	Bossbase	HUGO algorithm	$512 * 512$	SRM	0.4	>10
Binghamton (2012)	Bossbase	WOW algorithm	$128 * 128$	SRM	0.4	20
Holub et al. (2014)	Bossbase	S-UNIWARD algorithm	$512 * 512$	SRM	0.4	20
Lu et al. (2021)	Bossbase	LSB with halftone images	$256 * 256$	RLCM-100D	0.01	25

Table 6.2 Security comparison of cover generation steganography techniques

Cover generation technique	Dataset	Relative Capacity (bits per pixel)	Steganalyzer used	Embedding algorithm	Error of steganalyzer (%)
SGAN (Volkhonskiy et al. 2020)	CelebA	0.4	Self-defined	LSB	50
				HUGO	49.9
SSGAN (Shi et al. 2018)	CelebA	0.4	Self-defined	LSB	72
			Qian's Net	HUGO	71

Table 6.3 Security comparison of distortion learning steganography techniques

Method/references	Dataset	Relative capacity (bits per pixel)	Steganalyzer	Error of steganalyzer (%)
ASDL-GAN (Tang et al. 2017)	Bossbase	0.4	SRM	17
UT-SCA-GAN (Yang et al. 2018)	Bossbase	0.4	SRM	26
UT-6HPF-GAN (Yang et al. 2020)	Bossbase	0.4	SRM	29

Table 6.4 Security comparison of adversarial image embedding steganography techniques

Model/references	Dataset	Relative capacity (bits per pixel)	Embedding technique	Steganalyzer	Error of steganalyzer (%)
CNN-Adv-EMD (Tang et al. 2019)	Bossbase	0.4	Adversarial embedding (STC)	XuNet	26.3
ADV-EMD (Zhang et al. 2018)	Bossbase	0.4	Adversarial embedding (WOW)	XuNet	56.5

by using a given dataset. The cover generation models like SGAN and SSGAN are compared in Table 6.2, which shows the security performance of these cover generation techniques against the steganalysis test. These techniques use the CelebA dataset for training the model to generate artificial covers. The secret data is embedded in the generated covers using the LSB and HUGO methods with an embedding capacity of 0.4 bpp. The steganalyzer used by the SGAN is a self-defined steganalyzer whereas SSGAN employs a self-defined steganalyzer for LSB embedding and Qian's Net for HUGO embedding. The table shows that the SSGAN technique performs better than the SGAN technique for both HUGO and LSB embeddings. The results indicate that in general cover generation steganographic techniques are more secure than content-based steganographic techniques. In particular, the steganalysis test of

the SSGAN technique has produced very good results with an error rate of more than 70%.

The security comparison of the steganography methods that automatically learn the distortion function for steganography using GAN is performed. The distortion learning steganography models like ASDL-GAN, UT-6HPF-GAN, and UT-SCA-GAN are compared for security in Table 6.3. All these models employ the Bossbase dataset for the steganalysis test. The embedding capacity is set to 0.4 bpp. It employs SRM steganalyzer for the steganalysis. As depicted in the table, such techniques do not possess high resistance to steganalysis test and have security similar to that of the content-based traditional steganographic techniques.

The security performance of approaches that use the adversarial image embedding technique is performed. This technique uses adversarial images as cover images. The adversarial image embedding steganography models like ADV-EMD and CNN-Adv-EMD are compared in Table 6.4. The steganalysis test of the adversarial image embedding technique employs the Bossbase dataset. The embedding capacity is set to 0.4 bpp. The test makes use of the XuNET steganalyzer. As depicted in the table, these models achieve higher error rates than the distortion learning steganography models, but lower error rates than cover generation steganography models. Thus adversarial image embedding steganography models are more secure than the distortion learning steganography models and less secure than cover generation steganography models.

The security performance comparison of the GAN embedding technique is performed. This technique utilizes deep learning models for the entire steganography process without using any rules framed by humans. The technique uses cover images from a given dataset. The security measure of GAN embedding steganographic models like HIGAN and the model described in (Duan et al. 2019) is compared in Table 6.5. The steganalysis test uses the ImageNet dataset and XuNet steganalyzer. The model described in (Duan et al. 2019) produces an appreciable error rate of 60.3% during the steganalysis test but this error rate is less than 72% obtained from the cover generation steganography techniques.

The security performance comparison of embedding less technique is performed. This technique directly generates stego images using noise and secret data without embedding the secret data in the cover image. The security comparison of embedding less steganography models like SsteGAN, GSS, and SWE-GAN is shown in Table 6.6. The steganalysis test of these models uses the CelebA dataset. As depicted in the table, the test of SsteGAN and GSS models produces an error rate of around

Table 6.5 Security comparison of GAN embedding steganography techniques

Cover-based techniques	Dataset	Steganalyzer used	Embedding technique	Relative capacity (bits per pixel)	Error of steganalyzer (%)
Duan et al. (2019)	ImageNet	XuNet	Cover-Based	8	28.9
HIGAN (Fu et al. 2020)	ImageNet	XuNet	Cover-Based	8	60.3

Table 6.6 Security comparison of embedding less steganography technique

Generative models	Dataset	Relative capacity (bits per pixel)	Embedding technique	Steganalyzer used	Error of steganalyzer (%)
SsteGAN (Wang et al. 2018)	CelebA	0.4	Noise-based	Self-defined	40.84
GSS (Zhang et al. 2019)	CelebA	0.4	Noise-based	SPAM	<50
SWE-GAN (Hu et al. 2018) Case1	CelebA	0.0732	Noise-based	CRM	99.2
SWE-GAN (Hu et al. 2018) *Case 2*				*CRM*	*44*

Table 6.7 Security comparison of category label techniques

Generative models	Dataset	Embedding technique	Steganalyzer used	Relative capacity (bits per pixel)	Error of steganalyzer (%)
CIH-GAN (Liu et al. 2018)	MNIST	Noise-Label	EC + SPAM	$9.125 * 10^{-5}$	52
SSS-GAN (Zhang et al. 2020) Case1	MNIST	Noise-Label	CRM	$9.125 * 10^{-5}$	99.9
SSS-GAN (Zhang et al. 2020) Case 2			CRM		56

50%. The SWE-GAN model is tested for two cases: in case 1, stego images are kept private and are not used by the steganalyzer CRM for training. This results in low accuracy and high security; in case 2, stego images are made public and are used by the steganalyzer CRM for training. This decreases the error rate on the steganalyzer from 99.2 to 44% and hence security is decreased. These models have overall high security but when the stego data is made public there is an appreciable decline in the security.

The security performance comparison of category labels steganographic technique is performed. This technique directly generates stego images from the noise and the category labels. The category labels act as a driver for the stego image generation. The security performance comparison of the category labels models like CIH-GAN and SSS-GAN is shown in Table 6.7. CelebA dataset has been used for the steganalysis test. The CIH-GAN model has resulted in an error rate of 52%. SSS-GAN has been tested for two cases: in case 1, stego images are kept hidden, and are not used by the steganalyzer CRM for training; in case 2, stego images are made public, and are used by the steganalyzer CRM for training. The error rate decreases from 99.9 to

56% when stego images are made public and hence decreases the security. These models have overall high security but when the stego data is made public the security decreases appreciably.

6.4.3 Embedding Capacity Comparison

The embedding capacity performance comparison of the content-based steganographic technique is performed first. This technique makes use of rules framed by humans for the steganographic process. The content-based steganography models like OTPVD, EOTPVD, PBPVD, LSB + PVD, and LSB + QVD are compared in Table 6.8. The embedding capacity comparison test makes use of the USC-SIPI dataset with an image size of 512 ∗ 512. The table shows that OTPVD, EOTPVD, PBPVD, and LSB + PVD models have embedding capacity in the range 2.483–4.55 bpb with PSNR in the range 33.06–40.4. LSB + QVD model has the highest embedding capacity of 4.55 bpb with a PSNR value of 33.06.

The embedding capacity comparison of the GAN embedding steganography technique is performed. The embedding capacity measure of GAN embedding steganography models like End-to-End CNN and the one described in (Duan et al. 2019) is compared in Table 6.9. The models use the ImageNet dataset for the embedding capacity test. As depicted in the table, both the models have an embedding capacity of 8 bpp, but (Duan et al. 2019) produce better PSNR value of 40.47.

Table 6.8 Embedding capacity comparison of traditional steganography techniques

References	Dataset	Embedding technique	Image size	Capacity (bits per byte)	Average PSNR
Grajeda-Marín et al. (2018)	USC-SIPI	OTPVD	512 ∗ 512	2.483	37.774
		EOTPVD		2.733	37.635
Hussain et al. (2017)	USC-SIPI	PBPVD + iRMDR	512 ∗ 512	3	38.84
Swain (2016)	USC-SIPI	LSB + PVD	512 ∗ 512	3.1	40.4
Swain (2019)	USC-SIPI	LSB + QVD	512 ∗ 512	4.55	33.06

Table 6.9 Embedding capacity comparison of GAN embedding steganography models

Models	Embedding technique	Dataset	Relative capacity (bits per pixel)	Average PSNR
End-to-End CNN (ur Rehman et al. 2019)	Cover-based	ImageNet	8	32.9
Duan et al. (2019)	Cover-based	ImageNet	8	40.47

6.4.4 Invisibility Comparison

The invisibility performance comparison of the content-based steganographic technique is performed first. Invisibility is determined using distortion measuring metrics such as PSNR or SSIM as discussed in Sect. 6.2. The embedding capacity is set to 1 and 0.81 for this test. The invisibility performance of the difference expansion-based model and cubic reference table based model is compared in Table 6.10. Both the models use images of size $512 * 512$ from the USC-SIPI dataset for the invisibility test. As depicted in the table, both models produce high PSNR values in the range of 43–50.

The invisibility comparison of the GAN embedding technique is performed. The distortion measure of GAN-embedding models like End-to-End CNN and ISGAN is compared in Table 6.11. These models use the PASCAL-VOC12 dataset for the distortion measure. The distortion is measured using PSNR and SSIM and embedding capacity is set to bpp. From the table, it is clear that these methods produce a PNSR value of 33–34 and an SSIM value in the range of 0.94–0.95.

From the above Tables, it can be concluded that the most secure content-based steganography techniques produce an error rate of 50% only for the steganalysis test (Table 6.2). Embedding capacity and image quality of the steganography process is not very high. Fully deep learning techniques and hybrid steganography techniques have been explored by researchers to improve the performance of steganography. Embedding data in artificial covers or adversarial images has resulted in better security than the traditional methods. However, distortion learning methods proposed so far are not more secure than traditional distortion minimization methods (Table 6.3). Fully deep learning techniques such as GAN embedding methods have achieved very good results in terms of security and capacity compared to the traditional and hybrid methods. Embedding Capacity has been increased (Table 6.9). SWE-GAN and SSS-GAN have achieved high error rate on different steganalyzers when stego images are kept hidden (Tables 6.6 and 6.7). However, the error rate decreases appreciably to around when the stego images are made public. Further, the embedding capacity of these methods is limited. The invisibility of the deep learning methods measured by PSNR and SSIM is good but not better than traditional methods. A lot of work has been done in deep learning based steganography in terms of security. There is a scope for improving the deep learning steganographic techniques that can improve the capacity and invisibility while keeping the security high. There is also a need for

Table 6.10 Invisibility comparison of traditional steganography techniques

References	Dataset	Embedding technique	Image size	Average PSNR	Relative capacity (bpp)
Chang et al. (2017)	USC-SIPI	Difference Expansion	$512 * 512$	43	0.81
Liao et al. (2018)	USC-SIPI	CRT	$512 * 512$	48.615	1
		CRT-PVD		49.41	1

Table 6.11 Distortion measurement of GAN embedding techniques for improved imperceptibility

Model/references	Cover	Stego (Secret)	Stego-cover (Encoding)		Recovered-secret (Decoding)		Relative capacity (bpp)
			Average PSNR (db)	Average SSIM (db)	Average PSNR (db)	Average SSIM (db)	
End-to-End CNN (Rehman et al. 2019)	PASCAL-VOC12	PASCAL-VOC12	33.7	0.96	35.9	0.95	8
ISGAN (Zhang et al. 2019)	PASCAL-VOC12	PASCAL-VOC12	34.49	0.9661	33.31	0.9467	8

new suitable deep learning architectures that can improve the capacity and invisibility of image steganography.

6.5 Challenges and Future Directions

Despite the progress made, deep learning-based image steganography faces several challenges. A primary issue is balancing imperceptibility with payload capacity, as embedding more data can introduce detectable distortions, which compromise security. While deep learning has improved this trade-off, achieving a higher payload without sacrificing image quality remains difficult. Another challenge is the robustness of stego images against increasingly sophisticated steganalysis techniques that utilize deep learning models like XuNet, which can detect minute alterations in images. Additionally, the generalization of these models across different datasets poses problems. Although models work well on specific datasets, ensuring they perform equally well on new, unseen data requires further research.

Resource efficiency is another significant challenge. Deep learning models are computationally expensive, requiring large datasets and powerful hardware. This limits their accessibility for real-time applications. Furthermore, scaling models to handle high-resolution images introduces complexity, as the architecture must be more sophisticated to maintain imperceptibility and capacity in larger images. Lastly, deep learning models are susceptible to adversarial attacks, where small modifications in the input data can disrupt the embedding or extraction process, threatening the security of hidden communications.

Looking ahead, there are several promising directions for research in deep learning-based steganography. Future efforts could likely focus on increasing the payload capacity while minimizing distortion, possibly by employing more advanced architectures like transformers or diffusion models. Adversarial training can also enhance the security of stego images, making them more resistant to both traditional and deep learning-based steganalysis. Another exciting avenue is cross-media steganography, where information can be embedded across multiple types of media, such as images, audio, and video, enhancing the versatility of steganographic methods.

Lightweight models that reduce computational costs are crucial for expanding the use of steganography in real-time applications, such as secure video streaming. Techniques like model pruning and knowledge distillation can be explored to develop efficient, scalable solutions. On the security front, combining steganography with blockchain and cryptography could add layers of protection, offering stronger encryption and secure key management. Steganography may also become integral to privacy-preserving systems, protecting sensitive data in communications and cloud storage environments, which will be especially valuable in an era of increasing digital surveillance and cyber threats.

6.6 Summary

Deep learning-based image steganography techniques were discussed in this chapter. The techniques were divided into various categories that were based on the embedding strategy employed. The three criteria, namely security, capacity, and invisibility for measuring the performance of steganography were described, and were used to compare the performance of various deep learning based steganography techniques. The benchmark data sets CelebA, Bossbase, PASCAL-VOC12, CIFAR-100, and ImageNet were used for evaluating the performance of various image steganography techniques. It was observed from the results that the capacity of deep learning-based models is still less than the traditional techniques. Future work can include developing an improved deep learning-based image steganography technique that has better embedding capacity. There is also a scope to improve the imperceptibility of the deep learning-based image steganography technique as little work has been done on improving imperceptibility.

Bibliography

1. B. Sultan, M.A. Wani, Enhancing steganography capacity through multi-stage generator model in generative adversarial network based image concealment. J. Electron. Imaging **33**(3), 033026 (2024). https://doi.org/10.1117/1.JEI.33.3.033026
2. B. Sultan, M.A.Wani, Anewframework for analyzing colormodels with generative adversarial networks for improved steganography. Multimed. Tools Appl. (2023). https://doi.org/10.1007/s11042-023-14348-7
3. M.A. Wani, B. Sultan, Deep learning based image steganography: a review.Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 1–26 (2022). https://doi.org/10.1002/widm.1481
4. W. Lu, Y. Xue, Y. Yeung, H. Liu, J. Huang, Y.Q. Shi, Secure halftone image steganography based on pixel density transition. IEEE Trans. Dependable Secure Comput. **18**(3), 1137–1149 (2021). https://doi.org/10.1109/TDSC.2019.2933621
5. D. Volkhonskiy, I. Nazarov, E. Burnaev, Steganographic generative adversarial networks (2020), p. 97. https://doi.org/10.1117/12.2559429
6. J. Yang, D. Ruan, J. Huang, X. Kang, Y.Q. Shi, An embedding cost learning framework using GAN. IEEE Trans. Inf. Forensics Secur. **15**, 839–851 (2020). https://doi.org/10.1109/TIFS.2019.2922229
7. L. Zhou, G. Feng, L. Shen, X. Zhang, On security enhancement of steganography via generative adversarial image. IEEE Signal Process. Lett. **27**, 166–170 (2020). https://doi.org/10.1109/LSP.2019.2963180
8. Z. Zhang, G. Fu, R. Ni, J. Liu, X. Yang, A generative method for steganography by cover synthesis with auxiliary semantics. Tsinghua Sci. Technol. **25**(4), 516–527 (2020). https://doi.org/10.26599/TST.2019.9010027
9. M. Yedroudj, F. Comby, M. Chaumont, Steganography using a 3-player game. J. Vis. Commun. Image Represent. **72**, 102910 (2020). https://doi.org/10.1016/j.jvcir.2020.102910
10. Z. Fu, F. Wang, X. Cheng, The secure steganography for hiding images via GAN. EURASIP J. Image Video Process. (2020). https://doi.org/10.1186/s13640-020-00534-2
11. H. Shi, X.Y. Zhang, S. Wang, G. Fu, J. Tang, *Synchronized detection and recovery of steganographic messages with adversarial learning. Lecture Notes in Computer Science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*, vol. 11537. (Springer, 2019), pp. 31–43

12. W. Tang, B. Li, S. Tan, M. Barni, J. Huang, CNN-based adversarial embedding for image steganography. IEEE Trans. Inf. Forensics Secur. **14**(8), 2074–2087 (2019). https://doi.org/10.1109/TIFS.2019.2891237
13. S. Ma, X. Zhao, Y. Liu, Adaptive spatial steganography based on adversarial examples. Multimed. Tools Appl. **78**(22), 32503–32522 (2019). https://doi.org/10.1007/s11042-019-07994-3
14. Z. Zhang, J. Liu, Y. Ke, Y. Lei, J. Li, M. Zhang, X. Yang, Generative steganography by sampling. IEEE Access **7**, 118586–118597 (2019). https://doi.org/10.1109/ACCESS.2019.2920313
15. H. Zi, Q. Zhang, J. Yang, X. Kang, Steganography with convincing normal image from a joint generative adversarial framework, in 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, APSIPA ASC 2018: Proceedings, November (2019), pp. 526–532. https://doi.org/10.23919/APSIPA.2018.8659716
16. A. ur Rehman, R. Rahim, S. Nadeem, S. ul Hussain, End-to-end trained CNN encoder-decoder networks for image steganography. Lecture Notes in Computer Science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), 11132 LNCS (2019), pp. 723–729. https://doi.org/10.1007/978-3-030-11018-5_64
17. G. Swain, Very high capacity image steganography technique using quotient value differencing and LSB substitution. Arab. J. Sci. Eng. **44**(4), 2995–3004 (2019). https://doi.org/10.1007/s13369-018-3372-2
18. X. Duan, K. Jia, B. Li, D. Guo, E. Zhang, C. Qin, Reversible image steganography scheme based on a U-net structure. IEEE Access **7**, 9314–9323 (2019). https://doi.org/10.1109/ACCESS.2019.2891247
19. H. Shi, J. Dong, W. Wang, Y. Qian, X. Zhang, *SSGAN: Secure Steganography Based on Generative Adversarial Networks*, vol. 10735 (Springer International Publishing, LNCS, 2018)
20. M.M. Liu, M.Q. Zhang, J. Liu, P.X. Gao, Y.N. Zhang, Coverless information hiding based on generative adversarial networks. Yingyong Kexue Xuebao/J. Appl. Sci. **36**(2), 371–382 (2018). https://doi.org/10.3969/j.issn.0255-8297.2018.02.015
21. I.R. Grajeda-Marín, H.A. Montes-Venegas, J.R. Marcial-Romero, J.A. Hernandez-Servín, V. Muñoz-Jiménez, G.D.I. Luna, A new optimization strategy for solving the fall-off boundary value problem in pixel-value differencing steganography. Int. J. Pattern Recogn. Artif. Intell. **32**(1), 1–17 (2018). https://doi.org/10.1142/S0218001418600108
22. Z. Wang, N. Gao, X. Wang, X. Qu, L. Li, *SSteGAN: Self-learning steganography based on generative adversarial networks. Lecture Notes in Computer Science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*, vol. 11302 (Springer International Publishing, 2018). https://doi.org/10.1007/978-3-030-04179-3_22
23. X. Liao, S. Guo, J. Yin, H. Wang, X. Li, A.K. Sangaiah, New cubic reference table based image steganography. Multimed. Tools Appl. **77**(8), 10033–10050 (2018). https://doi.org/10.1007/s11042-017-4946-9
24. J. Yang, K. Liu, X. Kang, E.K. Wong, Y.-Q. Shi, Spatial image steganography based on generative adversarial network **1**, 1–7 (2018). http://arxiv.org/abs/1804.07939
25. Y. Zhang, W. Zhang, K. Chen, J. Liu, Y. Liu, N. Yu, Adversarial examples against deep neural network based steganalysis, in *Proceedings of the 6th ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec'18)*. (Association for Computing Machinery, 2018), pp. 67–72. https://doi.org/10.1145/3206004.3206012
26. D. Hu, L. Wang, W. Jiang, S. Zheng, B. Li, A novel image steganography method via deep convolutional generative adversarial networks. IEEE Access **6**, 38303–38314 (2018). https://doi.org/10.1109/ACCESS.2018.2852771
27. J. Ye, J. Ni, Y. Yi, Deep learning hierarchical representations for image steganalysis. IEEE Trans. Inf. Forensics Secur. **12**(11), 2545–2557 (2017). https://doi.org/10.1109/TIFS.2017.2710946
28. C.C. Chang, Y.H. Huang, T.C. Lu, A difference expansion based reversible information hiding scheme with high stego image visual quality. Multimed. Tools Appl. **76**(10), 12659–12681 (2017). https://doi.org/10.1007/s11042-016-3689-3

29. M. Hussain, A.W. Abdul Wahab, A.T.S. Ho, N. Javed, K.H. Jung, A data hiding scheme using parity-bit pixel value differencing and improved rightmost digit replacement. Signal Process. Image Commun. **50**, 44–57 (2017). https://doi.org/10.1016/j.image.2016.10.005
30. W. Tang, S. Tan, B. Li, J. Huang, Automatic steganographic distortion learning using a generative adversarial network. IEEE Signal Process. Lett. **24**(10), 1547–1551 (2017). https://doi.org/10.1109/LSP.2017.2745572
31. J. Hayes, G. Danezis, Generating steganographic images via adversarial training. Adv. Neural Inf. Process. Syst. **2017**, 1955–1964 (2017)
32. Y. Ke, M. Zhang, J. Liu, T. Su, X. Yang, Generative steganography with Kerckhoffs' principle based on generative adversarial networks 1–5 (2017). http://arxiv.org/abs/1711.04916
33. M. Abadi, D.G. Andersen, Learning to protect communications with adversarial neural cryptography. Nature 1–15 (2016). http://arxiv.org/abs/1610.06918. https://doi.org/10.1007/978-3-030-22741-8_3
34. G. Swain, A steganographic method combining LSB substitution and PVD in a block. Procedia Comput. Sci. **85**, 39–44 (2016). https://doi.org/10.1016/J.PROCS.2016.05.174
35. G. Xu, H.Z. Wu, Y.Q. Shi, Structural design of convolutional neural networks for steganalysis. IEEE Signal Process. Lett. **23**(5), 708–712 (2016). https://doi.org/10.1109/LSP.2016.2548421
36. D. Lerch-Hostalot, D. Megías, Unsupervised steganalysis based on artificial training sets. Eng. Appl. Artif. Intell. **50**(April), 45–59 (2016). https://doi.org/10.1016/j.engappai.2015.12.013
37. V. Holub, J. Fridrich, T. Denemark, Universal distortion function for steganography in an arbitrary domain. EURASIP J. Inf. Secur. **2014**, 1–13 (2014). https://doi.org/10.1186/1687-417X-2014-1
38. V. Holub, J. Fridrich, Designing steganographic distortion using directional filters, in *2012 IEEE International Workshop on Information Forensics and Security (WIFS)* (Costa Adeje, Spain, 2012), pp. 234–239. https://doi.org/10.1109/WIFS.2012.6412655
39. J. Fridrich, J. Kodovsky, Rich models for steganalysis of digital images. IEEE Trans. Inf. Forensics Secur. **7**(3), 868–882 (2012). https://doi.org/10.1109/TIFS.2012.2190402
40. T. Pevný, T. Filler, P. Bas, *Using high-dimensional image models to perform highly undetectable steganography. Lecture Notes in Computer Science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*, 6387 LNCS (2010), pp. 161–177. https://doi.org/10.1007/978-3-642-16435-4_13

Chapter 7
Two-Stage Depth-Balanced Generative Adversarial Networks for Image Steganography

7.1 Introduction

In steganography, a stego image is normally created by combining the secret message (M) with the cover image (C) by an algorithm or deep learning model (f). This process essentially embeds the secret message within the cover image to create the stego image (S):

$$S = f(C, M) \tag{7.1}$$

However, as the embedding capacity is increased the quality of stego image degrades. This chapter discusses a new two-stage depth-balanced GAN architecture designed for steganography. The new architecture addresses the inherent challenges in steganographic systems by leveraging the strengths of GANs to achieve a balance between data embedding capacity and imperceptibility. The approach improves the embedding capacity and it is robust against steganalysis, making it more secure than the traditional steganography methods.

The two-stage depth-balanced steganography approach is discussed in detail below.

7.2 Two-Stage Depth-Balanced Steganography Approach

The approach involves a depth balancing mechanism that optimizes the depth of the two stages of the generator network, ensuring efficient and effective data hiding while maintaining a good quality of the stego image. It also enhances its robustness against detection by advanced steganalysis techniques.

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_7

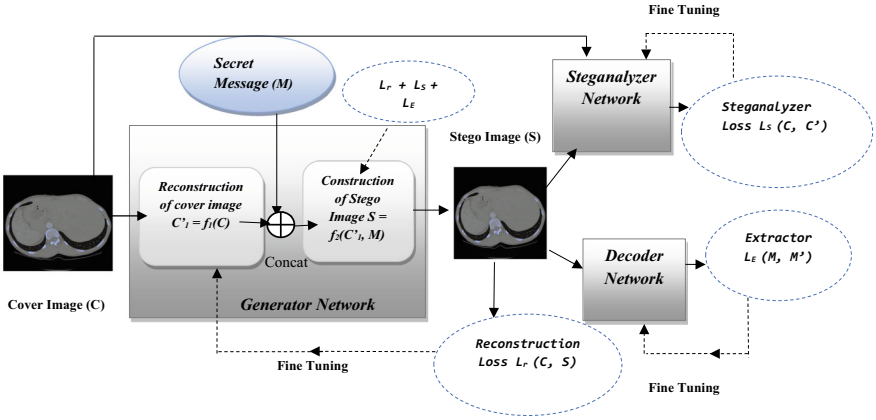


Fig. 7.1 Workflow diagram of the two-stage depth-balanced steganography system

7.2.1 Workflow Diagram of the Two-Stage Depth-Balanced Steganography System

The **two-stage depth-balanced steganography system** divides the steganographic process into two distinct stages: cover image reconstruction and stego image construction as depicted in Fig. 7.1.

The first stage of the workflow focuses on reconstructing high-quality cover images that are suitable for steganography. In this stage, the cover image is reconstructed to capture essential visual features, employing an architecture with depth that is inversely related to the depth of the second stage. If the first stage is shallow, the second is deep, and vice versa. The second stage uses the reconstructed cover image and embeds it with secret data to generate a stego image. In this stage, the secret message is embedded within the reconstructed cover image, ensuring secure data hiding. The final stego image retains the same dimensions as the original cover image, maintaining a good visual quality with increased embedding capacity. Essentially, the stego image is created by the two-stage depth-balanced generator.

7.2.2 Two-Stage Depth-Balanced Generator

The two-stage depth-balanced generator is designed to efficiently manage both cover reconstruction and secret message embedding through a two-stage process. Each stage utilizes a variable-depth architecture, as illustrated in Fig. 7.2. A key feature of this technique is the inverse relationship between the depth of these two stages. When a shallow architecture is chosen for cover image reconstruction, a deeper architecture is employed for stego image creation, and vice versa. This depth balance enhances both embedding capacity and security.

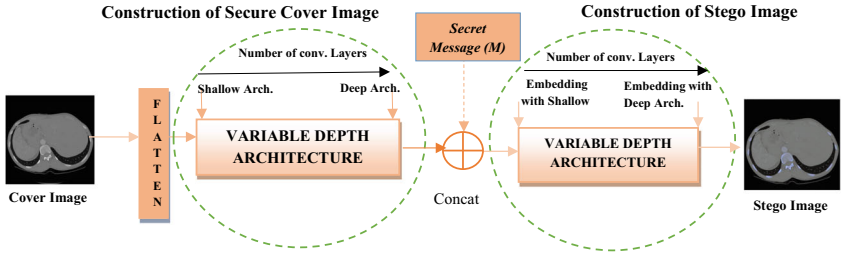


Fig. 7.2 Two stage generator for depth balanced steganography approach

7.2.2.1 Cover Image Reconstruction Stage

Here cover images (C) undergo an initial reconstruction process, which is unlike the traditional approach, where a cover image is directly used to create the stego image. The cover image is subjected to a series of transformations to produce the reconstructed cover image, referred to as C_R. The dimensions of the reconstructed cover image, C_R, are modified according to the depth of the cover image reconstruction architecture. The process of reconstruction is expressed as follows:

$$C_R = f_1(C) \tag{7.2}$$

- f_1 represents a set of transformations required for the reconstruction of the cover image.

7.2.2.2 Stego Image Construction Stage

This stage employs the reconstructed cover image and the secret message to produce the stego image (S). A set of convolution operations is applied to the reconstructed cover image (C_R) and the secret message (M) to create the stego image. The process is expressed as follows:

$$S = f_2(C_R, M) \tag{7.3}$$

where f_2 represents a set of transformations required for the construction of stego image.

7.3 Training Algorithms and Loss Functions

The training algorithms optimize the model parameters, ensuring faster convergence and improved performance. Binary cross-entropy and mean squared error loss functions are used for both the cover image reconstruction and the stego image generation tasks.

7.3.1 Training Algorithms

Algorithm 1 focuses on the reconstruction of the cover image. A random mini-batch of cover images are resized to a fixed dimension of 64 × 64 pixels. These images are then flattened and passed through the first stage of the Generator Network. In this stage, a series of transformations are applied to the cover image. The output of this algorithm is a mini-batch of reconstructed cover images, which are used for embedding the secret message.

Algorithm 1. Reconstruction of Cover Images

Input: *Mini-batch of Cover images* C

1. *Choose a random sample C with dimensions 64 × 64 from the dataset of cover images.*
2. *Flatten the selected sample C and feed it into stage 1 of the Generator Network.*
3. *Implement a series of transformations on this flattened cover image and subject it to training to generate a reconstructed cover image* C_R*, the dimensions of which are determined by the depth of the stage1 of the generator network.*

Output: *Mini-batch of Reconstructed Cover Images* C_R

Algorithm 2 embeds the secret messages with the reconstructed cover image. It takes a mini-batch of reconstructed cover images and a corresponding mini-batch of secret messages to produce stego images. The resulting stego images are indistinguishable from cover images but securely hide secret messages.

Algorithm 2. Stego Image Construction

Input: *Mini-batch of Reconstructed Cover Images* C_R*, Mini-batch of Secret Messages M*

1. *Concatenate the secret messages M with the reconstructed cover images* C_R*.*
2. *Send the concatenated input to stage 2 of the Generator Network.*
3. *Implement a series of transformations on the concatenated input and subject it to training to generate the stego image S with dimensions.*

Output: *Mini-batch of Stego images S*

7.3.2 Loss Functions

The loss functions compute the difference between the predicted and actual outputs, facilitating the model to optimize its parameters. The first stage of the generator model takes cover image C as input and it produces the reconstructed cover image C_R which is given as:

$$C_R = G_{CR}(\theta_{CR}, C)$$

- G_{CR} (.) represents the first stage of the Generator Model (cover image reconstruction stage).
- θ_{CR} represents the parameters of the first stage of the Generator Model.

The second stage of the generator model takes the reconstructed cover image C_R and the secret message M as input and it produces the stego image S which is given as:

$$S = G_{SC}(\theta_{SC}, C_R, M)$$

- G_{SC} (.) represents the 2nd stage of the generator model (stego image construction stage).
- θ_{SC} represents the parameters of the second stage of the generator model.

The decoder model uses the stego image S to recover the secret message M′ which is given as:

$$M' = D(\theta_D, S)$$

- D (.) represents the Decoder Model.
- θ_D represents the parameters of the decoder model.

7.3.2.1 Decoder Loss

The decoder network is fine-tuned by reducing the Euclidean distance 'd' between the original secret message (M) and its reconstructed version (M'). This optimization is accomplished by utilizing the loss function L_D given in Eq. (7.4) below.

$$\begin{aligned} L_D(\theta_D, M') &= d(M, M') \\ &= d(M \,.\, D(\theta_{D,S})) \end{aligned} \tag{7.4}$$

7.3.2.2 Steganalyzer Loss

The steganalyzer model conducts binary classification, with its loss function L_{ST} defined by the sigmoid cross-entropy equation as shown in Eq. (7.5) below.

$$L_{ST}(\theta_S, C, S) = -y \log(ST(\theta_{ST}, x)) - (1 - y) \cdot \log(1 - ST(\theta_{ST}, x)) \tag{7.5}$$

- θ_{ST} represents the parameters of the Steganalyzer Model.
- ST (.) signifies the Steganalyzer Model.

When the input x is the cover image C ($x = C$), the resulting output y is assigned the value of 1, and when x is stego image S ($x = S$), y is set to 0.

7.3.2.3 Cover Image Reconstruction Loss (Stage 1 of the Generator Network)

The generator network is trained in two stages. The cover image reconstruction stage and the stego images construction stage. The loss function of the cover image reconstruction stage L_{CR} is defined in Eq. (7.6) below.

$$L_{CR}(\theta_{CR}, C) = d(C, S) \tag{7.6}$$

7.3.2.4 Stego Image Construction Loss (Stage 2 of the Generator Network)

The stego image construction stage undergoes training using a weighted blend of three loss functions: the cover image reconstruction loss L_{CR}, the decoder loss L_D, the steganalyzer loss L_{ST}. The stego image construction loss function L_{SC} is shown in Eq. (7.7) below.

$$L_{SC}(\theta_{SC}, C_R, M) = \lambda_G \cdot L_{CR} + \lambda_D \cdot L_D + \lambda_S \cdot L_{ST} \tag{7.7}$$

- θ_{SC} represents the parameters of the stego image construction Stage.
- λ_G, λ_D, and λ_S are the weights assigned to the reconstruction loss, decoder loss, and steganalyzer loss, respectively.

7.4 Results and Discussions

The experimental setup and dataset employed are discussed in this section.

The performance of the approach discussed here is compared to other GAN-based models reported by other researchers.

7.4.1 Experimental Setup and Dataset Used

The experiments used 495 MRI and CT images of size $512 \times 512 \times 3$ from DICOM dataset, which were converted from dcm format to jpg format. Out of 495 images, 396 were used for training while the remaining 99 were utilized for testing purposes. Adam optimizer with a learning rate set at 0.0002 was used. The experiments were conducted using TensorFlow 1.15.0, running on the DGX A100 GPU.

7.4.2 Performance of Two-Stage Steganography Model with Different Payloads

The two-stage steganography model was trained with varying payloads ranging from 1 bit per pixel (bpp) to 3 bpp and results are shown in Table 7.1. It can be seen from Table 7.1 that as the embedding capacity is increased from 1 to 2 bpp, the message extraction accuracy decreases from 99.58 to 96.34%, peak signal-to-noise ratio (PSNR) decreases from 39.27 to 35.77, and structural similarity index (SSIM) decreases from 0.9375 to 0.9072. As the embedding capacity is further increased to 3 bpp, these metrics further decrease to 92.52, 32.66, and 0.8935 respectively. This decline can be attributed to the fact that as embedded information increases, images inevitably become noisier, resulting in a reduction in image quality.

The cover image and the corresponding stego image generated by the two-stage model for 1 bpp embedding capacity are shown in Fig. 7.4. It can be seen by visual inspection that the quality of stego image is not degraded at 1 bpp embedding capacity value.

Table 7.1 Performance of two-stage steganography model with different payloads using DICOM dataset

Evaluation criteria	Embedding capacity		
	1 bpp	2 bpp	3 bpp
PSNR	39.27	35.77	32.66
SSIM	0.9375	0.9072	0.8935
Extraction accuracy (%)	99.58	96.34	92.52
Steganalyzer error rate (%)	50	47	39

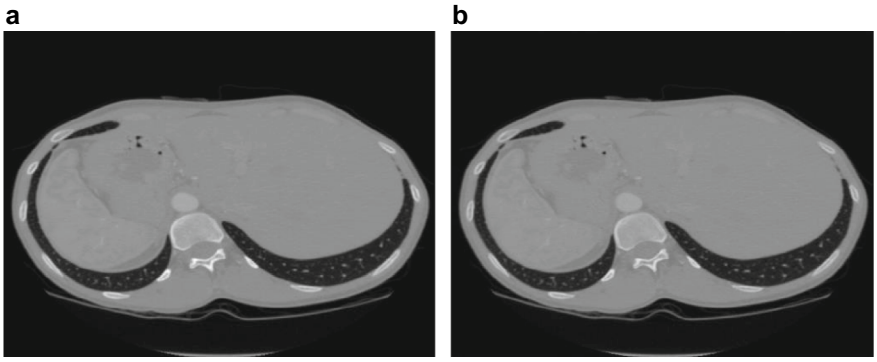


Fig. 7.4 Shows the cover and stego images generated by the two-stage model for visual comparison

7.4.3 Performance Comparison with Other GAN-Based Models

Table 7.2 compares the performance results of the two-stage depth balance approach with other existing GAN-based models using the CelebA dataset. The results presented in Table 2 indicate that the two-stage depth balance approach outperforms existing deep learning models across all evaluation metrics. It is noteworthy that deep learning methods often face challenges in achieving high extraction accuracy as the capacity increases. However, the proposed two-stage depth-balanced method achieves good extraction accuracy even with higher payloads compared to other existing deep learning methods.

Table 7.2 Comparison of two-stage depth balanced model with other GAN-based methods using CelebA dataset

Method	Capacity (bpp)	Extraction accuracy	Steganalyzer	Steganalyzer error (%)
GSIVAT (Hayes and Danezis [3])	0.4	100	Self-defined	21
SsteGAN (Wang et al. [4])	0.4	98.8	Self-defined	40.84
SWE-GAN (Hu et al. [6])	0.0732	89.3	CRM	44
Generative sampling (Zhang et al. [7])	0.05	91	SPAM	<50
Two-stage depth balanced method	1 bpp	99.98	XuNET	50
Two-stage depth balanced method	2 bpp	97.68	XuNET	49

Table 7.3 Comparison of two-stage depth balanced model with hybrid methods using CelebA dataset

Method	Capacity (bpp)	Steganalyzer	Error rate of steganalyzer (%)
SSGAN (Shi et al. [9])	0.4	Self-defined	50
CNN-Adv-Emb (Tang et al. [12])	0.4	Qian's NET	71
Two-stage depth balanced method	1 bpp	XuNET	50
Two-stage depth balanced Method	2 bpp	XuNET	49

7.4.4 Performance Comparison with Hybrid Methods

Hybrid systems blend traditional methods with deep neural networks or GANs to improve steganography. The performance of the two-stage depth-balanced method is compared with hybrid methods in Table 7.3. The two-stage depth-balanced method demonstrates superior performance in terms of capacity. It is worth noting that existing hybrid methods generally operate at a capacity level of around 0.4 bits per pixel (bpp). However, with the two-stage depth-balanced method, as the capacity is increased to more than four times, the robust security measures are still maintained.

7.5 Challenges and Future Directions

Exploring alternative generator architectures and embedding strategies can provide insights into improving steganography capacity while maintaining security and minimizing distortion. Experimentation with different network depths and embedding techniques may lead to more effective steganographic methods. Extending the evaluation of the two-stage depth balanced approach to diverse datasets can provide a comprehensive understanding of their performance across different domains and scenarios. Experimentation with larger datasets and real-world applications can validate the effectiveness and scalability of the two-stage depth-balanced technique.

7.6 Summary

A new two-stage depth-balanced approach for image steganography was discussed in this chapter. The technique generated the stego image in two stages. During the first stage, the cover image was reconstructed which facilitated in generating good quality stego image. During the second stage, the reconstructed cover image was embedded with the secret message to produce the stego image. The results were compared

with other GAN-based models. It was observed that the two-state depth-balanced approach produced better results than other GAN-based steganography methods.

Bibliography

1. J. Mao, Y. Yang, T. Zhang, Empirical analysis of attribute inference techniques in online social network. IEEE Trans. Netw. Sci. Eng. **8**(2), 881–893 (2021). https://doi.org/10.1109/TNSE.2020.3009864
2. M.A. Wani, B. Sultan, Deep learning based image steganography: a review. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 1–26 (2022). https://doi.org/10.1002/widm.1481
3. J. Hayes, G. Danezis, Generating steganographic images via adversarial training. Adv. Neural Inf. Process. Syst. 1955–1964 (2017)
4. Z. Wang, N. Gao, X. Wang, X. Qu, L. Li, *SSteGAN: Self-Learning Steganography Based on Generative Adversarial Networks*, vol. 11302 (Springer International Publishing, LNCS, 2018)
5. M. Yedroudj, F. Comby, M. Chaumont, Yedroudj-Net: an efficient CNN for spatial steganalysis, in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 1 (2018), pp. 2092–2096. https://doi.org/10.1109/ICASSP.2018.8461438
6. D. Hu, L. Wang, W. Jiang, S. Zheng, B. Li, A novel image steganography method via deep convolutional generative adversarial networks. IEEE Access **6**(c), 38303–38314 (2018). https://doi.org/10.1109/ACCESS.2018.2852771
7. Z. Zhang et al., Generative steganography by sampling. IEEE Access **7**, 118586–118597 (2019). https://doi.org/10.1109/ACCESS.2019.2920313
8. W. Tang, S. Tan, B. Li, J. Huang, Automatic steganographic distortion learning using a generative adversarial network. IEEE Signal Process. Lett. **24**(10), 1547–1551 (2017). https://doi.org/10.1109/LSP.2017.2745572
9. H. Shi, J. Dong, W. Wang, Y. Qian, X. Zhang, *SSGAN: Secure Steganography Based on Generative Adversarial Networks*, vol. 10735 (Springer International Publishing, LNCS, 2018)
10. Y. Zhang, W. Zhang, K. Chen, J. Liu, Y. Liu, N. Yu, Adversarial examples against deep neural network based steganalysis, in *IH MMSec 2018 Proceedings of the 6th ACM Workshop on Information Hiding and Multimedia Security* (2018), pp. 67–72. https://doi.org/10.1145/3206004.3206012
11. I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in *3rd International Conference on Learning Representations, ICLR 2015. Conference Track Proceedings* (2015). pp. 1–11
12. W. Tang, B. Li, S. Tan, M. Barni, J. Huang, CNN-based adversarial embedding for image steganography. IEEE Trans. Inf. Forensics Secur. **14**(8), 2074–2087 (2019). https://doi.org/10.1109/TIFS.2019.2891237
13. J. Tan, X. Liao, J. Liu, Y. Cao, H. Jiang, Channel attention image steganography with generative adversarial networks. IEEE Trans. Netw. Sci. Eng. **9**(2), 888–903 (2022). https://doi.org/10.1109/TNSE.2021.3139671
14. B. Sultan, M. ArifWani, A new framework for analyzing color models with generative adversarial networks for improved steganography. Multimed. Tools Appl. (2023). https://doi.org/10.1007/s11042-023-14348-7

Chapter 8
Two-Stage Generative Adversarial Networks for Image Steganography with Multiple Secret Messages

8.1 Introduction

Steganography security can be increased by including multiple secret messages, one of which is a true message and other messages have no significance for a given receiver. The use of multiple messages improves security as it is harder for the intruder to extract secret messages from the stego image.

Shallow neural networks have been used to embed multiple secret messages inside a cover image. However, the results reported in the literature are not very encouraging. Deep CNN-based techniques have been used to embed multiple secret messages inside a cover image. However, there is only one extractor network in these models, which attempts to extract all the secret messages in one go, which imposes restrictions in developing an efficient model for image steganography with multiple messages.

Embedding more than one secret message inside the cover image also decreases the stego image quality. Although image steganography has advanced, it is still challenging to effectively hide multiple secret messages inside a single cover image.

This chapter describes a new approach for image steganography for multiple secret messages. The approach can conceal multiple secret messages for multiple recipients in a cover image. A separate extractor model is used to extract the secret message intended for each receiver. The generator model is developed using a two-stage approach. A flattened cover image is used in the first stage to generate the reconstructed cover image. The output of the first stage is concatenated with the multiple secret messages (intended for multiple receivers) before being input to the second stage of the generator network to produce the stego image.

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_8

8.2 GAN Based Image Steganography System for Multiple Secret Messages

The GAN-based image steganography system for multiple secret messages incorporates multiple decoder networks, each dedicated to decoding a distinct secret message.

8.2.1 Workflow Diagram of Image Steganography Process for Multiple Secret Messages

The workflow diagram of the GAN-based image steganography process for multiple secret messages is given in Fig. 8.1.

The cover image (C) is fed into the Generator network's first stage, where several transformations are applied. The two secret messages (M_1 and M_2) are combined with the output of the first stage and provided as input to the generator's second stage. The combined input is subjected to a series of transformations in the second stage, which creates the stego images (S). The stego image is used in the extractor networks and the steganalyzer network. The task of the extractor1 network is to recover the first secret message so that $M_1' = M_1$ and the extractor2 network is to recover the second secret message so that $M_2' = M_2$. The steganalyzer network distinguishes between the cover image and the stego image. The generator network's first stage is trained on the cover image. The generator network's second stage is

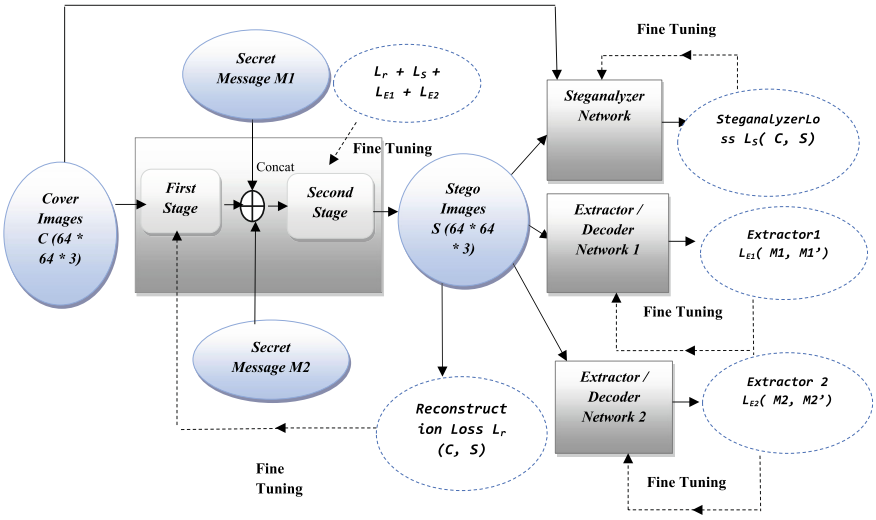


Fig. 8.1 Workflow diagram of image steganography process for multiple secret messages

trained simultaneously against the steganalyzer to learn the two hidden message embeddings. As a result, the generator network can create a stego image that is difficult for the steganalyzer network to distinguish it from the cover image, and the extractor networks can successfully recover the hidden messages from the stego image. Initially, the extractor networks cannot retrieve the secret messages, while the steganalyzer network can easily distinguish between cover images and stego images. As training progresses, the generator learns to create stego images that cannot be distinguished from the cover images, and the secret messages are fully decoded. The generator networks proposed here are described in the following section.

8.2.2 Generator Networks

Four generator networks are described which differ in the depth of the network at which the secret message is embedded. These four networks are the conventional embedding generator network, the early embedding generator network, the mid-embedding generator network, and the late embedding generator network. The architecture of these four generator networks is discussed below.

8.2.2.1 Conventional Embedding Generator Network

The conventional embedding generator network for multiple secret messages is displayed in Fig. 8.2. The cover image (C) is flattened and two different secret messages, M_1 and M_2, are concatenated with it. The generator network's first layer, a fully connected (FC) layer with 8192 elements, receives this concatenated input. 512 feature maps of size 4×4 are created from the FC layer's output after it has been reshaped. The size is increased to $64 \times 64 \times 3$ by up sampling with the four fractionally-strided convolution layers. This generator network uses a single-stage network.

8.2.2.2 Early Embedding Generator Network

The early embedding generator network for multiple secret messages is shown in Fig. 8.3. Only the FC layer of the generator constitutes the first stage. The second stage comprises four fractionally strided convolution layers. The first stage of the network does not use the secret messages M_1 and M_2. The cover image (C) is flattened and input into the first stage of the generator network. The output of the first stage (FC) is reshaped into 256 feature maps of size 4×4. Secret messages M_1 and M_2 of size 4096 each are also reshaped into two tensors of size $4 \times 4 \times 256$ to match the shape of the feature maps. The input for the second stage is prepared by concatenating these two reshaped tensors ($4 \times 4 \times 256$) with the reshaped output of the first stage ($4 \times 4 \times 512$) resulting in 1024 feature maps of size 4×4. The concatenated input

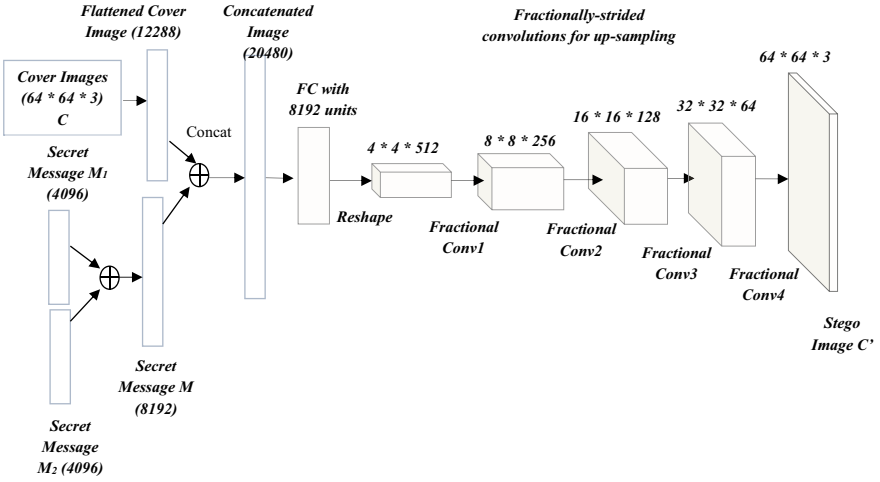


Fig. 8.2 Conventional embedding generator network for multiple secret messages

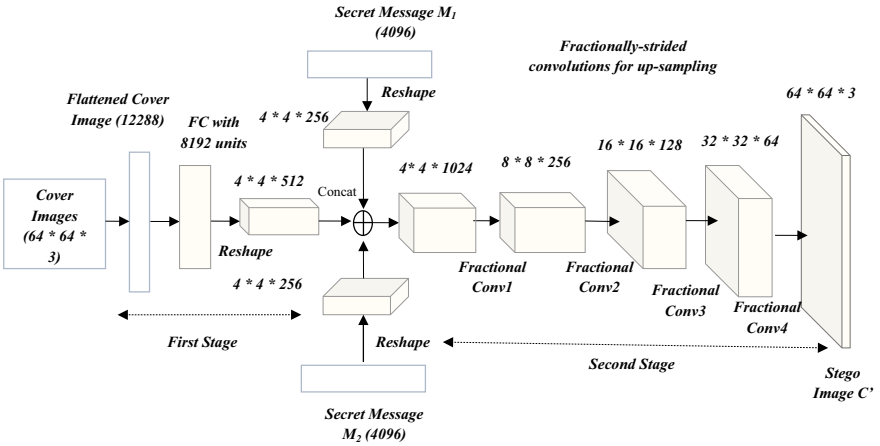


Fig. 8.3 Early embedding generator network for multiple secret messages

is then passed through the layers of the second stage of the generator network to generate the stego image S of size $64 \times 64 \times 3$.

8.2.2.3 Mid-Embedding Generator Network

The mid-embedding generator network for multiple secret messages is depicted in Fig. 8.4. The FC layer and the first fractionally-strided convolution layer constitute the first stage while the last three fractionally-strided convolution layers constitute

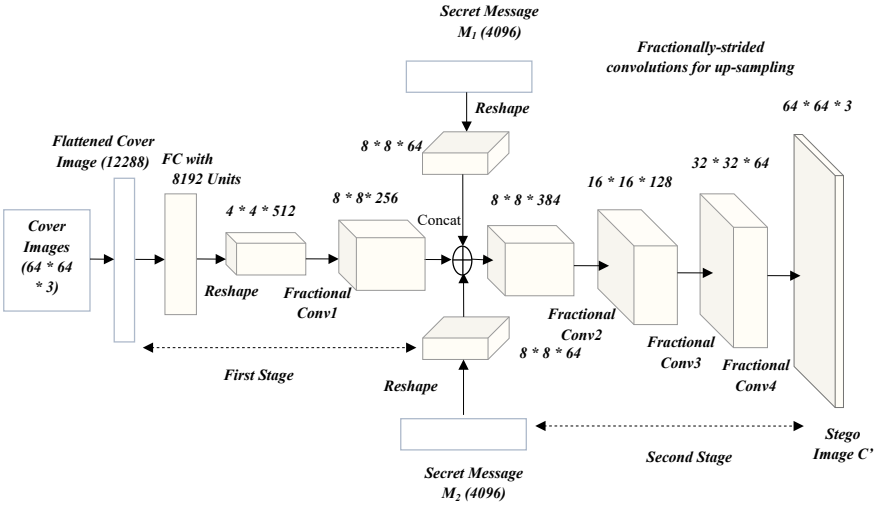


Fig. 8.4 Mid-embedding generator network for multiple secret messages

the second stage of the generator network. The cover image C is flattened and passed through the first stage of the generator network. The output of the first stage is 256 feature maps of size 8 × 8. The secret messages M_1 and M_2 of size 4096 each are reshaped into two tensors of size 8 × 8 × 64 to match the output shape of the first stage. The input for the second stage is prepared by concatenating the two reshaped tensors with the output of the first stage resulting in 384 feature maps of size 8 × 8. The concatenated input is then passed through the second stage of the generator network to generate the stego image S of size 64 × 64 × 3.

8.2.2.4 Late Embedding Generator Network

The late embedding generator network for multiple secret messages is displayed in Fig. 8.5. The FC layer and the first two fractionally-strided convolution layers constitute the first stage while the last two fractionally-strided convolutions constitute the second stage of the generator network. The cover image (C) is flattened and fed into the first stage of the generator network. The output of the first stage is 128 feature maps of size 16 × 16. The secret messages M_1 and M_2 each of size 4096 are reshaped into two tensors of size 16 × 16 × 16 each to match the output shape of the first stage. The input for the second stage is prepared by concatenating these reshaped tensors with the output of the first stage, resulting in 160 feature maps of size 16 × 16. This concatenated input is then passed through the second stage of the generator network to generate the stego image S of size 64 × 64 × 3.

The generator networks employ ReLU activation and batch normalization in all layers except the last layer, which only uses tanh activation.

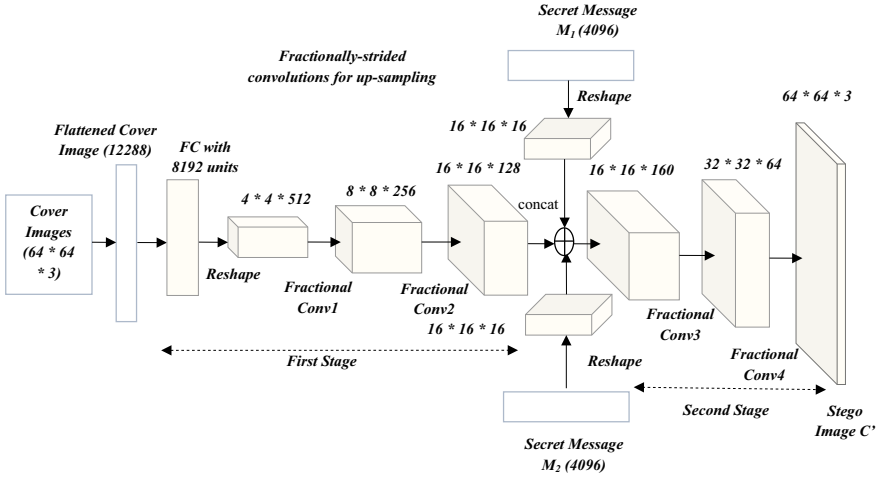


Fig. 8.5 Late embedding generator network for multiple secret messages

8.2.3 Steganalyzer Network

The steganalyzer network consists of four convolution layers followed by a fully connected layer. The input to the network is the cover image (C) or the stego image (S), which is down-sampled from $64 \times 64 \times 3$ to $4 \times 4 \times 256$ by a series of convolution layers. The fully connected layer has one output unit that provides the likelihood that the input is a cover image or stego image. A probability value close to 1 denotes that the input is the cover image, while a value close to 0 denotes a stego image. The network employs leaky ReLU activation and batch normalization in all layers except the last layer, which uses sigmoid activation.

8.2.4 Extractor Network

The architecture of the extractor network is similar to the steganalyzer except that its fully connected layer has several elements equal to the message size. Instead of sigmoid activation, it employs tanh activation.

8.2.5 Algorithm for Training the Networks

The algorithms used for training the networks are discussed in this section.

8.2.5.1 Algorithm for Training the Generator Network

The generator network receives the cover image and secret messages to be embedded to produce the stego image. Algorithm 1 gives the steps to generate the stego image.

Algorithm 1. Training the Generator Network

Input: *Secret Message M_1, Secret Message M_2, Cover image C,*

Output: *Stego image S, Recovered Message M_1', Recovered Message M_2'*

1. *Select a Random sample C from the cover image Dataset.*
2. *Flatten C and input it into the first stage of the generator network.*
3. *Concatenate the secret message M_1 and M_2 with the output of the first stage of the generator network.*
4. *The concatenated input is passed through the second stage of the generator network to produce the stego image S.*
5. *Calculate the reconstruction error between C and C_R, back-propagate the error through the first stage of the generator network, and update the trainable parameters (θ_{GF}) of the generator network.*
6. *Calculate the weighted sum of steganalyzer error, extractor1 and extractor2 error, and reconstruction error between C and C_R, back-propagate the errors through the second stage of the generator network, and update the trainable parameters (θ_{GS}) of the generator network.*

8.2.5.2 Algorithm for Training the Steganalyzer Network

The steganalyzer network's task is to differentiate between the cover image and the generated stego image. Algorithm 2 gives the steps for this task.

Algorithm 2. Training the Steganalyzer Network Network

Input: *Cover image C, Stego Image S, Secret Message M_1, Secret Message M_2*

Output: *Probability value whether the input Image is Cover Image or stego Image*

1. *Select a Random sample C from the cover image Dataset.*
2. *Flatten C and input it into the first stage of the generator network.*
3. *Concatenate the secret message M_1 and M_2 with the output of the first stage of the generator network.*
4. *The concatenated input is passed through the second stage of the generator network to produce the stego image S.*
5. *Pass C and S through the Steganalyzer to predict Cover and Stego.*
6. *Calculate the classification errors and back-propagate the total error to update the trainable parameters (θ_s) of the Steganalyze..*

8.2.5.3 Algorithm for Training the Extractor Networks

Each extractor network receives the stego image (S) and outputs the extracted secret message. Algorithm 3 gives the steps for this task. The algorithm is repeated for each extractor network.

Algorithm 3 Training the Extractor Networks

Input: *Secret Message M_1, Secret Message M_2, Cover image C, Stego image S*

Output: *Recovered Secret Message M_1'or M_2'*

1. *Select a random sample C from the cover image Dataset.*
2. *Flatten C and input it into the first stage of the generator network.*
3. *Concatenate the secret messages M_1 and M_2 with the output of the first stage of the generator network.*
4. *The concatenated input is passed through the second stage of the generator network to produce the stego image S.*
5. *Pass S through the Extractor Network. The Extractor recovers the secret message M_1' or M_2'.*
6. *Calculate the error between M_1 and M_1' (or M_2 and M_2'), back-propagate the error, and update the trainable parameters (θ_{E1}) of the extractor network.*

8.3 Results and Discussions

The experimental design and dataset used here are discussed in this section. The performance of the proposed models is compared to that of other GANs-based methods described in the literature.

8.3.1 Dataset Used

The CelebA dataset containing 200 k images of celebrity faces is used here for a comparison of proposed models with other models. The images were reduced to 64 $*$ 64 size for experimental purposes. The proposed models were trained using a set of 60 k images chosen randomly. With a batch size of 32, the stochastic gradient descent approach was used to train the generator, steganalyzer, and extractor networks. With the learning rate set to 0.0002, the model was optimized using Adam's optimizer. Tensorflow 1.15.0 with the DGX A100 GPU was used to conduct the experiments.

8.3.2 *Performance of Generator Models*

All four generator models were trained for a capacity of 2 bits per pixel (bpp), allowing 1 bpp for each secret message. The performance of the four generator models is shown in Table 8.1, which shows the accuracy for the extraction of secret messages and distortion measurement of these models.

It can be seen from Table 8.1 that the late embedding generator model outperforms the other three models on all the evaluation metrics. The extraction accuracy is also significantly increased with the late embedding model compared to the other three generator models. The late embedding model is capable of embedding 2bpp without deteriorating the other evaluation metrics. As the capacity is increased, the extractor is not able to successfully extract the secret message.

To assess the security of the steganography, the steganalyzer network (Xu's NET) was trained with 4000 images, and 2000 images (1000 cover and 1000 stego) were utilized for validation, while 500 cover and 500 stego images were used for testing. Table 8.2 lists the steganalyzer error rates of the four generator models. A high error rate denotes higher security because the model can conceal the secret data without being noticed by steganalyzer. The late embedding generator model has the highest error rate compared to other models.

Table 8.3 shows the results of changing the embedding capacity from 1 to 3 bpp. As the embedding capacity is increased from 1 to 2 bpp, the image quality depicted

Table 8.1 Performance comparison of the four generator models

Evaluation criteria	Generator model			
	Conventional embedding generator model	Early embedding generator model	Mid embedding generator model	Late embedding generator model
Capacity	2 bpp	2 bpp	2 bpp	2 bpp
PSNR	73.81	74.30	74.20	74.32
SSIM	0.9351	0.9490	0.9349	0.9494
Accuracy (Extractor 1)	70.58	72.78	89.84	99.93
Accuracy (Extractor 2)	50.22	72.68	96.80	99.99

Table 8.2 Security comparison of the four generator models for 2 bpp

Generator model	# Of images misclassified	Error rate of steganalyzer (%)
Conventional embedding generator model	440	44
Early embedding generator model	470	47
Mid embedding generator model	289	28.90
Late embedding generator model	500	50

Table 8.3 Performance of late embedding generator model with different payloads

Evaluation criteria	Capacity (Late embedding generator model)		
	1 bpp	2 bpp	3 bpp
PSNR	77.04	74.32	75.11
SSIM	0.9658	0.9494	0.9457
Accuracy % (Extractor 1)	99.99	99.93	50.13
Accuracy % (Extractor 2)	99.99	99.99	95.88
steganalyzer error rate (%)	50	50	22.1

by PSNR drops slightly from 77.04 to 74.32 but the extraction accuracy and security are still good. However, when the capacity is increased to 3bpp, it is observed that all metrics degrade. This is because as embedded information increases, images become noisier, lowering the quality of the images.

The visual comparison of cover images and stego images generated by the Late Embedding Generator model for the capacity of 1 and 2 bpp values is given in Table 8.4.

Table 8.4 Cover and stego images of late embedding generator model for capacity of 1 and 2 bpp values

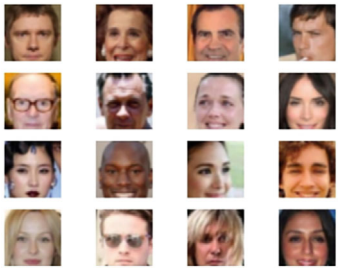
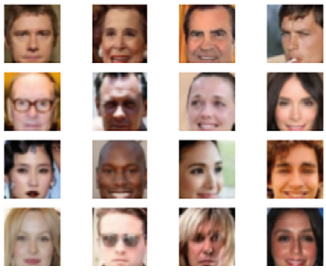
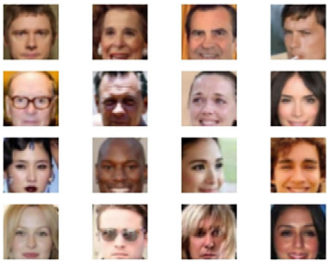
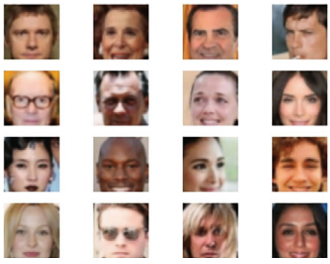
Capacity	Cover images	Stego images
1 bpp	Block 1(a) 	Block 1(b) 
2 bpp	Block 2(a) 	Block 2(b) 

Table 8.5 Performance of late embedding generator model with other steganography methods

Model	Capacity (bpp)	SSIM	PSNR
Multi-image steganography (Sharma et al. [13])	0.114	–	68.67
RSM (Jarusek et al. [12])	0.0015	0.9744	30
Hiding images within images (Baluja [2])	2	0.98	41.2
Multi-data steganography (Sultan and Wani [3])	0.2	0.89	69.54
Late embedding model	1	0.9658	77.04
Late embedding model	2	0.9494	74.32

8.3.3 Performance of Late Embedding Generator Model and Steganography Methods

The performance of the late embedding generator model and other steganography methods described in the literature is given in Table 8.5. It can be seen that the late embedding generator model outperforms the other steganography methods in terms of invisibility and capacity metrics.

8.3.4 Performance of the Late Embedding Generator Model and Deep Learning Models

The performance of the late embedding generator model and deep learning models described in the literature on the CelebA dataset is given in Table 8.6. It can be seen that the late embedding generator model outperforms the other deep learning models. The other deep learning models mostly have problems in achieving good extraction accuracy with increased capacity.

8.4 Challenges and Future Directions

Embedding multiple secret messages in a cover image, while enhancing steganographic security, faces significant challenges, such as maintaining the stego image's quality, managing complex encryption, and designing efficient extraction networks. Current approaches, including shallow neural networks and deep CNNs models, struggle with these issues. Future research can focus on developing better multi-stage neural network architectures that can balance the trade-off between capacity and image quality while enhancing the reliability of message extraction. Additionally, there is a need to improve robustness against steganalysis and explore hybrid approaches that combine steganography with advanced cryptographic protocols.

Table 8.6 Performance of late embedding generator model and deep learning models

Model	Capacity (bpp)	Extraction accuracy	Steganalyzer	Error rate of steganalyzer (%)
GSIVAT (Hayes and Danezis [4])	0.4	100	Self-defined	21
SsteGAN (Wang et al. [8])	0.4	98.8	Self-defined	40.84
SWE-GAN (Hu et al. [11])	0.0732	89.3	CRM	44
Generative sampling (Zhang et al. [1])	0.05	91	SPAM	<50
Late embedding generator model	1 bpp	99.99	XuNET	50
Late embedding generator model	2 bpp	97.96	XuNET	50

8.5 Summary

This chapter addressed the challenges of embedding multiple secret messages within a cover image, primarily due to the degradation of stego image quality and the complexity of accurately extracting the secret messages. Four generator models using traditional, early, mid, and late embedding were discussed and their performances were compared. The late embedding generator model produced the best results. The performance of the late embedding generator model was also compared with other steganography methods and deep-learning models used for steganography. The late embedding generator model outperformed all these models.

Bibliography

1. Z. Zhang et al., Generative steganography by sampling. IEEE Access **7**, 118586–118597 (2019). https://doi.org/10.1109/ACCESS.2019.2920313
2. S. Baluja, Hiding images within images. IEEE Trans. Pattern Anal. Mach. Intell. **42**(7), 1685–1697 (2020). https://doi.org/10.1109/TPAMI.2019.2901877
3. B. Sultan, B., M.A. Wani, Multi-data image steganography using generative adversarial networks, in *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)* (2022), pp. 454–459
4. J. Hayes, G. Danezis, Generating steganographic images via adversarial training. Adv. Neural Inf. Process. Syst. 1955–1964 (2017)
5. M. Yedroudj, F. Comby, M. Chaumont, Steganography using a 3-player game. J. Vis. Commun. Image Represent. **72**(July), (2020). https://doi.org/10.1016/j.jvcir.2020.102910.
6. H. Kweon, J. Park, S. Woo, D. Cho, Deep multi-image steganography with private keys. Electronics **2021**, 10 (1906). https://doi.org/10.3390/electronics10161906
7. A. Das, J.S. Wahi, M.S. Anand, Y. Rana, Multi-image steganography using deep neural networks (2021). https://arxiv.org/abs/2101.00350

8. Z. Wang, N. Gao, X. Wang, X. Qu, L. Li, *SSteGAN: Self-Learning Steganography Based on Generative Adversarial Networks*, vol. 11302 (Springer International Publishing, LNCS, 2018)
9. G. Xu et al., Structural design of convolutional neural networks for steganalysis. IEEE Signal Process. Lett. **23**, 708–712
10. M.A. Wani, B. Sultan, Deep learning based image steganography: a review. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 1–26 (2022). https://doi.org/10.1002/widm.1481
11. D. Hu, L. Wang, W. Jiang, S. Zheng, B. Li, A novel image steganography method via deep convolutional generative adversarial networks. IEEE Access **6**(c), 38303–38314 (2018). https://doi.org/10.1109/ACCESS.2018.2852771
12. R. Jarusek, E. Volna, M. Kotyrba, Robust steganographic method based on unconventional approach of neural networks. Appl. Soft Comput. J. **67**, 505–518 (2018). https://doi.org/10.1016/j.asoc.2018.03.023
13. H. Sharma, D.C. Mishra, R.K. Sharma, N. Kumar, Multi-image steganography and authentication using crypto stego techniques. Multimed. Tools Appl. **80**(19), 29067–29093 (2021). https://doi.org/10.1007/s11042-021-11068-8

Chapter 9
Deep Learning in Healthcare and Computational Biology

9.1 Introduction

Deep learning has demonstrated remarkable efficacy in advancing the field of healthcare and bioinformatics. It has made significant contributions across diverse areas, encompassing sequence analysis, structure prediction, biomolecular property prediction, biomedical image processing, and biomolecule interaction prediction. In sequence analysis, deep learning has been employed to predict the impact of non-coding sequence variants, model transcription factor binding affinity landscapes, enhance DNA sequencing, and analyze DNA sequence modifications. For structure prediction, it predicts protein secondary structures, models protein interactions, and accelerates fluorescence microscopy super-resolution. Deep learning excels in predicting bio-molecular properties, such as enzyme functions and protein sub-cellular locations. In biomedical image processing and diagnosis, deep learning methods have achieved significant success in disease detection and prediction from images such as prediction of brain tumors through the analysis of MRI scans. In biomolecule interaction prediction and systems biology, deep learning models hierarchical cell structures and predicts drug-target interactions. The success of deep learning in bioinformatics is attributed to both increased computational capacity and enhanced algorithms. Crucially, the abundance of biological data, once perceived as a challenge, aligns well with the capabilities of deep learning. Notably, deep learning excels in handling sequence data (DNA, RNA, and protein sequences) and 2D/tensor-like data (biomedical images and gene expression profiles). Deep learning's effectiveness stems from its ability to handle raw data without extensive feature extraction, performing end-to-end feature extraction and classification automatically.

This chapter delves into the intricate realms of deep learning for protein secondary structure prediction, PAN-cancer classification using gene expression data, and brain tumor prediction via the analysis of MRI scans. Each application represents a distinct facet, covering sequence data, gene expression data, and image data. This targeted

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_9

exploration aims to highlight the nuanced contributions of deep learning in addressing critical challenges at the intersection of bioinformatics and medical diagnostics.

9.2 Deep Learning for Sequence Data: A Case Study of Protein Secondary Structure Prediction

Protein secondary structure prediction (PSSP) is a key area of study within computational biology and bioinformatics. It is essentially a sequence-to-sequence classification problem where the input is a protein's amino acid sequence, and the output is the projected secondary structure (SS) sequence. Each residue in the protein is assigned a secondary structure label. Over the past several decades, this area of research has grown significantly, making substantial strides in accuracy. The importance of PSSP lies in its ability to enhance our understanding of protein structure and function and its pivotal role in predicting protein tertiary structure. After all, SS elements significantly influence protein stability, folding, and interaction with other molecules. Although protein structures can be manually determined in experimental labs using various techniques like X-ray crystallography, this process can be slow, difficult, and costly. Given the exponential growth in the number of protein sequences, there is an evident gap as the number of solved structures is considerably lower. As of January 2023, the number of determined structures according to PDB statistics was 203,084. This gap necessitates the use of computational methods as an alternate route for predicting protein structures. While significant research has been devoted to predicting the three-dimensional (3D) structures of proteins from primary sequences, this task remains one of the most challenging in computational biology and bioinformatics. Therefore, significant focus has been to develop methods to predict certain aspects of the structure with increasing accuracy, such as identifying regular spatial arrangements of amino acids like alpha helices and beta sheets.

PSSP is an important area of study in protein research due to its impact on stability and function. It has garnered considerable attention as it serves as a crucial stepping stone in predicting 3D structures. The objective of this prediction is to infer the SS of proteins, given the knowledge of their primary structure, i.e., their amino acid sequence. The assembly and organization of SSs within the protein significantly influence its final conformation. The general outline of the PSSP problem can be viewed in the graphical abstract depicted in Fig. 9.1.

To pave the way for the development of AI-based prediction methods, it is of utmost importance to meticulously annotate protein sequences with their respective SSs. These SSs are derived experimentally utilizing a multitude of techniques. Of these, the Dictionary of SS of Proteins (DSSP) is preeminent, serving as the standard tool for attributing SS annotations to amino acids. This algorithm employs hydrogen bond energy calculations and assigns three states (Q3) which can be further expanded to eight states (Q8) (Kabsch et al. 1983) (see Table 9.1). DSSP remains the standard technique for assigning SSs to amino acids using experimental 3D coordinates.

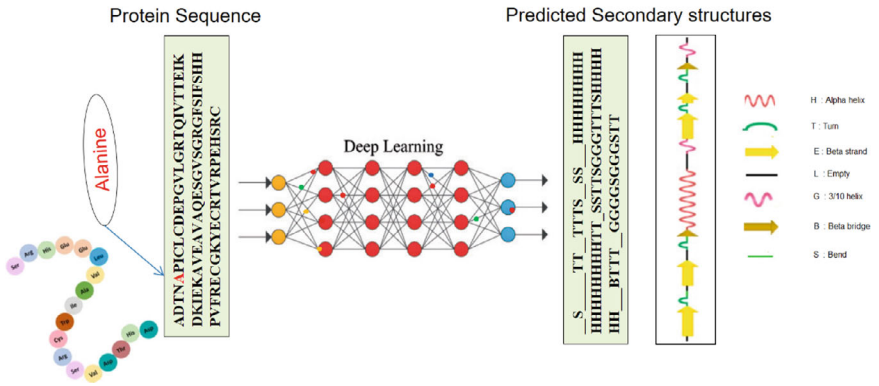


Fig. 9.1 Graphical representation of PSSP problem

Table 9.1 Three state and eight-state classification of protein SSs

SS	3-State	8-State
Alpha-helix (α)	H	H
Beta-bridge	E	B
3_{10} helix	H	G
Pi-helix (π)	H	I
Extended-strands (β)	E	E
Turn	C	T
Bend	C	S
Irregular loop/coil/unknown	C	L

Protein SSs can be used to provide valuable information for predicting the protein's three-dimensional structure. Moreover, it can help to identify specific regions within the protein that are important for its function, which can be used to guide experiments where specific amino acids are mutated to study the impact on the protein's function. Thus, accurately predicting the SS of a protein can have a significant impact on predicting its three-dimensional structure, solvent accessibility (how easily it can be accessed by other molecules). Remarkably, protein SSs reduces the degree of freedom in a protein which helps to limit the possible ways in which the protein can fold, leading to more accurate predictions of the protein's final, three-dimensional structure. Three-state SS prediction categorizes residues into alpha-helix, beta-sheet, or coil, providing a simplified view. In contrast, eight-state prediction captures more detailed structural features within alpha-helices and beta-sheets, offering a higher-resolution representation. Figure 9.2 illustrates the 3-state and 8-state SS of the 1AKD protein from the Protein Data Bank (PDB) (Berman 2002).

Deep learning has had a significant impact on PSSP, offering improved accuracy and speed. Traditional methods relied on statistical and machine learning techniques,

Fig. 9.2 3-State and 8-state secondary structures; Q_3 (left), Q_8 (right) of 1AKD protein

focusing mainly on 3-state prediction and achieving an average accuracy of around $75 \pm 5\%$. Deep learning models, such as recurrent neural networks (RNN) and convolutional neural networks (CNN), have outperformed these traditional methods. The availability of large datasets, including the Protein Data Bank (PDB) and SS annotation databases, has been instrumental in training deep learning models. RNNs are adept at modeling long-range interactions in sequential protein data, while CNNs capture important local sequence patterns and residue-residue interactions. Deep learning models have also been developed for 8-state SS prediction, achieving higher accuracy. These advancements can be attributed to factors such as large datasets, feature learning from raw input data, and modeling of complex relationships between sequence and structure. Incorporating additional features like PSSM, conservation score, and physical properties has further enhanced prediction accuracy.

In summary, deep learning has revolutionized PSSP, allowing for more accurate and detailed predictions than traditional methods. However, there are still several challenges and issues that need to be addressed in order to further improve the accuracy and reliability of deep learning models for this task.

9.3 Deep Learning for Genomic Data: A Case Study of Pan-Cancer Classification

Cancer, a multifaceted disease, is characterized by unbridled cell proliferation and its potential to metastasize. Various etiological agents, both genetic and environmental, contribute to cancer onset and progression. Genetic underpinnings, especially mutations, significantly influence an individual's predisposition to specific cancer types. Such genetic aberrations, predominantly affecting genes regulating mitosis, foster uncontrollable cellular division, leading to tumor genesis. If these tumors acquire malignant traits, they metastasize, jeopardizing human health. The intricate nature of these genetic alterations encompasses somatic mutations (SM), copy number variations (CNV), and a plethora of epigenetic modifications. Furthermore, gene expression alterations, instigated by environmental triggers or hereditary factors, play a paramount role in tumorigenesis. Inherited genetic mutations, passed from generation to generation, can notably elevate cancer risks. For instance, BRCA1

and BRCA2 gene mutations amplify the likelihood of breast and ovarian cancers, whereas Lynch syndrome genes are associated with escalated risks of colorectal and various other cancers. Alterations in gene expression can sometimes modify protein synthesis patterns, consequently disrupting normal cellular functions. Such aberrant cells undergo rapid division, resulting in tumorous growths within the affected region. Therefore, delineating these genetic alterations is pivotal as they hold the promise for targeted therapeutic strategies.

9.3.1 Pan-Cancer Classification: A Paradigm Shift

Historically, cancers were classified based on their anatomical origin. However, with the advent of molecular biology and genomics, there's a paradigm shift towards a more holistic classification: the pan-cancer classification. This innovative approach seeks to categorize cancers based on their genetic and molecular signatures, sidelining the conventional organ-centric view. Such a classification offers profound insights, illuminating shared molecular themes across different cancer types, thus refining therapeutic strategies and bolstering personalized medicine's efficacy.

The monumental Pan-Cancer Atlas project, spearheaded by TCGA, sought to meticulously analyze and chronicle the genetic similarities and disparities across diverse cancer types. With the advent of Next Generation Sequencing, the scrutiny of human genomics has reached unprecedented levels of precision and efficiency. TCGA harnessed this power to sequence an impressive repertoire of tumor tissues, eventually analyzing over 11,000 tumors from 33 predominant cancer forms. The culmination of these Herculean efforts is the Pan-Cancer Atlas, a goldmine of information for researchers worldwide. However, the voluminous data generated, especially by whole-genome sequencing, presents an analytical challenge. Traditional manual and experimental methodologies for cancer classification are not only time-consuming but are susceptible to human error, especially in the backdrop of the multifactorial nature of cancer. As cancer research evolves, the complexity of data interpretation escalates, rendering manual methods less feasible and more error-prone. Consequently, there's a pressing need for sophisticated prediction methods for pan-cancer classification. Here, deep learning, a subset of machine learning, emerges as a promising solution. By training on expansive genetic and molecular datasets, deep learning algorithms can classify new samples with remarkable accuracy. These algorithms can discern intricate patterns in gene expression data, often elusive to manual inspection, paving the way for an enhanced and precise pan-cancer classification paradigm.

The integration of computational techniques in pan-cancer classification has emerged as a promising avenue, with the potential to refine clinical outcomes. This is achieved by facilitating enhanced diagnostic precision, tailoring treatments, and unearthing novel therapeutic avenues. Consequently, a myriad of machine learning (ML) methodologies has been employed to address challenges related to cancer detection and prognosis using gene-expression data. However, a notable limitation of

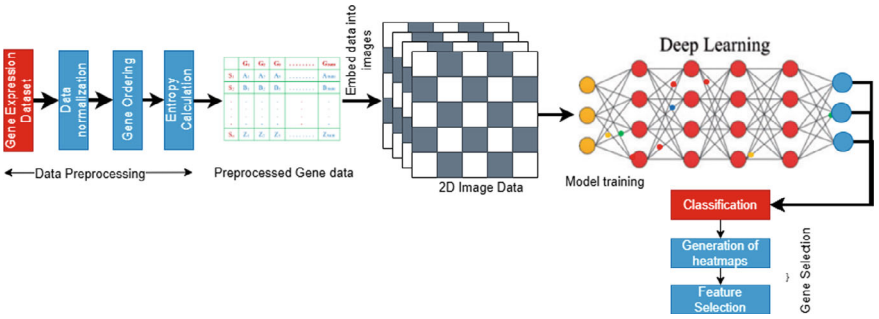


Fig. 9.3 Overview of pan-cancer classification using deep learning

traditional classification strategies is their restricted focus on a limited set of cancer types, often neglecting the inherent heterogeneity present across diverse cancers. Classical ML approaches, when applied to pan-cancer classification, frequently falter in the face of high-dimensional data, compromising output accuracy. These techniques are plagued by challenges including data scarcity, imbalance, overfitting, and restricted interpretability, which collectively impede the efficacy and reliability of subsequent models. Drawing inspiration from the commendable successes of deep learning in realms such as image, text, and sequence classifications, methodologies encompassing Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) have gained traction in the sphere of cancer predictions from gene expression datasets. The overall framework is shown in Fig. 9.3. Notably, ensemble techniques stand out as a robust solution for pan-cancer classification within the deep learning paradigm. By synergizing predictions from multiple models, ensemble approaches not only enhance model performance and resilience but also fortify their interpretative capabilities.

9.4 Deep Learning for Image Data: A Case Study of Tumor Prediction Using MRI Scans

Brain tumors pose a substantial challenge in the medical field due to their intricate and potentially life-threatening nature. In the United States alone, the diagnosis of malignant brain tumors in 2022 reached 25,050 cases, highlighting the severity of this condition. Despite being a relatively small subset among primary central nervous system (CNS) tumors, brain tumors account for a significant percentage, ranging from 85–90%, emphasizing their clinical importance. On a global scale, an estimated 308,102 individuals were diagnosed with primary brain or spinal cord tumors in 2020, with around 4,170 cases identified among children under 15 years old, accentuating the widespread impact of this medical issue. Hence, early detection of brain tumors plays a pivotal role in improving treatment outcomes and patient

prognosis. Advanced imaging techniques, particularly Magnetic Resonance Imaging (MRI), serve as indispensable tools in detecting these tumors by providing intricate and detailed images of the brain. These imaging modalities are instrumental in identifying the location, size, and characteristics of brain tumors, aiding in their accurate diagnosis. The most common brain tumor classes are Glioma, Meningioma, Pituitary, Medulloblastoma, Schwannoma and Pineal Tumors. Among all types, Glioma, Meningioma, and Pituitary are prevalent, diverse in characteristics, clinically significant, and present distinct diagnostic challenges. Glioma is a broad category covering various types of tumors that start in the glial cells of the brain. These tumors are known for their infiltrative nature and can occur at any age. Meningiomas, arising from the meninges, the protective layers covering the brain and spinal cord are typically slow-growing tumors. They can cause symptoms based on their size and location. Pituitary tumors develop in the pituitary gland, a small gland at the base of the brain. They can affect hormone levels and lead to various health issues due to hormonal imbalances. Focusing on three primary brain tumor classes, Glioma, Meningioma, and Pituitary tumors, is a strategic approach in medical research and diagnostics. By concentrating efforts on these common tumors, researchers gain a deeper understanding of their complexities, aiding in precise diagnosis and specific treatment approaches. The brain tumor dataset from figshare containing 3064 T1-weighted contrast-inhanced images from 233 patients with three kinds of brain tumor. The samples of the three classes are shown in Fig. 9.4. This focused research allows for more detailed investigations into their biology, genetics, and therapeutic options.

As shown in Fig. 9.5, the process starts with an input image, which is an MRI scan of the brain. This image is typically pre-processed to ensure that it is in a format that the deep learning algorithm can understand. The pre-processed image is then passed through a series of convolutional layers. Each convolutional layer applies a filter to the image, which extracts features from the image. The activation function then determines which of these features are important. This process is repeated multiple times, with each layer learning more complex features from the previous layer. Pooling layers are used to reduce the dimensionality of the data. This is important because it helps to prevent the deep learning algorithm from overfitting to the training data. There are different types of pooling layers, but they all work by summarizing the information in a small region of the image into a single value. After the final convolutional layer, the data is flattened into a single vector. This vector is then fed into a fully connected layer, which is a type of artificial neural network that makes the final prediction. The output of the fully connected layer is a probability score that indicates the likelihood that the image contains a tumor. This score can be used to make a binary classification (tumor or no tumor) or to predict the type and grade of the tumor.

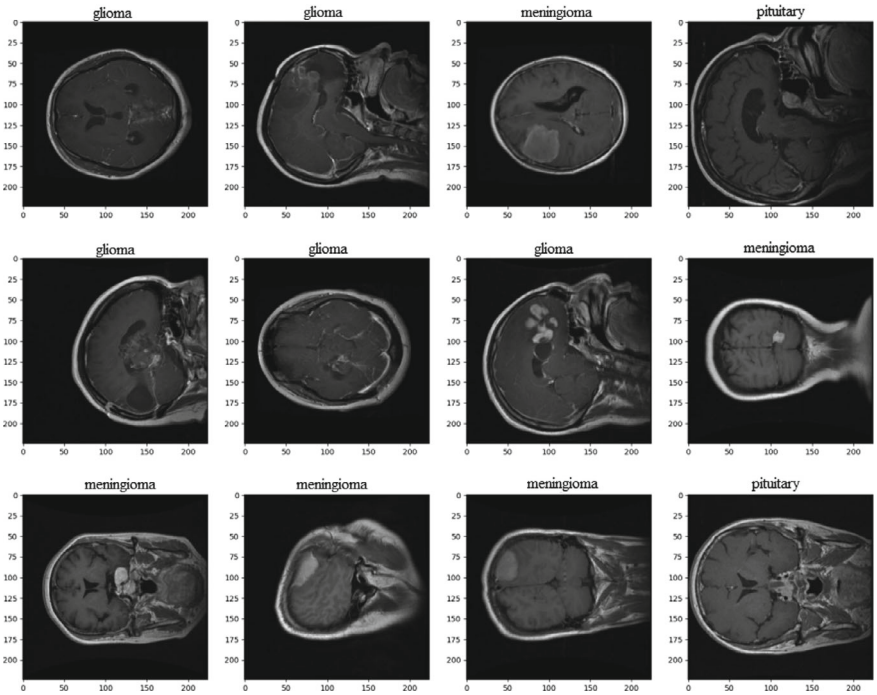


Fig. 9.4 Glioma, Meningioma, and Pituitary tumor samples

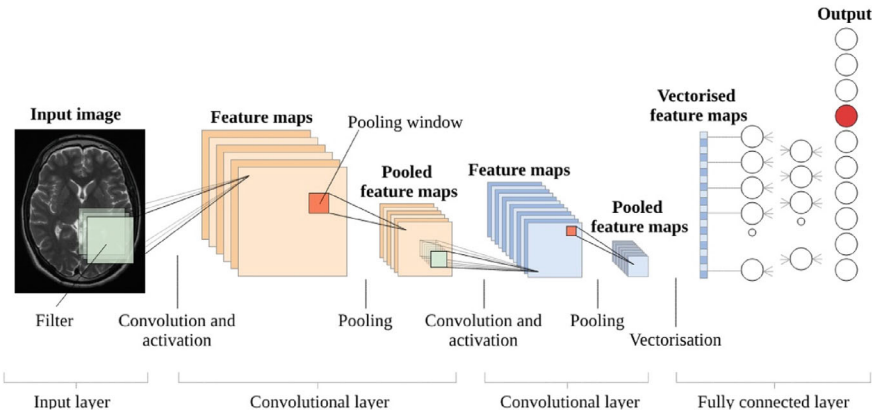


Fig. 9.5 Graphical representation of brain-tumor prediction using deep learning

9.5 Challenges and Future Directions

In the realm of Deep Learning (DL) applications within healthcare and computational biology, we're confronted with a series of challenges that span across protein secondary structure prediction (PSSP), pan-cancer classification using gene expression data, and brain tumor prediction through MRI scans analysis. These challenges underscore the complexity and dynamic nature of biological systems and the need for advanced computational methodologies to decipher them effectively.

A primary challenge in the field is the accurate prediction and classification of protein secondary structures (PSSP) from amino acid sequences. This task is complicated by the vast diversity and complexity of proteins, making it difficult to capture the intricate patterns and interactions that dictate protein folding and function. Developing hybrid deep learning architectures that can integrate multiple types of data and leverage the strengths of various DL models is crucial for improving prediction accuracy and understanding protein dynamics on a deeper level. In the domain of pan-cancer classification, a significant hurdle is the heterogeneity and high dimensionality of cancer genomic data. Traditional DL models often struggle to cope with the vast array of genetic alterations that characterize different cancer types. An ensemble approach that combines multiple DL models could potentially enhance the ability to classify cancers more accurately. However, this introduces the challenge of effectively integrating diverse models to handle the complexities of cancer genomics, requiring innovative strategies to balance the contributions of different models and manage the computational complexity. For brain tumor prediction using MRI scans, the challenge lies in processing and analyzing high-resolution imaging data efficiently while maintaining high accuracy in tumor detection and classification. The adoption of transfer learning with lightweight architectures presents a promising avenue, yet optimizing these models for medical imaging tasks and ensuring they are adaptable to the nuances of brain tumor characteristics requires careful consideration. This challenge is compounded by the need for models that can operate efficiently on limited computational resources, making it accessible for widespread clinical use.

9.6 Summary

In this chapter, we have delved into the multifaceted applications of deep learning within the realms of bioinformatics and medical diagnostics. Through a targeted exploration, we have highlighted the profound impact of deep learning in addressing critical challenges across three distinct areas: protein secondary structure prediction, PAN-cancer classification using gene expression data, and brain tumor prediction through the analysis of MRI scans. The first application focuses on leveraging deep learning techniques for protein secondary structure prediction, demonstrating the ability to unravel complex sequence data with high accuracy. Next, we explored the realm of PAN-cancer classification, where deep learning algorithms analyze

gene expression data to classify different cancer types, showcasing the potential for personalized medicine and treatment strategies. Finally, we examined the use of deep learning in brain tumor prediction via the analysis of MRI scans, highlighting its role in aiding clinicians in early detection and treatment planning.

Each of these case studies represents a distinct facet of the intersection between deep learning, bioinformatics, and medical diagnostics. In the subsequent chapters, we delve deeper into each of these applications, showcasing the specific deep learning methodologies employed and how they have contributed to improving performance metrics such as accuracy, sensitivity, and specificity. Through these detailed explorations, we aim to underscore the nuanced contributions of deep learning in revolutionizing the field of bioinformatics and medical diagnostics, paving the way for more efficient and effective disease detection, diagnosis, and treatment.

Bibliography

1. B. Alberts et al., Molecular biology of the cell. Biochem. Mol. Biol. Educ. **36**(4), 317–318 (2008). https://doi.org/10.1002/bmb.20192
2. H.M. Berman, The protein data bank. Acta Crystallogr. Sect. D Biol. Crystallogr. **58**(6 I), 899–90 (2002). https://www.rcsb.org
3. Q. Jiang, X. Jin, S.J. Lee, S. Yao, PSSP: a survey of the state of the art. J. Mol. Graph. Model. **76**, 379–402 (2017). https://doi.org/10.1016/j.jmgm.2010015
4. Evaluation of machine learning algorithm utilization for lung cancer classification based on gene expression levels. Asian Pac. J. Cancer Prev. **17**(2), 835–838. https://doi.org/10.7314/APJCP.2016.12.835
5. Z. Wang, M.A. Jensen, J.C. Zenklusen, A practical guide to the cancer genome atlas (TCGA), in *Statistical Genomics* (Humana Press, New York, 2016), pp. 111–141. https://doi.org/10.1007/978-1-4939-3578-9_6
6. E.H. Yau, I.R. Kummetha, G. Lichinchi, R. Tang, Y. Zhang, T.M. Rana, Genome-wide CRISPR screen for essential cell growth mediators in mutant KRAS colorectal CancersGenome-wide CRISPR screen of KRAS-mutant tumor xenografts. Can. Res. **77**(22), 6330–6339 (2017). https://doi.org/10.1158/0008-5472.can-17-2043
7. M.A. Sofi, M.A. Wani, IRNN-SS: deep learning for optimised protein secondary structure prediction through PROMOTIF and DSSP annotation fusion. Int. J. Bioinfor. Res. Appl. **20**(6), 608-626 (2024).

Chapter 10
Selected Deep Learning Architectures for Medical Applications

10.1 Introduction

Deep Learning has become a highly popular technology in the last decade due to its ability to process massive amounts of data with state-of-the-art computational resources. Deep Learning has tremendous potential to be used in medical applications. Architectures such as Convolutional neural networks (CNN) and Recurrent neural networks (RNN) have achieved significant success in tasks such as image classification, text classification, and sequence-to-sequence classification (Jurtz et al. 2017). In this chapter, we discuss selected deep learning architectures that are commonly used for sequence and image data. These models were selected because they have been shown to be particularly effective in learning complex relationships present in datasets related to medical applications.

10.2 Convolutional Neural Networks

A convolutional neural network (CNN) is one of the popular Deep Artificial Neural Network made up of learnable weights and biases. CNNs have been specifically designed for visual tasks such as object recognition and image classification. A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers and then followed by one or more fully connected layers as in a standard multilayer neural network. It receives an input and the features from that input are extracted through the convolution operation (Saha 2018). The convolution operation is mathematically defined as:

$$p(x) = w * y = \sum_{a=-s}^{s} w(a)y(x-a)$$

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_10

where y represents the input vector, w represents the weights/filter/kernel, and $s = (t - 1)/2$ where $1 \times t$ is the odd size of the kernel.

Convolutional Neural Networks (CNNs) possess similar attributes and adhere to the same fundamental principles, irrespective of whether they are 1D, 2D, or 3D. The main distinction arises in the dimensionality of the input data and the manner in which the feature detector (i.e., filter) moves across the data. The general CNN architecture is shown in Fig. 10.1.

The one-dimensional Convolutional Neural Network (Conv1D) architecture is typically utilized when processing one-dimensional data (see Fig. 10.2). In this architecture, the kernel convolves along a single dimension, making it ideal for processing text or sequence data. The one-dimensional nature of this architecture allows for the effective processing of sequences with temporal dependencies and has been widely used in natural language processing and speech recognition tasks (Nils 2018; Kiranyaz et al. 2021).

One-dimensional CNN has been popularly used for time series data, activity recognition, and sequence data. The one-dimensional convolution operation is mathematically defined as:

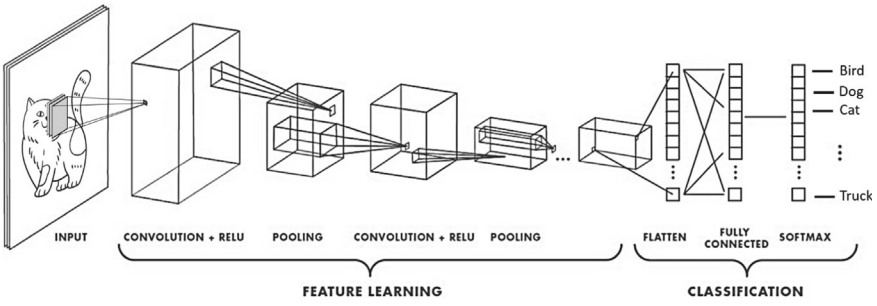


Fig. 10.1 Architecture of convolutional neural networks

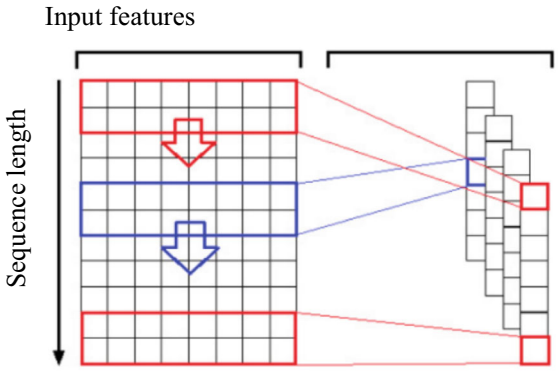


Fig. 10.2 Process of 1D convolution operation

$$Z_i = f(W * a_{i:i+k-1}) + b$$

where convolution operation is denoted by '$*$', W are the weights and a_i denote element at ith position, 'k' represents the 1D kernel size, and 'b' is the bias term.

10.3 Recurrent Neural Networks

Multilayer perceptron and convolutional neural networks have been popularly used for vector and image data. However, for Natural language processing, sequence, and time series data, sequence information is extremely important. But conventional techniques such as Bag of Words, TFIDF, ARGQZV, etc., completely discard the sequence information. Therefore, the core idea was to develop a new type of neural network which gives importance to the sequence information and leverages the information to perform better than non-sequence based approaches.

Recurrent Neural networks (Medsker 2001), is a popular technique with a lot of repeating structures which uses the sequence information for training the models. Over the years, RNNs received an immense attention and have been widely applied for time series prediction, Machine translation problems, speech recognition, and other sequence based problems. RNNs, instead of breaking the input into intervals or windows, worked on an entire input at once. In addition, RNNs were able to efficiently handle the input data of varying lengths. A simple representation of RNN is shown in Fig. 10.3.

To connect input vector $\left(x_i^1, x_i^2, x_i^3, x_i^4, \ldots\ldots, x_i^t\right)$ to the activation unit (see Fig. 10.3), the same weight is used in this repeated structure. Similarly, to use the sequence information, we have another weight matrix W' which connects outputs from a previous layer as an input to the next layer. After processing the whole sequence, the softmax or logistic layer is used to get the final output. The weight matrix W'' used at softmax layer is $\mathrm{R_d} * \mathrm{i}$ dimensional vector where $\mathrm{R_d}$ is the dimensionality of the input vector and i represents number of output classes. In case of

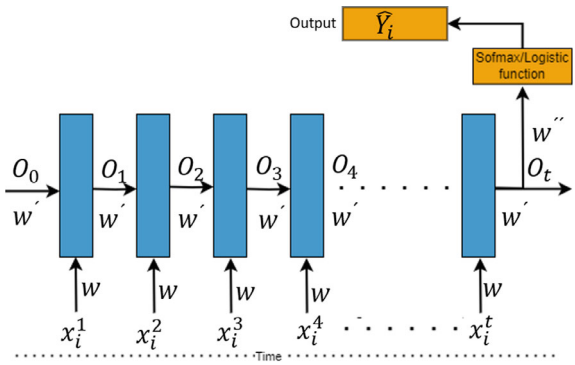


Fig. 10.3 Schematic representation of recurrent neural network

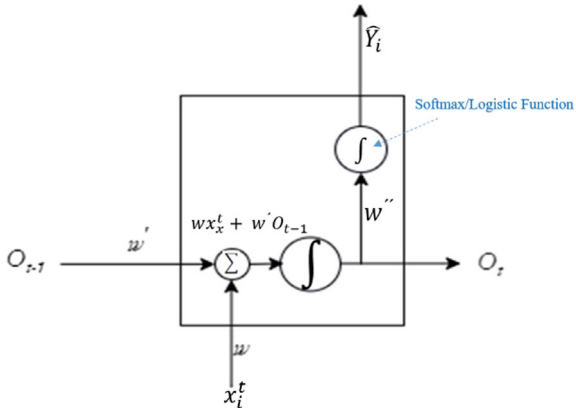


Fig. 10.4 Box representation of recurrent neural network

binary classification, the value of i is 1. The output o_0 is a vector of zeros or Gaussian random numbers and the weight matrices W, W', W'' in Fig. 10.3 are all initialized based on the activation functions used in the network. The box representation of what a simple RNN looks like is shown in Fig. 10.4.

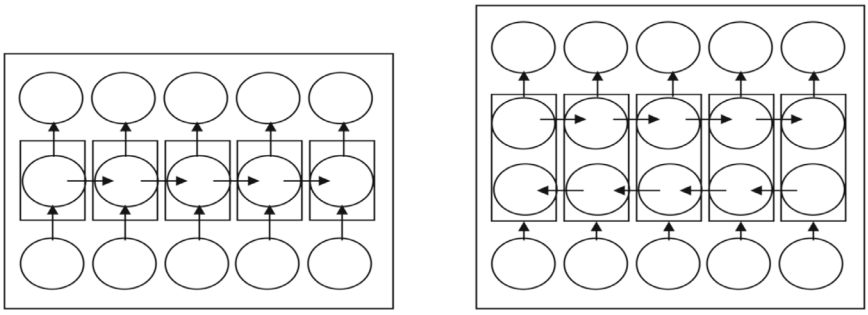
As shown in Fig. 10.4, the first step performed in RNN is to take a summation of the previous output O_{t-1} and present the input x_i^t. Both inputs have their specific weight matrices attached. The summation of these inputs is passed to the activation unit and is represented as the equation below.

$$o_t = f\left(w \cdot x_i^t + w' \cdot O_{t-1}\right)$$

where f is the activation unit, o_t is the output, w, and w' are the weight matrices, and O_{t-1} represents the output of the previous layer.

10.4 Bidirectional Recurrent Neural Networks

Simple RNNs have a major limitation known as the vanishing gradient problem, which occurs when gradients become very small during back-propagation, making it difficult to learn long-term dependencies (Bengio et al. 2014). In addition, simple RNNs can only use information from the past to predict the future, which limits their ability to capture complex patterns in sequential data. In problems that require knowledge of the elements preceding and following the current element, simple or unidirectional Recurrent Neural Networks (RNNs) have limitations as they can only move forward in the sequence of input elements (Schuster 1997). Bidirectional RNNs (BRNNs) have been developed to address this issue, enabling the capture of information in both the forward and reverse directions. The structure of a unidirectional RNN



(a) Unidirectional RNN (b) Bidirectional RNN

Fig. 10.5 Architecture of bidirectional RNN and unidirectional RNN

and BRNN is illustrated in Fig. 10.5a and b, with BRNNs allowing for the integration of context-dependent information in both forward and backward directions. By utilizing two time directions, BRNNs are able to extract relevant information from input sequences and have been utilized in a variety of applications, such as speech recognition and natural language processing.

Although recurrent neural networks can handle sequence information efficiently, they suffer from vanishing and exploding gradient problem. The main reason for the problem is not because there are huge numbers of layers but because the derivatives with respect to weights have a lot of multiplication of other partial derivatives as the forward propagation and backward propagation is performed over time resulting in vanishing and exploding gradients (Hochreiter et al. 1997). Therefore, to avoid the vanishing and exploding gradients of RNNs, several modified and enhanced RNN methods such as Long short-term memory networks (LSTM) and Gated recurrent units (GRU) have been developed.

10.5 Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) architecture that are specifically designed to overcome the vanishing gradient problem, which can occur in standard RNNs. Gradient magnitudes are influenced by two main factors: weights and activation functions, particularly their derivatives. When either factor is less than 1, gradients can diminish over time (vanishing gradients), whereas values greater than 1 can cause gradients to grow exponentially (exploding gradients). For example, the tanh activation function has a derivative less than 1 for all inputs except zero, and the sigmoid function's derivative is always less than or equal to 0.25, exacerbating the vanishing gradient problem. LSTM networks mitigate these issues with their unique architecture. In the recurrence of LSTM cells, the activation function is the identity function, which has a derivative of 1.0. This

ensures that the backpropagated gradient remains constant, avoiding both vanishing and exploding gradients. The effective weight of the recurrent connection in LSTM is controlled by the forget gate, whose activation typically ranges between 0 and 1. If the forget gate activation is close to 1.0, the gradient does not vanish. Since the forget gate activation is never greater than 1.0, the gradient cannot explode either. Thus, the LSTM architecture effectively stabilizes gradient flow during training.

LSTMs are particularly useful for processing and making predictions on sequential data, such as time series data, speech, and text. The main advantage of LSTMs over standard RNNs is their ability to selectively remember or forget information from previous time steps, which makes them well-suited for modeling long-term dependencies in sequential data. They do this by incorporating memory cells, which allow the network to selectively store and retrieve information over multiple time steps. The architecture of LSTM cell is shown in Fig. 10.6 (*source* Zarzycki et al. 2021). LSTM architecture is able to handle short-term as well as long-term dependencies efficiently because of short circuit connections (Christopher 2015; Gers et al. 2002).

In the LSTM unit, the calculations are performed sequentially using the following equations. The Forget Gate in LSTM unit decides what fraction of the previous cell state $\left(s_{\{t-1\}}\right)$ should be forgotten and is computed using the following equation.

$$f_t = \sigma\left(W_f a_t + r_f s_{\{t-1\}} + q_f\right)$$

The Input Gate (i_t) determines how much new information to add to the cell state from the current input (a_t) and is computed using the following equation.

$$i_t = \sigma\left(W_i a_t + r_i s_{\{t-1\}} + q_i\right)$$

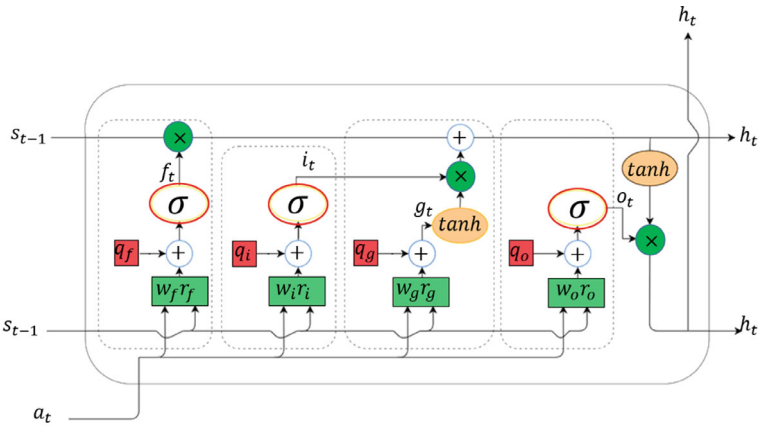


Fig. 10.6 Architecture of LSTM with 3 gates; f_t represents forget gate, i_t is the input gate, and o_t is the output gate. G_t is the updated cell state. w_f, w_i, w_g, w_o are the weight matrices associated with the input data., r_f, r_i, r_g, r_o are the recursive weights. q_f, q_i, q_g, and q_o are the bias values

The Cell gate generates new candidate values (g_t) to potentially add to the cell state and is computed using the following equation.

$$g_t = \tan h\big(W_g a_t + r_g s_{\{t-1\}} + q_g\big)$$

The new cell state (s_t) is updated by combining the retained cell state and new candidate values as shown in equation. The symbol $\circ$ indicates element-wise product.

$$s_t = f_t \circ s_{\{t-1\}} + i_t \circ g_t$$

The output gate decides the part of the cell state to output based on the current input and previous hidden state.

$$o_t = \sigma\big(W_o a_t + r_o s_{\{t-1\}} + q_o\big)$$

To compute the current hidden state (h_t) by applying the output gate to the cell state, following equation is used.

$$h_t = o_t \circ \tan h(s_t)$$

LSTMs are designed to capture long-term dependencies in sequential data by using a memory cell that can selectively forget or retain information over time. The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate. The input gate controls which information should be added to the memory cell, while the forget gate controls which information should be removed from the memory cell. The output gate controls which information from the memory cell should be used as the output.

10.6 Bidirectional Gated Recurrent Units

Gated Recurrent Units (GRUs) are a type of Recurrent Neural Network (RNN) architecture that was developed as an alternative to LSTMs (Cho et al. 2014). GRUs have several advantages over LSTMs. First, they have a simpler architecture with fewer parameters, which makes them faster to train and less prone to overfitting. Second, they have been found to perform well in a variety of sequence modeling tasks, such as speech recognition, natural language processing, and image captioning, among others. The architecture of GRU is shown in Fig. 10.7 (*source* Zarzycki et al. 2021).

In the GRU unit, the calculations are performed sequentially using the following equations. The reset gate r_t determines how much of the previous hidden state s_{t-1} to forget and is computed using the following equation.

$$r_t = \sigma\big(W_r a_t + r_r s_{\{t-1\}} + q_r\big)$$

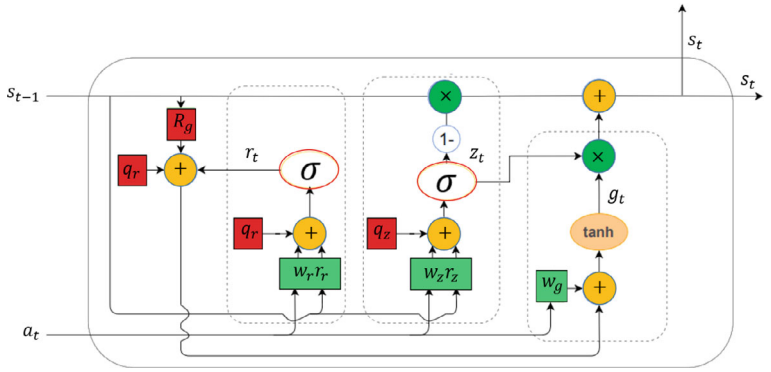


Fig. 10.7 Architecture of GRU; r_t is the reset gate, g_t is the candidate-state gate, and z_t is the update gate. Bias for reset and update gate is represented by q_r and q_z. s_t is the current hidden state and a_t is the current input. W_r, w_z, w_g are the weight matrices associated update gate, state candidate, and candidate hidden state. r_r, r_z are the recursive weights

The update gate z_t controls the degree to which the previous hidden state s_{t-1} is retained in the current state. It is computed using the current input a_t, the previous hidden state s_{t-1}, and a bias term q_z.

$$z_t = \sigma\left(W_z a_t + r_z s_{\{t-1\}} + q_z\right)$$

The Candidate Hidden State: The candidate hidden state g_t provides new potential values for the current hidden state. It is influenced by the current input a_t, the previous hidden state s_{t-1} adjusted by the reset gate, and a weight matrix w_g.

$$g_t = \tan h\left(W_g a_t + r_t \cdot \left(r_t s_{\{t-1\}}\right)\right)$$

The hidden state s_t is updated by blending the previous hidden state s_{t-1} and the candidate hidden state g_t and is computed using the following equation.

$$s_t = (1 - z_t) \cdot s_{\{t-1\}} + z_t \cdot g_t$$

Unlike LSTM, Gated Recurrent Units (GRUs) are better and simplified model developed in 2014, mainly used to train deep recurrent neural networks (Cho et al. 2014). It has 2 gates; reset (r_k) and update (z_k). The update gate controls how much information from the previous time step should be passed to the current time step, while the reset gate controls how much of the previous hidden state should be forgotten. GRU is as powerful as LSTM but most often it is faster to train as it has fewer equations than LSTM.

The remarkable success of deep neural networks (DNNs) in artificial intelligence has fueled a growing desire to deploy these networks in resource-constrained devices, such as mobile phones and edge devices. These devices require more compact and

efficient models for practical use as these devices are inherently constrained by limited computational power, storage capacity, and energy resources, all of which pose substantial obstacles to the seamless deployment of DNNs, especially in the context of Internet of Things (IoT) and various on-device applications at the edge. Against this backdrop, light-weight deep learning emerges as an attractive solution. These new, streamlined architectures are characterized by their enhanced computational efficiency. Unlike their larger counterparts that comprise millions or billions of parameters, light-weight architectures are less demanding in terms of processing power. This makes them an ideal choice for devices with limited computational capabilities, broadening the scope of where and how advanced AI models can be utilized. Furthermore, light-weight architectures address the substantial memory requirements of traditional deep learning models. Their compact nature allows them to be stored and operated on devices such as mobile phones and Internet of Things (IoT) sensors, which typically have limited storage capacity. This attribute enhances their practical applicability in today's increasingly mobile and connected world. Their versatility, efficiency, and compactness make them particularly well-suited for tasks that require processing substantial amounts of data in real time, while also being constrained by device capabilities. Among such tasks, image classification and object detection stand out as prominent examples. These tasks typically involve processing high-resolution images and making rapid decisions, making them prime candidates for the application of light-weight architectures.

10.7 Light Weighted Depth-Wise Separable Convolutional Neural Networks (DSCNN)

Convolutional Neural Networks (CNNs) have revolutionized the field of image analytics, with architectures such as ResNet and GoogleNet achieving outstanding performance in image classification tasks. However, the significant computational resources required for training and deployment of these models present a major challenge. The Depth-wise Separable Convolutional Neural Network (DSCNN) is a novel architecture (see Fig. 10.8) that addresses this problem. It is essentially a more efficient and lightweight version of the standard CNN, designed to significantly reduce the number of parameters and computational cost, thereby making it possible to train the network more quickly.

A DSCNN splits the standard convolution operation into various separate layers, each of which carries out a different task.

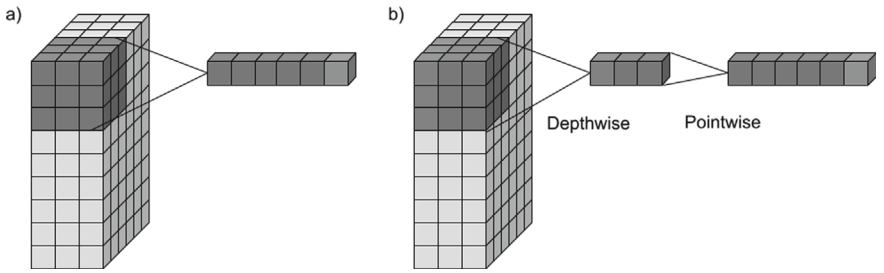


Fig. 10.8 Architecture of depth-wise separable convolutional neural networks consisting depth-wise convolution followed by pointwise convolution

10.7.1 Depth-Wise Convolution

First, a convolution is applied to each input channel separately. This is different from standard convolutions, where each filter is applied across all input channels. By applying a convolution to each input channel separately, depth-wise convolution effectively filters the inputs.

As shown in Fig. 10.9, in depth-wise convolution, each channel of the input data is convolved with its own filter, as opposed to the standard convolution where each input channel is convolved with multiple filters. For a single channel, when a $D_k \times D_k \times 1$ filter is applied, it requires $D_k \times D_k$ multiplications for each position the filter is applied to. The filter slides across the entire spatial dimensions of the input. If the output dimension is $D_m \times D_m$, there are $(D_m \times D_m)$ positions. The total multiplications for single and multi-channel are:

$$\textit{Single channel multiplications} = (D_k)^2 \times (D_m)^2$$

$$\textit{All channel multiplications} = (D_k)^2 \times (D_m)^2 xC$$

whereas in standard convolution (also known as full convolution), Standard convolution processes all input channels together with each filter. For each position in the output, a $D_k \times D_k$ filter is applied across C channels, resulting in $(D_k)^2 \times (D_m)^2 xC$ multiplications per position and with N filters applied to the entire input, the total multiplications are:

$$\textit{Total Multiplications in standard convolution} = N \times (D_k)^2 \times (D_m)^2 xC$$

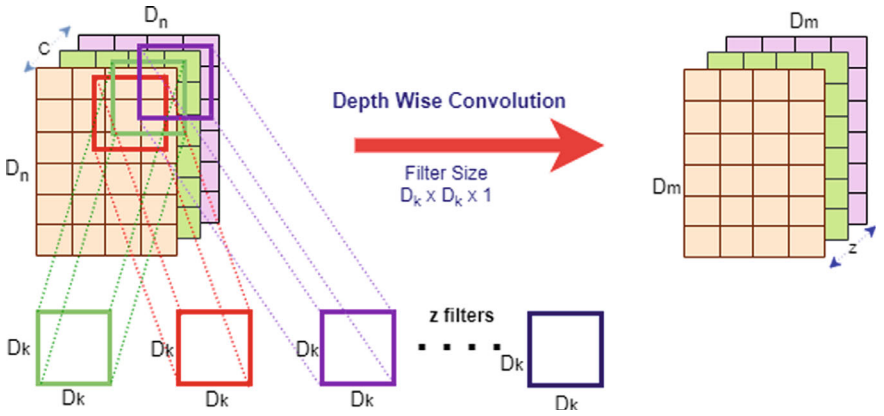


Fig. 10.9 Architecture of Depth-wise Convolution (D_n): Dimension of the input. D_k is Dimension of the filter (both width and height). D_m Dimension of the output. (C) represents the number of channels in the input. (z) is the number of filters (typically ($z = C$) in depth-wise convolution)

10.7.2 Pointwise Convolution

The second step is the pointwise convolution, which involves a 1×1 convolution operation. This step is used to create a linear combination of the output of the depth-wise convolution. The pointwise convolution increases the number of channels without looking at the spatial information and is shown in Fig. 10.10.

These two steps together make up a depth-wise separable convolution operation. The benefit of this operation over standard convolution is that it significantly reduces the computational cost, making it suitable for resource-constrained devices. For a single input channel and a single 1×1 filter, the convolutional operation necessitates a singular multiplication for each spatial position. Given the multi-channel nature of

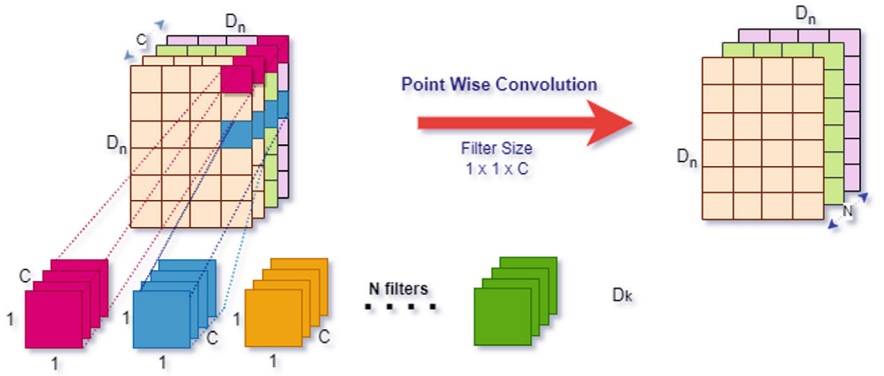


Fig. 10.10 Architecture of pointwise convolution

typical CNN inputs, the 1×1 convolution operation is executed across all C channels, resulting in C multiplications for every spatial location. The process is further generalized by applying N such 1×1 filters, thereby enabling the transformation of the channel dimensions from C to N. Each filter produces an output channel, making the depth of the output N. This entails a computational cost of C multiplications per spatial position. When extended over the entire spatial grid and across all N filters, the cumulative computational overhead is succinctly captured by the following formula:

$$C \times D_p^2 \times N.$$

where N represents the number of filters, D_p denotes the dimension of output, and C is the number of channels.

The depth-wise separable convolution is named as such because the depth-wise convolution applies to the depth of the input (the channels), and the pointwise convolution separately applies to the output of the depth-wise convolution. This separation of operations reduces the computational complexity while still maintaining a high level of model performance.

10.7.3 Group Convolution

Group convolution and channel shuffle are techniques that have been introduced in the deep learning domain, especially in the design of convolutional neural networks (CNNs), to optimize computational efficiency and model accuracy. Introduced with AlexNet and popularized by models like ResNeXt, group convolution is a method of partitioning the input and output channels of a convolutional layer into smaller groups, and then convolving each group separately. This can greatly reduce computational cost.

For a standard convolution with C_{in} input channels and C_{out} output channels, the convolution would require $C_{in} \times C_{out}$ operations. With group convolution, where the channels are split into G groups, the number of operations required by each convolution are:

$$C_{in}/_G \times C_{out}/_G$$

So, the total operations are

$$G \times C_{in}/_G \times C_{out}/_G$$

Which is $1/G$th, of the original cost if each group has the same number of channels.

Channel shuffle is a technique introduced with ShuffleNet. It's designed to allow group convolutions to learn from information across different channel groups. After group convolution, channels in one group might only be able to communicate with

channels in the same group. Channel shuffle aims to remedy this by reorganizing the channels so that subsequent layers can mix information across the original groups.

10.7.4 Comparison of Convolution Types

We compared different convolution types to evaluate their computational efficiency and effectiveness in feature extraction. The comparison (see Table 10.1) includes standard convolution, depth-wise convolution, pointwise convolution, and group convolution, using the same input, output, and filter sizes for accurate assessment of total multiplications required.

10.8 Challenges and Future Directions

Deep learning, while revolutionary, grapples with inherent limitations within individual models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which hinder their effectiveness across diverse data types. CNNs struggle to efficiently capture long-term dependencies crucial in sequence or genomic data, while RNNs face challenges such as vanishing gradients and difficulty in retaining information over long sequences. Moreover, the high computational cost associated with training and deploying deep learning models exacerbates scalability issues, necessitating more efficient algorithms and hardware solutions. Additionally, slow convergence during training and difficulties in parameter tuning impede the rapid deployment of deep learning solutions in real-world scenarios, highlighting the need for accelerated learning techniques and automated optimization strategies.

Future research endeavors can focus on overcoming these challenges by charting innovative pathways to enhance the efficacy and scalability of deep learning technologies. Combining multiple deep learning models in a hybrid or ensemble approach efficiently can harness the complementary strengths of individual models, improving performance across various tasks and data types. Automated hyperparameter optimization techniques and novel architectures tailored to specific applications can streamline the parameter tuning process and enhance model interpretability. Moreover, advancements in hardware accelerators and distributed computing frameworks can alleviate the computational burden associated with deep learning, enabling more widespread adoption and deployment in resource-constrained environments. By addressing these challenges and leveraging emerging technologies, the field of deep learning can continue to drive transformative advancements in artificial intelligence, paving the way for more robust and scalable solutions to complex real-world problems.

Table 10.1 Comparison of various Convolution types against filter size, total number of multiplications for the input parameters ($D_k = 3$, $D_m = 10$, $C = 3$, $N = 16$, G = 2, $C_{in} = 3$, $C_{out} = 16$)

Convolution type	Description	Filter size	Total multiplications	Example
Standard	Uses multiple filters that process all input channels together. Each filter is applied across all channels, capturing cross-channel features	$D_k \times D_k \times C$	$N \times (D_k)^2 \times (D_m)^2 xC$	$16 \times 3^2 \times 10^2 \times 3 = 43{,}200$
Depth-Wise	Each input channel is processed independently with its own filter. Reduces the number of multiplications by handling channels separately	$D_k \times D_k \times 1$	$(D_k)^2 \times (D_m)^2 xC$	$3^2 \times 10^2 \times 3 = 2700$
Pointwise	Uses 1×1 filters to combine information across channels after depth-wise convolution. Adds flexibility and reduces dimensionality	$1 \times 1 \times C$	$(D_m)^2 xC \times N$	$10^2 \times 3 \times 16 = 48{,}00$
Group	Divides input channels into groups and performs convolutions independently within each group, balancing between computational efficiency and cross-channel feature learning	${C_{in}}/{G} \times {C_{out}}/{G}$ Or ${C_{in}}/{G} \times D_k \times D_k$	$G \times {C_{in}}/{G} \times {C_{out}}/{G}$ Or $G \times \left({C_{in}}/{G} \times {C_{out}}/{G}\right) \times (D_k)^2 \times (D_m)^2$ $= (C_{in} \times C_{out} \times (D_k)^2 \times (D_m)^2)/G$	$\frac{3 \times 16 \times 3^2 \times 10^2}{2}$ $\frac{43{,}200}{2} = 21{,}600$

10.9 Summary

In this chapter, we have provided a comprehensive overview of selected deep learning architectures that are prominently cited in the literature for handling sequence, genomic, and image data. These architectures have been meticulously examined to elucidate their efficacy and suitability for various applications within the fields of bioinformatics and medical diagnostics. The discussion commenced with Convolutional Neural Networks (CNNs), exploring both 1D and 2D CNN variants. CNNs have demonstrated remarkable performance in tasks involving image data, with applications ranging from image classification to object detection. Additionally, Recurrent Neural Networks (RNNs) and their variants, including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), were examined for their effectiveness in sequence data analysis, particularly in tasks such as time series prediction and natural language processing. Furthermore, attention was given to light-weight architectures such as ShuffleNet, SqueezeNet, Inception, Xception, MobileNet, YOLO (You Only Look Once), and SSD (Single Shot MultiBox Detector). These architectures offer practical solutions for scenarios where embedding the model into resource-constrained devices is imperative, without compromising on performance. Their compact designs make them particularly suitable for deployment in edge devices and IoT (Internet of Things) applications, extending the reach of deep learning into domains with limited computational resources. This chapter serves as a valuable resource for researchers, practitioners, and developers seeking to leverage deep learning techniques in bioinformatics and medical diagnostics. These architectures provide a diverse toolkit to address a wide array of challenges, laying the groundwork for innovative solutions that push the boundaries of what is possible in data-driven healthcare and biological research.

Bibliography

1. P. Adarsh, P. Rathi, M. Kumar, YOLO v3-Tiny: object detection and recognition using one stage improved model, in 2020 *6th International Conference on Advanced Computing and Communication Systems (ICACCS)* (2020, March), pp. 687–694. IEEE
2. F. Chollet, Xception: Deep learning with depthwise separable convolutions, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1251–1258.
3. F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size (2016). arXiv:1602.07360
4. D. Sinha, M. El-Sharkawy, Thin mobilenet: an enhanced mobilenet architecture, in 2019 *IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)* (2019, October), pp. 0280–0285. IEEE
5. M.A. Sofi, M.A. Wani, Protein secondary structure prediction using data-partitioning combined with stacked convolutional neural networks and bidirectional gated recurrent units. Int. J. Inf. Technol. **14**(5), 2285–2295 (2022)
6. Y. Yu, X. Si, C. Hu, J. Zhang, A review of recurrent neural networks: LSTM cells and network architectures. Neural Comput. **31**(7), 1235–1270 (2019)

Chapter 11
Hybrid Deep Learning Architecture for Protein Secondary Structure Prediction

11.1 Introduction

Traditional deep learning models may struggle to capture all the relevant information and patterns due to the diverse and intricate nature of protein structures. By integrating multiple models or techniques, hybrid architectures aim to overcome these limitations and enhance prediction performance. In this chapter, we propose a novel hybrid deep learning architecture that combines the strengths of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to enhance PSSP accuracy. Our approach integrates Inception modules with Bidirectional Gated Recurrent Units (BGRUs) and incorporates attention mechanisms to dynamically focus on the most relevant features within the protein sequences. By leveraging this hybrid architecture, we aim to address the limitations of existing methods and improve the prediction performance across diverse protein datasets.

11.2 Data Preprocessing

Preprocessing techniques play a crucial role in PSSP by preparing the input data compatible for deep neural networks and analysis. Following are some common preprocessing steps used in PSSP.

- **Sequence cleaning**: This step involves removing or handling non-standard characters, gaps, or ambiguous residues in the protein sequence. It ensures that the sequence is in a consistent and suitable format for further analysis. During this step, missing or incorrect amino acids are identified and marked as 'X' in the input sequence and the corresponding output label as 'No sequence' is assigned for these positions.

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_11

- **Feature Fusion**: Feature fusion involves integrating additional features such as PSSMs, physical properties, and conservation scores, extracted using standalone tools and webservers. These features are merged or integrated together to create a unified representation that captures multiple aspects of the protein sequence.
- **Sequence Encoding**: In PSSP, the protein sequence and associated features are encoded into numerical representations to facilitate computational analysis. Various encoding techniques can be employed, including:
  - One-hot encoding: In protein sequences, every amino acid is depicted as a binary vector. This vector is characterized by having a value of 1 only at the position corresponding to the particular amino acid, with all other positions assigned a value of 0.
  - Numerical encoding: A numerical value is assigned to each amino acid based on a predefined mapping or custom encoding scheme. This encoding can be based on properties such as hydrophobicity, charge, or molecular weight.
  - Embedding: A learned or pre-trained embedding is used to represent amino acids as dense, lower-dimensional vectors. These embedding capture relationships and similarities between amino acids based on their context in the protein sequence.

  These encoding techniques allow the protein sequence and associated features to be transformed into numerical representations that can be effectively processed by machine learning or deep learning models. This enables the models to learn patterns and relationships within the sequence data and make predictions about the protein's SS.
- **Sequence Padding**: Sequence padding is a crucial step in protein sequence analysis due to the variable lengths of protein sequences. It involves adjusting the length of sequences to ensure uniformity for efficient batch processing. The commonly used threshold, such as 700 amino acids, is employed where sequences longer than 700 amino acids are trimmed, and shorter sequences are padded with zeros. This standardization enables streamlined data handling during model training, optimizing computational performance and facilitating parallelization.
- **Data normalization**: Scaling or normalizing the input data is important to ensure that different features have a similar range and influence during model training. Common normalization techniques include z-score normalization or min–max scaling.
- **Dataset splitting**: The dataset is divided into training, validation, and testing sets. The training set is employed for model learning, the validation set is instrumental in adjusting hyper-parameters and gauging model performance throughout the training process, while the test set is used to evaluate the model performance on unknown data.

11.3 Inception-BGRU Model Architecture for PSSP

Combining RNN and CNN in PSSP allows for the integration of local and non-local information. CNNs capture local patterns and spatial relationships, while RNNs model sequential dependencies and long-range interactions. By combining these architectures, the CNN component extracts local features from the protein sequence, which are then fed into the RNN component to capture non-local dependencies. This integration enables the model to effectively capture both local and non-local information, leading to improved accuracy in predicting the protein SS. This hybrid approach has been widely adopted and contributes to advancing protein structure prediction. Figure 11.1 illustrates a network comprising two Inception blocks, followed by convolutional, recurrent, and dense layers. Convolving over the data with multiple filter sizes ensures effective extraction of local and non-local interactions among residues across a diverse range. Convolution layers, like 'Conv (3)' operation involve four consecutive steps: first, a one-dimensional convolution is applied using a kernel size of three; next, Batch Normalization is used to speed up training and improve regularization; then, RELU activation function is applied; finally, Dropout is employed to prevent overfitting by randomly deactivating neurons during training. The proposed network is developed, trained, and tested using TensorFlow and Keras, with various parameters explored. A dropout rate of 0.4 was set, and a learning rate scheduler controlled the learning rate, decreasing it gradually every 40 epochs. Early stopping criteria were employed to halt training when validation metrics ceased to improve, with Tensor Board used for dynamic visualization of training and validation metrics.

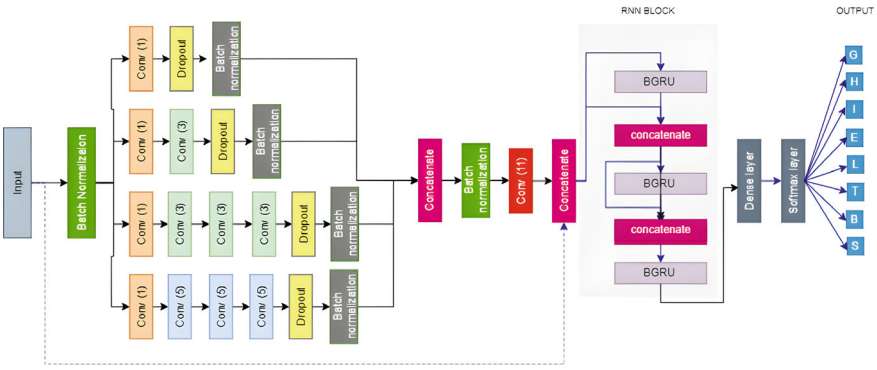


Fig. 11.1 Architecture of the inception-BGRU method

11.4 Attention Enhanced Inception-BGRU Architecture for PSSP

Attention mechanisms have become a cornerstone in modern neural network architectures, significantly enhancing model performance by allowing the network to dynamically focus on the most relevant parts of the input data. In the context of protein secondary structure prediction, attention mechanisms can improve the model's ability to capture intricate dependencies and interactions within the sequence, leading to more accurate predictions. Therefore, for improving PSSP, by integrating the attention mechanism into the Inception-BGRU architecture, the model can dynamically highlight important features. The proposed attention enhanced Inception-BGRU architecture for PSSP is shown in Fig. 11.2. It starts with an input layer followed by batch normalization. The model incorporates an Inception block, which includes multiple convolutional layers with different filter sizes (1, 3, 5), each followed by dropout and batch normalization. The outputs from these layers are concatenated and passed through an attention block, emphasizing critical features. This is followed by a 1D convolution layer. The next stage is the Bidirectional Gated Recurrent Unit (BGRU) block, which captures long-range dependencies in the sequence. Each BGRU is followed by an attention block, and the outputs are concatenated to merge information from different stages. The concatenated output is passed through a dense layer and finally through a SoftMax layer, which predicts the probabilities for various secondary structure elements (G, H, I, E, L, T, B, S). The key enhancement in this architecture is the integration of attention mechanisms, which dynamically focus on relevant parts of the sequence, thereby improving the capture of complex dependencies and interactions, and ultimately enhancing prediction accuracy.

The attention mechanism involves focusing on specific parts of the input data or features while generating an output sequence. This process as shown in Fig. 11.3 calculates a probability distribution over the elements in the input sequence, then takes a weighted sum of these elements based on this distribution to produce

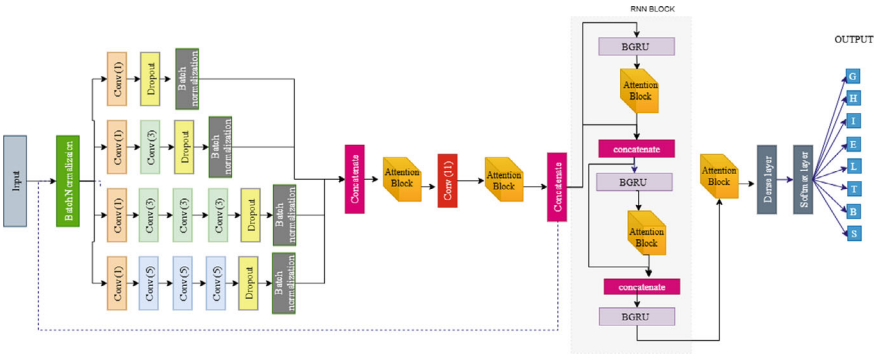


Fig. 11.2 The architecture of the inception-BGRU method is integrated with the attention mechanism. The attention block is shown in Fig. 11.3

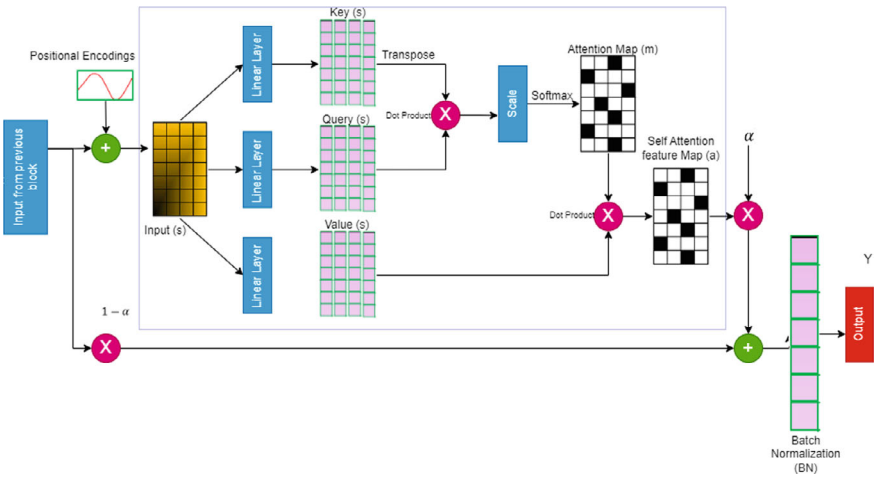


Fig. 11.3 Attention module for PSSP. s denotes input features and $s \in \mathbb{R}^{d_{proteins} \times d_{features}}$

the outputs. In the self-attention mechanism, each vector in the input sequence is converted into three different vectors: query, key, and value. The output vectors are created as a weighted sum of the value vectors, where the weights are determined based on how well the query vectors match with the key vectors, using a special function known as the compatibility function.

The input features 's' are transformed into three distinct feature spaces: Query (Q), key (K), and Value (V). These spaces are essential for computing the scaled dot-product (SDP) attention. To Compute the SDP '$m_{i,j}$' for two vectors $s_i$ands_j, the score is computed using the following equation

$$m_{i,j} = \frac{Q(s_i) \cdot K\left(s_j\right)^T}{\sqrt{d_K}}$$

where, (d_k) is defined as the dimensionality of the feature space $K \cdot \sqrt{d_k}$ is the scaling factor, which ensures that the result of the dot-product does not get prohibitively large for very long sequences. The numerator $Q(s_i)K\left(s_j\right)^T$ represents the dot product of the query vector $Q(s_i)$ and the key vector $K\left(s_j\right)$, which measures the similarity between the two vectors. To compute the attention weights, $a_{i,j}$, as shown in equation below, the weights are obtained by taking the exponential of the similarity score $m_{i,j}$ and normalizing it by the sum of exponentials of all similarity scores for the input sequence. This normalization ensures that the attention weights sum up to 1.

$$a_{j,i} = \frac{\exp\left(m_{i,j}\right)}{\sum_{n=1}^{d_{protein}} \exp\left(m_{i,j}\right)}$$

The attention weights a are then used to weight the feature vectors $V(s)$. To mitigate internal covariate shift (changes in the input distribution of each layer), the result is normalized using batch normalization (BN) as shown in equation below. This process results in, which is the output of the scaled dot-product attention mechanism.

$$r_j = BN\left(\sum_{n=1}^{d_{protein}} a_{j,i} \cdot V(s_i)\right)$$

Finally, to synthesize the output y, the output r is scaled by a learnable parameter α, as shown in equation below, while the original input feature map x is scaled by $1 - \alpha$. The final output y is obtained by summing these two scaled components.

$$y_i = (\alpha)r_i + (1 - \alpha)x_i$$

11.5 Evaluation Metrics for PSSP

- Accuracy: Accuracy is one of the popular and simplest types of measure to access the quality of the model. It is the percentage at which our model predicted or classified our data points correctly and is calculated as:

$$Q_3Acc = \frac{N_H + N_E + N_C}{N} \times 100$$
$$Q_8Acc = \frac{N_H + N_E + N_G + N_I + N_B + N_T + N_S + N_L}{N} \times 100$$

  where H, E, G, I, E, T, S, and L represent 8-state SSs and H, E, and C represent 3-state SSs predicted correctly and N denotes total no. of residues.
- Precision: Since Accuracy does not fit well for models trained on imbalanced data, Precision-Recall metrics give a more generalized assessment of the model. Precision is quantified as the proportion of true positives relative to the total number of positive predictions made by the model, as formulated below.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

- Recall: Recall is computed as the proportion of true positives in relation to the total number of actual positive data points in the given dataset, as represented below.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- F_1-score: The F_1-score, represented in equation below, is the harmonic mean of Precision and Recall. It serves as a crucial evaluation metric due to the inadequacy of accuracy as a sole determinant for model generalizability. Precision and Recall reflect the model's generalization capability, which is encapsulated effectively by the F_1-score.

$$F_1 = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)}$$

- Macro and micro average: Macro averaging treats each category with equal importance, while micro averaging assigns equal significance to each individual sample. The computation of both macro and micro averaging follows the subsequent formulae.
- Segment overlap score: It is a metric used in PSSP to evaluate the accuracy of predicted SSs. It measures the degree of overlap between the predicted SS segments and the corresponding segments in the ground truth or reference structure. SOV is calculated using the equation.

$$\mathrm{SOV} = \frac{1}{\mathrm{N}} \times \sum_{\mathrm{i} \in \begin{cases} |\mathrm{H,E,G,I,B,T,S,L}| \\ |H, E, C| \end{cases}} \sum_{\mathrm{z(i)}} \frac{\mathrm{minov}(z_1^i, z_2^i) + \delta(z_1^i, z_2^i)}{\mathrm{maxov}(z_1^i, z_2^i)} \times \mathrm{len}(z_1^i) \times 100$$

where z_1^i and z_2^i signify a pair of overlapping segments within the predicted and actual assignments, respectively.

$\mathrm{minov}(z_1^i, z_2^i)$ denotes the length of two overlapped segments z_1^i and z_2^i. Total extent of overlapping pair is computed using $\mathrm{maxov}(z_1^i, z_2^i)$. N signifies total protein residues. $\delta(z_1^i, z_2^i)$ is calculated using equation:

$$\delta(z_1^i,\ z_2^i) = \min\Big(\mathrm{maxov}(z_1^i,\ z_2^i) - \mathrm{minov}(z_1^i,\ z_2^i), \mathrm{minov}(z_1^i,\ z_2^i), {}^{z_1^i}/_2, {}^{z_2^i}/_2\Big)$$

11.6 Results and Discussion

In this section, we present a comprehensive comparison of several prominent deep learning architectures applied to PSSP. Through the analysis, we aim to highlight the strengths, limitations, and relative performance of these models, shedding light on their suitability for accurate SS prediction. A commonly used dataset of protein sequences is CB6133 which consists of 6133 non-redundant protein sequences with similarity less than 25%. The dataset is mostly used for training deep learning models

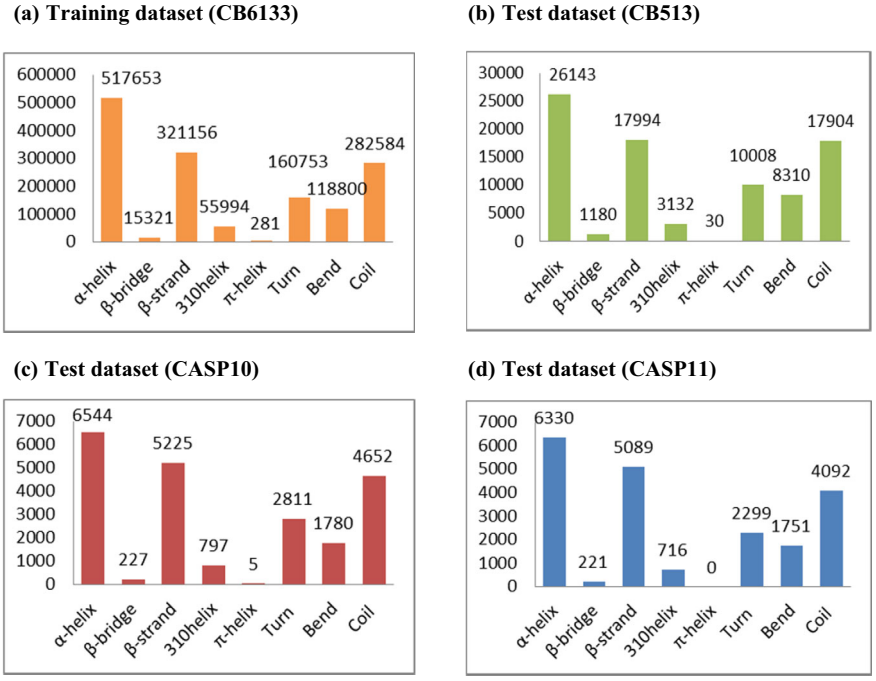


Fig. 11.4 Number of amino acids of each SS type in Training (CB6133) and testing (CB513, CASP10, and CASP11) datasets

while CB513, CASP10, and CASP11 datasets are used for evaluating the performance of deep learning methods. The amino acid frequency (structure wise) is shown in Fig. 11.4.

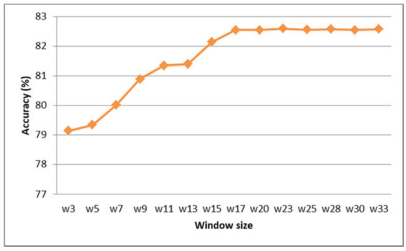
Several experiments have been conducted to evaluate the performance of various deep learning models individually as well as in hybrid. The first set of experiments are conducted to evaluate the performance of CNN with different model configurations for PSSP. The results are shown in Table 11.1. The 1D-CNN, configured with 2 layers and 42 filters for each of the filter sizes 3 and 5, achieved an accuracy of 76.9%, effectively capturing essential structural patterns. When the additional layer of convolution with same filter size and numbers is added, the accuracy improved to 77.40%. We also tested the effect of filter sizes and we observed that increasing the filter size beyond 7 adds more to the computational cost than accuracy. For further improvements, we added 2D-CNN with filter size of 3×1 and 5×1 on the output produced by the 1D-CNN with 4 layers to capture the patterns along the feature dimension and we observed that accuracy improved with a combination of both 1D and 2D CNNs as shown in Table 11.1.

In addition, we conducted experiments using various window sizes and evaluated the model's performance. Figure 11.5 illustrates the results obtained from testing the model with 14 different window sizes. It was observed that the model's performance

Table 11.1 Performance comparison of CNN with different model configurations for PSSP

Method	Filter size	Accuracy (%)
1D-CNN with 2 layers	3, 5	Q3–76.91
1D-CNN with 3 layers	3, 5	Q3–77.40
1D-CNN with 3 layers	3, 5, 7	Q3–77.42
1D-CNN with 4 layers	3, 5	Q3–82.03
Cascaded 1D-CNN and 2D-CNN with 2 layers	3, 5 and 3 × 1, 5 × 1	Q3–83.48
Cascaded 1D-CNN and 2D-CNN with 3 layers	3, 5 and 3 × 1, 5 × 1	Q3–84.55

(a) Q_3 accuracy of CNN on different window sizes



(b) Q_8 accuracy of CNN on different window size

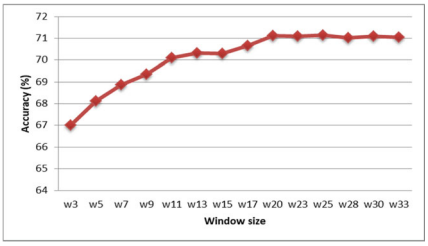
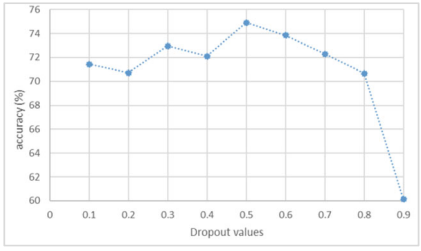


Fig. 11.5 Performance (Q_3/Q_8 accuracy) of CNN on different window sizes

remained consistent for window sizes larger than 20. Using window sizes greater than 20 increased computational requirements without significantly improving the accuracy of the predicted SSs.

We conducted several experiments to determine the optimal dropout rate for both local and non-local blocks. Figure 11.6 (left) illustrates the results, indicating that a dropout rate of 0.5 achieved the highest accuracy. Additionally, we tested different batch sizes to identify the optimal value for training our model in PSSP (PSSP). Figure 11.6 (right) displays the results, revealing that a batch size of 64 yielded the highest accuracy.



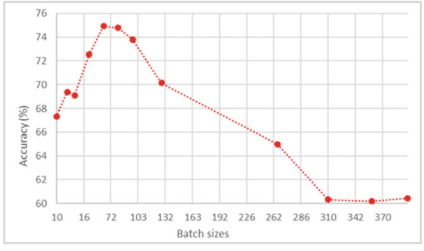


Fig. 11.6 Effect of dropout and batch size on PSSP accuracy

Table 11.2 Performance comparison (Q3 accuracy %) of Simple RNN and its variant methods for PSSP

Method	Q_3 (%)
Simple recurrent neural network	68.2
Bidirectional recurrent neural network	73.1
Long short-term memory network	77.9
Bidirectional long short-term memory network	81.3
Gated recurrent unit	78.5
Bidirectional gate recurrent unit	82.5

We also conducted various experiments on recurrent neural networks for PSSP. The performance of simple recurrent neural networks and their comparison with different RNN variants such as bidirectional RNN, LSTM, and Gated recurrent units on PSSP is shown in Table 11.2.

Hybrid deep learning methods combine multiple deep learning architectures to leverage their complementary strengths. These hybrid approaches aim to enhance the accuracy and reliability of predictions by effectively integrating different information sources and capturing complex relationships within protein sequences. Inspired by the success of hybrid methods, multiple experiments have been conducted to assess the performance of hybrid methods on PSSP accuracy. The performance analysis of various hybrid deep learning models is shown in Table 11.3. The results indicate that the inception-BGRU network achieves the best accuracy Q_3 and Q_8 accuracy of 86.4 and 74.2%. When Attention mechanism is added to the Inception-BGRU model, 1.1 and 1.2% improvement in accuracy is observed for Q_3 and Q_8.

Table 11.3 Performance comparison of various hybrid deep learning methods for prediction of protein SSs

Methods	Q_3 (%)	Q_8 (%)
2D-CNN and bidirectional LSTM	85.4	73.0
1D-CNN and bidirectional LSTM	83.7	72.9
Inception inside Inception networks with 1D-CNN	85.1	70.5
Dilated 1D-CNN	83.6	71.1
1D-CNN + 2D-CNN and bidirectional LSTM	85.7	70.5
1D-CNN-2D-CNN-BGRU and data partitioning	85.8	74.1
Inception-BGRU	86.4	74.2
Attention enhanced inception-BGRU	87.5	75.4

11.7 Challenges and Future Directions

PSSP remains a challenging task, with the current methods still lacking the desired accuracy for practical use. Over the past two decades, significant progress has been made in developing deep learning models for predicting protein SSs. However, the accuracy of these models is still below the desired level, with the best reported accuracies ranging from $70 \pm 2\%$ for eight-state predictions to $83 \pm 2\%$ for three-state predictions. Despite the challenges, there has been a slow but steady improvement in prediction accuracy, driven by the increasing availability of protein sequences and solved structures. One key challenge in PSSP is the selection and extraction of important input features. Since the raw protein sequence does not provide sufficient information for prediction, various numerical features such as amino acid composition, dipeptide composition, sequence profiles, and position-specific scoring matrices (PSSMs) are used to represent the sequence information. The choice and representation of these features greatly impact the performance of prediction models, and careful consideration is required in their selection. Capturing non-local interactions between amino acids is another critical challenge in PSSP. While the primary sequence plays a role, the SS is also influenced by spatial relationships between residues. Existing techniques, such as sliding windows and recurrent neural networks (RNNs), have limitations in effectively capturing non-local interactions. More research attention is needed to develop new models that can efficiently capture complex non-local interactions for accurate prediction. Protein length is another significant factor that poses a challenge in SS prediction. Proteins can vary widely in length, and their shape and function are impacted by their sequence length. However, current methods often require fixed-length proteins as input, resulting in accuracy issues at boundary regions. Excessive zero-padding of shorter sequences has been shown to negatively impact deep learning-based prediction methods. Further research is needed to address this challenge and improve accuracy, particularly in boundary regions. Another challenge in PSSP is the separate prediction of regular and non-regular SSs. Regular SSs, such as alpha-helices and sheets, are distinct from non-regular structures like turns and coils. Current methods often develop separate prediction models for regular and irregular structures, making their utilization for structural analysis and function prediction complex and challenging.

11.8 Summary

This work proposed a novel hybrid deep learning architecture, integrating Inception modules with Bidirectional Gated Recurrent Units (BGRUs) and attention mechanisms, aimed at enhancing protein secondary structure prediction (PSSP) accuracy. Extensive experiments were conducted across various CNN, RNN, and hybrid configurations to evaluate and compare their effectiveness. The results demonstrated significant improvements over traditional models. Specifically, the Inception-BGRU

model achieved a Q3 accuracy of 86.4% and a Q8 accuracy of 74.2%. When enhanced with attention mechanisms, the model's Q3 accuracy increased to 87.5%, and Q8 accuracy improved to 75.4%, reflecting a 1.1% and 1.2% increase, respectively. These results underscore the effectiveness of combining CNN and RNN architectures with attention mechanisms in capturing both local and global dependencies in protein sequences, leading to more accurate predictions. The proposed architecture sets a new benchmark for PSSP and offers a robust framework for further advancements in the field.

Bibliography

1. B. Alberts et al., Molecular biology of the cell. Biochem. Mol. Biol. Educ. **36**(4), 317–318 (2008). https://doi.org/10.1002/bmb.20192
2. H.M. Berman, The protein data bank. Acta Crystallogr. Sect. D Biol. Crystallogr **58**(6 I), 899–907 (2002). https://www.rcsb.org
3. Q. Jiang, X. Jin, S.J. Lee, S. Yao, PSSP: a survey of the state of the art. J. Mol. Graph. Model. **76**, 379–402 (2017). https://doi.org/10.1016/j.jmgm.2017.07.015
4. W. Kabsch, C. Sander, Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. Biopolymers **22**(12), 2577–2637 (1983). https://doi.org/10.1002/bip.360221211
5. M.A.R. Ratul, M.T. Elahi, M.H. Mozaffari, W.S. Lee, PS8-Net: a deep convolutional neural network to predict the eight-state protein secondary structure. 2020 Digital image computing: techniques and applications. DICTA **2020**, 8–10 (2020)
6. M.A. Sofi, M.A. Wani, Protein secondary structure prediction using data-partitioning combined with stacked convolutional neural networks and bidirectional gated recurrent units. Int. J. Inf. Technol. 1–11 (2022). https://doi.org/10.1007/s41870-022-00978-x
7. M.A. Sofi, M.A. Wani, Improving prediction of protein secondary structures using attention-enhanced deep neural networks, in *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)* (2022, March), pp. 664–668. IEEE. https://doi.org/10.23919/INDIACom54597.2022.9763114
8. M.A. Sofi, M.A. Wani, RiRPSSP: a unified deep learning method for prediction of regular and irregular protein secondary structures. J. Bioinform. Comput. Biol. **21**(01), 2350001 (2023)
9. M.A. Wani, F.A. Bhat, S. Afzal, A.I. Khan, *Advances in Deep Learning* (Springer, 2020)
10. Y. Yang, J. Gao, J. Wang, R. Heffernan, J. Hanson, K. Paliwal, Y. Zhou, Sixty-five years of the long march in PSSP: the final stretch? Brief. Bioinform. **19**(3), 482–494 (2018). https://doi.org/10.1093/bib/bbw129
11. B. Zhang, J. Li, Q. Lü, Prediction of 8-state protein secondary structures by a novel deep learning architecture. BMC Bioinform. **19**(1), 1–13 (2018). https://doi.org/10.1186/s12859-018-2280-5
12. J. Zhou, O.G. Troyanskaya, Deep supervised and convolutional generative stochastic network for protein secondary structure prediction. 31st Int. Conf. Mach. Learn. (ICML 2014) **2**, 1121–112 (2014)

Chapter 12
Enhanced Accuracy in Pan-Cancer Classification Using Ensemble Deep Learning

12.1 Introduction

Ensemble Learning is a technique that combines several individual models to create a stronger and more powerful model than any of its individual components. This method also helps reduce the risk of overfitting due to the diversity of the models being combined. Ensemble learning has been successfully applied in many fields and often outperforms single models. There are various ensemble techniques, such as averaging, bagging, random forest, stacking, and boosting. These techniques differ in how they train and combine the individual models. Many studies and reviews have explored these ensemble learning methods, both for traditional machine learning and, more recently, for deep learning. Traditional ensemble learning focuses on combining simpler models, but recent research has increasingly applied these methods to deep learning. However, many of these attempts use basic approaches like averaging, which can be biased towards weaker models. Although there are several strategies for combining deep learning models, they often face challenges related to generalization, training difficulties, and other issues.

The fundamental concept behind any ensemble learning system is to utilize an aggregation function, denoted as (G), to merge multiple base classifiers $(d_1, d_2, ..., d_h)$ into a single predictive output as shown in Fig. 12.1. Given a dataset (D) containing (l) samples, each with (m) features, the prediction for each sample l is derived using this ensemble method. The final output (y_i) for a given input (x_i) is computed as:

$$y_i = f(z_i) = G(d_1, d_2, \ldots, d_h)$$

where y_i is the predicted output for the ith sample, z_i is the input feature vector for the ith sample, $f(z_i)$ prediction function, and G is the aggregation function for combining the outputs of the base classifiers.

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_12

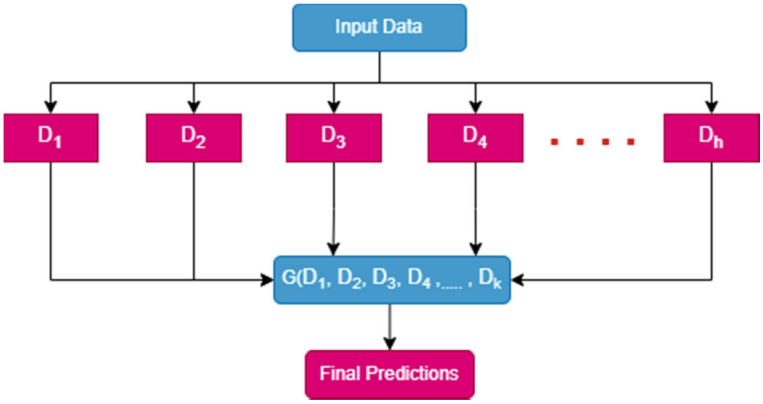


Fig. 12.1 Overview of ensemble learning structure

12.2 Ensemble Deep Learning Method

We utilized the normalized gene expression data for 33 cancer types, obtained from the Pan-cancer atlas. The dataset comprises 10,267 cancer samples corresponding to 20,531 genes. Due to the presence of non-essential information in the dataset, we combined data from 33 different cancer types, carefully curated it, and removed any genes with low variability. Additionally, because genes located close to each other are more likely to interact, they were organized based on their chromosomal positions. After curating the data, the high-dimensional gene expression data, originally in a one-dimensional format of 10,381 $\times$ 1, was converted into a two-dimensional form with dimensions of 102 $\times$ 102 (see Fig. 12.2 for sample data). This transformed data serves as the input for the ensemble deep learning architecture.

12.2.1 Ensemble Architecture

This section introduces a new ensemble method for classifying 33 different types of cancer. The proposed method integrates advanced deep learning architectures, including DenseNet, ResNet, Inception-V3, and the Xception model. The initial phase of the approach involves preprocessing the pan-cancer dataset to convert the raw gene expression data into formats suitable for deep learning models. Following this, the preprocessed data is input into the proposed method for cancer type classification. The 33 classification output computed by individual models is fused and the final output is obtained using different aggregation methods. The structure of Ensemble method for Pan-Cancer Subtype classification is illustrated in Fig. 12.3.

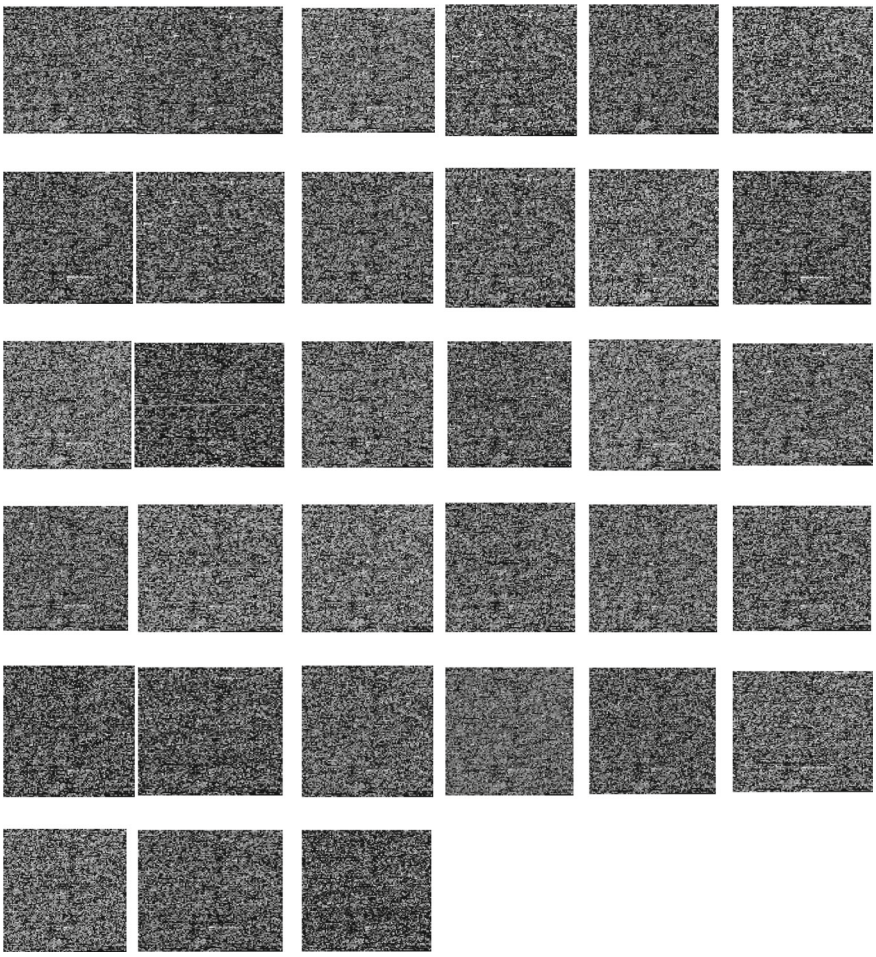


Fig. 12.2 Representation of size 102 × 102 samples of 33 cancer types transformed from gene expression data

12.2.2 Ensemble Aggregation Methods

The aggregation of multiple models is achieved using various strategies, among which voting methods are most frequently employed. Voting methods are crucial in ensemble learning, primarily used in classification and regression tasks to enhance predictive accuracy. These methods aggregate the outputs of multiple classifiers to form a final prediction. The three primary voting strategies are max voting, averaging voting, and weighted averaging voting.

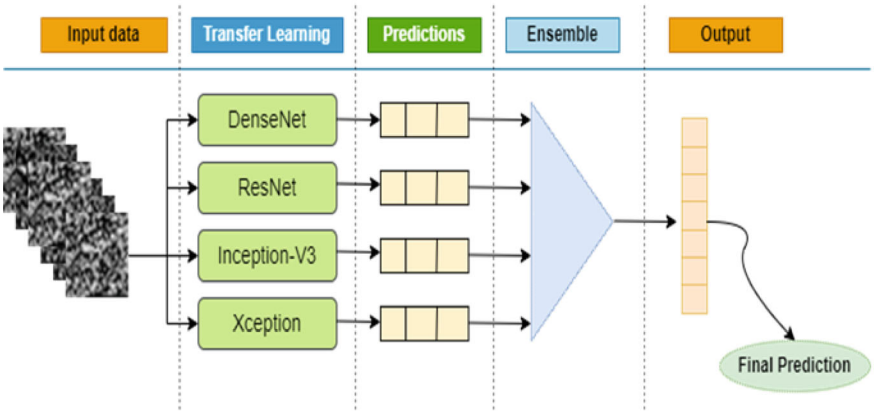


Fig. 12.3 Overall framework of ensemble method for pan-cancer classification

- **Max Voting**

Max Voting is a popular ensemble technique used to combine the predictions of multiple classifiers. It comes in two forms: Hard Voting and Soft Voting. Hard (Majority) voting involves selecting the class label that receives the most votes among all classifiers. Suppose classifiers $(d_1, d_2, \ldots, d_k)$ give class predictions for an input (z). The final output (y^*) is determined by the most frequent prediction:

$$y^* = Mod\left[d_{1(z)}, d_{2(z)}, \ldots, d_{k(z)}\right]$$

where y^* is the predicted class label, $d_{1(z)}, d_{2(z)}, \ldots, d_{k(z)}$ are the predictions from the individual classifiers.

In soft voting, the predicted probabilities from each base classifier d_j are aggregated to determine the final prediction. The final class label y^* is chosen based on the highest combined weighted probability.

$$y^* = \underset{c}{\text{argmax}} \sum_{j=1}^{k} w_j d_j(z)$$

where y^* is the predicted class label, *argmax* selects the class c that maximizes the sum of weighted predictions. w_j is the weight assigned to the jth classifier. d_j represents the predicted probability or score for class c from the jth classifier for the input feature vector z.

- **Average Voting**

Averaging Voting is a method where predictions are gathered from multiple classifiers, and their average is used to make the final prediction. This approach calculates

the arithmetic mean of the predictions, where the mean is the sum of all predictions divided by the number of predictions. This method is powerful in terms of predictive strength and is generally more accurate than majority voting. However, it is more computationally intensive because it requires averaging the results from all classifiers. The mathematical equation for averaging voting is:

$$y^* = \underset{c}{\text{argmax}} \left(\frac{1}{k} \sum_{j=1}^{k} d_j(z) \right)$$

where y^* is the predicted class label, *argmax* selects the class c having highest average score.k is the total number of classifiers in the ensemble. d_j represents the predicted probability or score for class c from the jth classifier for the input feature vector z.

- **Weighted Average Method**

It is an advanced version of the averaging voting method where different weights are assigned to each classifier in the ensemble. These weights indicate the relative importance of each model in making the final prediction. To compute the weighted average, each prediction from the classifiers is multiplied by its corresponding weight. The sum of these weighted predictions is then divided by the sum of the weights. The equation for weighted average voting can be expressed as:

$$y^* = \underset{c}{\text{argmax}} \left[\frac{\sum_{j=1}^{k} w_j \cdot d_j(z)}{\sum_{j=1}^{k} w_j} \right]$$

where y^* is the predicted class label, *argmax* selects the class c having highest average score.k is the total number of classifiers in the ensemble. w_j is the weight assigned to the jth classifier, d_j represents the predicted probability or score for class c from the jth classifier for the input feature vector z.

12.3 Experimental Setup and Results

For the classification of 33 cancer types using the proposed deep learning architecture, several evaluation metrics are essential to assess the model's performance accurately. Accuracy measures the percentage of correctly classified samples out of the total, providing a basic but sometimes insufficient metric, especially with imbalanced datasets, and is computed using the following equation.

$$\text{Accuracy} = \frac{\text{No.of correctly classified points}}{\text{Total no.of points}} \times 100$$

Precision focuses on the proportion of true positives among the instances predicted as positive, which is crucial in determining the reliability of positive predictions. Recall assesses the ratio of true positives against all actual positive instances in the dataset, highlighting the model's ability to capture all relevant cases. To compute the precision and recall, following equations are used.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

The F1-score, a harmonic mean of Precision and Recall, offers a balanced metric when dealing with models where either metric alone could be misleading. It is computed using the equation below.

$$\text{F}_1 = 2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}$$

All experimental undertakings for this study were orchestrated on an NVidia DGX A100 server, equipped with 8 GPUs, each boasting 40 GB of dedicated memory, supplemented by a RAM capacity of 1 TB. The architectures for pan-cancer classification were meticulously developed and trained using the Tensorflow and Keras deep learning frameworks. The ensemble model was initialized with the default Tensorflow weights. In ensemble model, to circumvent model overfitting, L2 regularizers were employed in tandem with an early stopping strategy, which was set with a monitoring patience threshold of 7 epochs. Training leveraged a learning rate of 0.0001, accompanied by a decay factor of 0.5, implemented to adjust the learning rate after intervals of 40 epochs. Training persisted for a comprehensive span of 300 epochs. After rigorous experimentation, the optimal batch size was determined to be 128. The terminal output layer was characterized by a softmax activation function. To address the variance predicament—often manifesting as overfitting—a dropout rate of 0.5 was instituted. This methodology selectively deactivates nodes, a strategic move grounded in the uneven explanatory power some nodes present during training.

12.3.1 Handling Data Imbalance Using Weight Balancing

In the realm of cancer genomics and molecular profiling, the concept of a pan-cancer analysis aims to discern overarching patterns and insights across multiple cancer types. While this holistic approach promises a comprehensive understanding of oncogenic processes, it also introduces a series of challenges, chief among them being dataset imbalance. Given the conspicuous imbalance in pan-cancer data, a class weighting strategy becomes indispensable. This approach, often referred to as

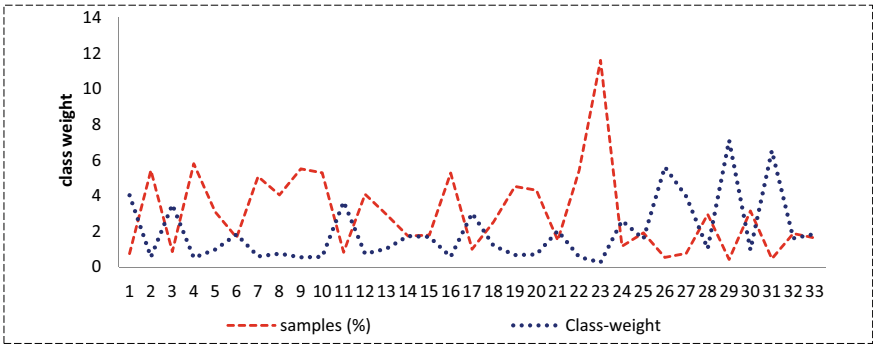


Fig. 12.4 Magnitude of class weight value on major and minor classes of pan-cancer

'Weight Balancing', involves the formulation of a dictionary, where keys represent the classes and the corresponding values signify the weights. When assimilated into the training process, if class j possesses a greater class weight relative to class j', the gradients computed from samples of class j will have a pronounced influence on the neural network's training relative to j'. Such a deliberate weighting ensures that the model pays adequate attention to underrepresented classes, culminating in a more balanced and clinically relevant model. By leveraging this strategy, we not only address the inherent bias but also refine the model's sensitivity, particularly vital given the clinical implications of pan-cancer classifications.

As illustrated in Fig. 12.4, the weight allocation strategy employed in this study assigns heightened weights to minority classes and reduced weights to majority classes. The rationale behind this weighting is grounded in the distribution of class samples. Specifically, if class j' is represented 100 times more frequently than class j in the dataset, then a single sample from class j is accorded the same significance as 100 samples from class j'. Consequently, the class weight for j should be magnified 100-fold relative to j'. This ensures that the gradient magnitudes derived from samples of class j are amplified, being 100 times greater than those from class j', thereby directing the model's attention proportionally during the learning phase and countering the inherent class imbalance.

12.3.2 Results

A comprehensive evaluation was carried out on the Pan-cancer dataset employing cutting-edge deep learning architectures, including DenseNet, ResNet50, Inception-V3, and Xception. Each model underwent individual training and testing to ascertain its performance efficacy relative to the Pan-cancer dataset. The performance of these deep architectures is shown in Table 12.1.

Table 12.1 Comparative performance metrics of individual deep learning architectures and the ensemble method on the pan-cancer dataset. Metrics include Accuracy, Precision, Recall, and F-score for each method

Method	Accuracy	Precision	Recall	F-score
DenseNet	87.32	87.32	87.67	87.44
ResNet50	90.45	91.05	90.77	91.22
Inception-V3	93.10	93.53	93.05	93.15
Xception	92.54	93.76	93.45	93.71
Ensemble method	**96.88**	**96.94**	**96.90**	**96.93**

The empirical outcomes, delineated in Table 12.1, show that DenseNet obtained an accuracy of 87.32%. In contrast, ResNet50, Inception-V3, and the Xception method achieved accuracies of 90.45%, 93.10%, and 92.54%, respectively. Analyzing these standalone results, the Xception architecture emerges as superior, recording a precision of 93.76%, a recall of 93.45%, and an F-score of 93.71%. The differential performance can be attributed to the intrinsic architectural distinctions of each model. For instance, Xception's depth-wise separable convolutions likely offer enhanced feature extraction capabilities for this specific dataset compared to the other architectures. In pursuit of augmenting classification efficacy, an ensemble strategy was deployed, amalgamating the strengths of all aforementioned architectures. The resultant ensemble model, leveraged an averaging mechanism for its final output determination. Notably, as presented in Table 12.1, the Ensemble method outperformed individual models, registering a commendable accuracy of 96.89%, complemented by a precision of 96.94%, a recall of 96.90%, and an F-score of 96.93%. The augmented performance of the ensemble, can be rationalized by its ability to harness diverse feature extraction proficiencies from each individual model. This diversity, when combined, offers a more holistic representation, mitigating individual model biases and capturing a broader spectrum of patterns intrinsic to the Pan-cancer dataset.

We evaluated the performance of our ensemble architecture on the classification of 33 cancer types using different voting methods, as shown in Fig. 12.5. Max Voting, which relies solely on majority rule, achieved the lowest accuracy, precision, recall, and F-score, as it doesn't account for the varying confidence levels of the models. Average Voting improved performance across all metrics by incorporating the probability distributions of each prediction, reflecting a more balanced and informed aggregation. However, the best results were obtained using Weighted Average Voting, which assigns different importance to each classifier based on their reliability. This approach capitalized on the strengths of the more accurate models, leading to the highest scores in accuracy (97.15%), precision (97.2%), recall (97.1), and F-score (97.15). The superior performance of Weighted Average Voting underscores the importance of tailoring the aggregation strategy to the varying contributions of individual classifiers, especially in complex tasks like pan-cancer classification.

To further evaluate the ensemble method, a confusion matrix was employed, providing a cross-tabulation of true versus predicted classes. As depicted in Fig. 12.6,

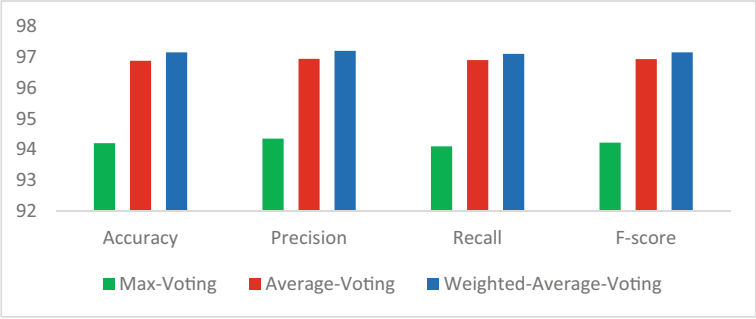


Fig. 12.5 Performance metrics (Accuracy, Precision, Recall, F-score) for different voting methods in the ensemble model for pan-cancer classification

each column of the matrix signifies the predicted class instances, while each row indicates actual class instances. This matrix elucidates both the general performance of our classifier and specific error types.

The intricacies of cancer classification are underscored by the multitude of unique malignancies, each exhibiting its own molecular and histological signature. In this realm, the efficacy of a predictive model is truly gauged by its ability to discern between these subtle variations and categorize them accurately. Table 12.2 provides an in-depth exposition into the class-wise performance of the Ensemble method, offering insights into its diagnostic precision across a diverse spectrum of cancer types. As depicted in Table 12.2, each cancer type is denoted by both its full name

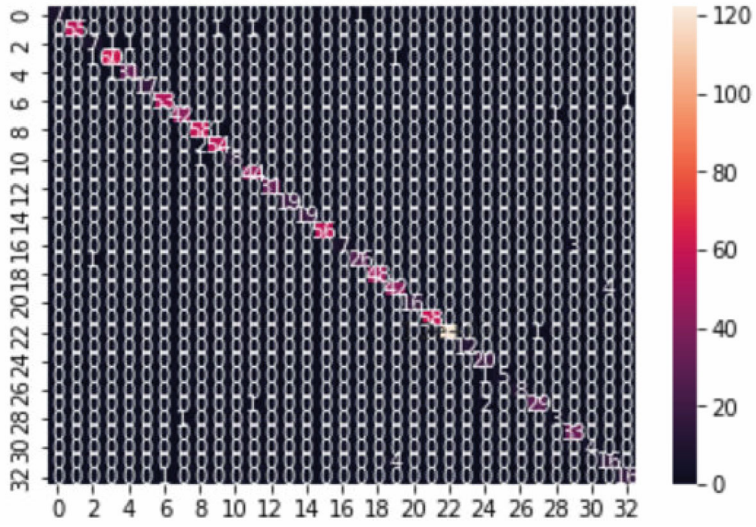


Fig. 12.6 Confusion matrix for the Ensemble method applied to the pan-cancer test dataset

and an associated code. For each cancer type, the precision, recall, and f1-score are meticulously documented. Metrics such as precision capture the proportion of true positive predictions among all positive predictions, while recall evaluates the proportion of true positive predictions among all actual positives. The F1-score offers a harmonized measure, considering both precision and recall. The table reveals nuanced insights into how effectively the ensemble model classifies each cancer type, providing a granular perspective on its diagnostic capabilities across a diverse spectrum of malignancies. Through this granular breakdown, we aim to underscore the model's strengths and areas of improvement in distinguishing between specific cancer classes.

The table delineates detailed classification metrics for a wide array of cancer types using the ensemble of deep learning method. Each cancer type is identified both by its full designation and a concise code. Across the spectrum of malignancies:

- Breast invasive carcinoma (BRCA) exhibits a precision of 1, coupled with a recall of 0.88, resulting in an F1-score of 0.93.
- Brain Lower Grade Glioma (LGG) stands out with perfect precision and an impressive recall of 0.96, culminating in an F1-score of 0.98.
- The table also highlights certain cancer types, such as Liver hepatocellular carcinoma (LIHC), Colon adenocarcinoma (COAD), and Cervical and endocervical cancers (CESC), that achieved exemplary performance metrics, registering a perfect score across all three categories.
- On the other hand, Adrenocortical carcinoma (ACC) has a slightly lower precision, recall, and F1-score, all at 0.75.

The majority of cancer types, like Lung adenocarcinoma (LUAD) and Head and Neck squamous cell carcinoma (HNSC), showcase metrics in the high nineties, underlining the method's efficacy. The comparative analysis of F-scores for various cancer types across individual and ensemble deep learning models is shown in Fig. 12.7.

12.4 Challenges and Issues

While methods tailored for various data types, including RNASeq, show versatility, they underscore challenges in broad applicability. The inherent preprocessing complexities, spanning both filter and wrapper approaches, emphasize the intricacies involved in data preparation for efficient model learning. Genomic datasets, due to their high-dimensional nature, present substantial challenges for traditional deep learning architectures, primarily tailored for 2-D imagery. Naively mapping genomic data onto these images, while innovative, may miss capturing the nuanced relationships intrinsic to genomic sequences, potentially leading to classification inaccuracies. A significant concern in pan-cancer classification is the stark imbalance in data samples across tumor types. Some tumors are well-represented with thousands of samples, whereas rarer ones might only feature sparsely. This disparity

Table 12.2 Class-wise Precision and Recall comparison between proposed method and baseline methods

Cancer Type	Code	Precision	Recall	F1-score
Breast invasive carcinoma	BRCA	**1**	0.88	0.93
Brain lower grade glioma	LGG	**1**	**0.96**	0.98
Uterine corpus endometrial carcinoma	UCEC	0.78	0.78	0.78
Lung adenocarcinoma	LUAD	**0.97**	**0.97**	0.97
Head and neck squamous cell carcinoma	HNSC	**0.97**	**0.97**	0.97
Thyroid carcinoma	THCA	**1**	**1**	1
Prostate adenocarcinoma	PRAD	**0.98**	**0.98**	0.98
Lung squamous cell carcinoma	LUSC	**0.98**	**0.98**	0.98
Bladder urothelial carcinoma	BLCA	**0.95**	**0.98**	0.97
Stomach adenocarcinoma	STAD	**0.96**	**0.96**	0.96
Skin cutaneous melanoma	SKCM	**1**	**0.89**	0.94
Kidney renal clear cell carcinoma	KIRC	**0.96**	**1**	0.98
Liver hepatocellular carcinoma	LIHC	**1**	**1**	1
Colon adenocarcinoma	COAD	**1**	**1**	1
Cervical and endocervical cancers	CESC	**1**	**1**	1
Kidney renal papillary cell carcinoma	KIRP	**1**	**1**	1
Sarcoma	SARC	**1**	0.7	0.82
Ovarian serous cystadenocarcinoma	OV	**0.96**	**0.96**	0.96
Esophageal carcinoma	ESCA	**1**	**1**	1
Pheochromocytoma and paraganglioma	PCPG	**0.89**	**0.91**	0.9
Pancreatic adenocarcinoma	PAAD	**1**	**1**	1
Testicular germ cell tumors	TGCT	**1**	**1**	1
Glioblastoma multiforme	GBM	**1**	**0.99**	1
Thymoma	THYM	**1**	**1**	1
Rectum adenocarcinoma	READ	**0.87**	**1**	0.93
Acute myeloid leukemia	LAML	**1**	0.83	0.91
Mesothelioma	MESO	**1**	**1**	1
Uveal melanoma	UVM	**0.97**	**0.91**	0.94
Adrenocortical carcinoma	ACC	0.75	0.75	0.75
Kidney chromophobe	KICH	0.92	**1**	0.96
Uterine carcinosarcoma	UCS	**1**	**1**	1
Lymphoid neoplasm diffuse large B-cell lymphoma	DLBC	0.8	0.8	0.8
Cholangiocarcinoma	CHOL	**0.94**	**0.94**	0.94

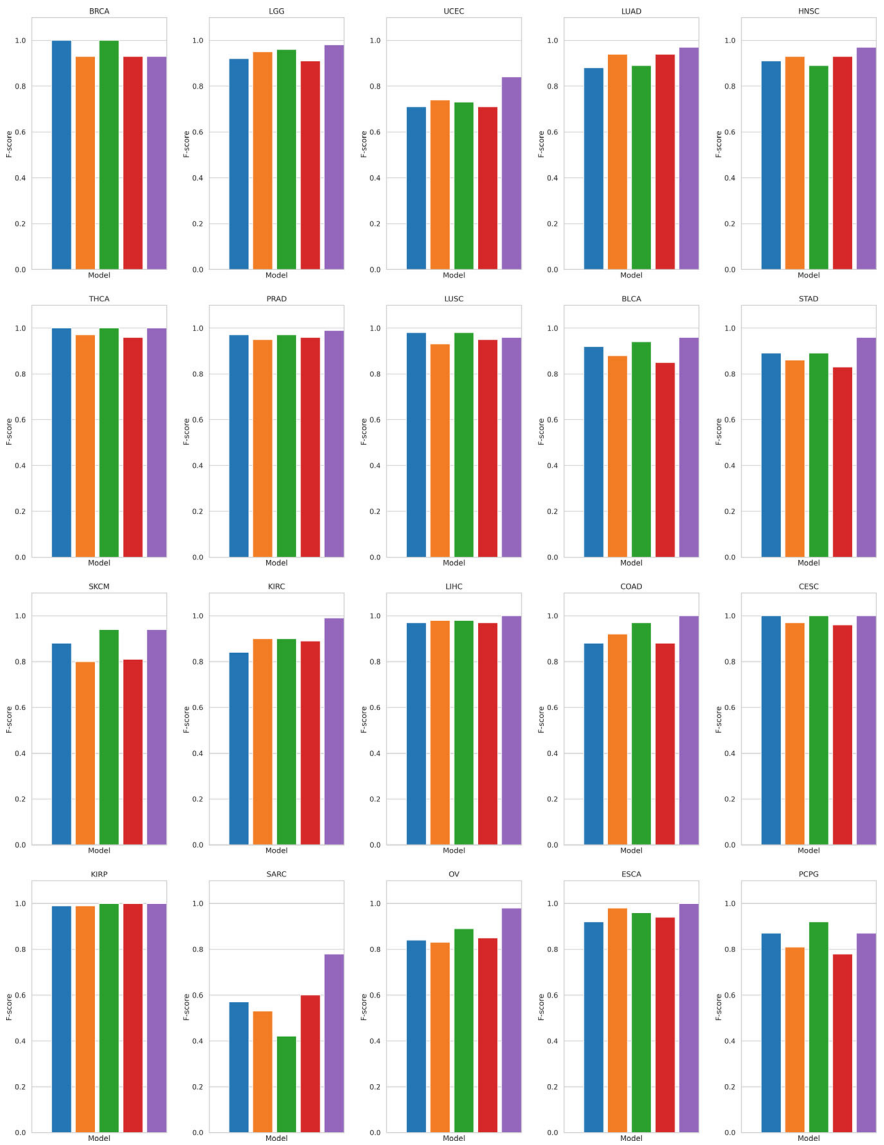


Fig. 12.7 Comparative analysis of F-scores for various cancer types across individual and ensemble deep learning models. Each subplot represents a specific cancer type, with the F-score of each model depicted as bars. The models under evaluation include Inception-V3, DenseNet-201, Xception Model, ReSNET50, and an Ensemble Model. Overall, the ensemble model consistently showcases competitive or superior performance across most cancer types

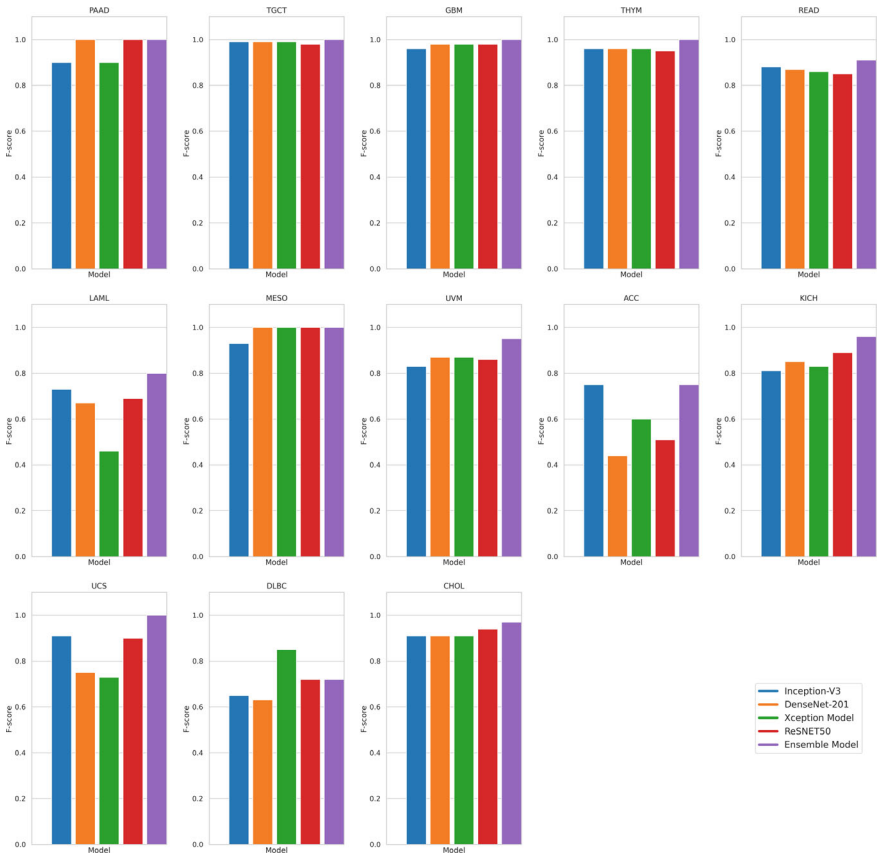


Fig. 12.7 (continued)

can bias deep learning models, potentially skewing them towards overrepresented classes. The current focus on highly expressed genes also neglects those with lower expression levels. While this strategy minimizes false positives, it may omit key biomarkers that are expressed more subtly. An added dimension to this challenge is the primary concentration on overexpressed genes, possibly sidelining markers downregulated in specific cancer types.

Deep learning models, by nature, strive to optimize specific objectives. In this context, they might bypass genes showcasing particular cancer type expressions if other genes better suit their objective, potentially missing out on pivotal biomarkers. For instance, the NAT1 gene's higher expression in breast cancer was notably overlooked. Moreover, while neural network visualization methods promise insights into influential genes, ensuring that these visual insights resonate with biological significance remains a challenge. Given the noted advancements over previous methods, there's an imperative for continual benchmarking against emergent models in tumor type classification. Bridging the gap between the swift progression in computer vision

methodologies and the unique challenges posed by genomic data becomes essential in this nascent field of genomic deep learning. Addressing these concerns will help unlock profound insights and refine diagnostic tools in pan-cancer classification.

12.5 Summary

Pan-cancer classification emerges as a pivotal avenue in oncological research, steering the focus from singular cancer types to identifying commonalities spanning multiple malignancies. Deep learning, with its profound capabilities, has cemented its relevance in this sphere, particularly in discerning cancer subtypes via gene expression data analysis. Throughout this chapter, the versatility of numerous deep learning architectures, ranging from Convolutional Neural Networks to more intricate designs like DenseNet, ResNet50, Inception-V3, Xception, and VGG16, has been meticulously explored. Leveraging ensemble approaches and the nuances of transfer learning not only escalate performance but also adeptly circumvents the perennial issue of limited labeled data.

With data preprocessing and feature extraction at its core, the chapter underscores the pivotal role of effective methodologies to derive meaningful patterns from complex datasets. Tackling high-dimensional samples, particularly in the realm of gene expression data for cancer prediction, necessitates the integration of regularizers and dropouts to thwart overfitting. One of the salient achievements in this domain has been the attainment of a remarkable 96.88% accuracy, setting a new benchmark in pan-cancer classification. Beyond sheer accuracy, the efficacy of class weighting schemes in addressing data imbalance stands out, offering a robust solution to a longstanding challenge. Furthermore, the principles and methodologies elucidated here, especially the ensemble technique, hold promise for a broader spectrum of applications, including sequence-to-sequence and image classification. As the chapter culminates, it underscores the transformative potential of deep learning in pan-cancer subtype classification, marking a significant stride towards a more nuanced understanding of the intricacies of cancer.

Bibliographys

1. F. Bray, J. Ferlay, I. Soerjomataram, R.L. Siegel, L.A. Torre, A. Jemal, Global cancer statistics 2018: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries. CA: Cancer J. Clin. **68**(6), 394–424 (2018). https://doi.org/10.3322/caac.21492
2. D. Crosby, S., Bhatia, K.M. Brindle, L.M. Coussens, C. Dive, M. Emberton, S. Balasubramanian, Early detection of cancer. Science **375**(6586), eaay9040 (2022). https://doi.org/10.1126/science.aay9040
3. A. Cruz-Roa, H. Gilmore, A. Basavanhally, M. Feldman, S. Ganesan, N.N. Shih, J. Tomaszewski, F.A. González, A. Madabhushi, Accurate and reproducible invasive breast cancer detection in

whole-slide images: a Deep Learning approach for quantifying tumor extent. Sci. Rep. **7**(1), 1–14 (2017). https://doi.org/10.1038/srep46450

4. B. Lyu, A. Haque, Deep learning based tumor type classification using gene expression data, in *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics* (2018, August), pp. 89–96. https://doi.org/10.1155/2022/4715998
5. M.D. Podolsky, A.A. Barchuk, V.I. Kuznetcov, N.F. Gusarova, V.S. Gaidukov, S.A. Tarakanov, Evaluation of machine learning algorithm utilization for lung cancer classification based on gene expression levels. Asian Pac. J. Cancer Prev. **17**(2), 835–838 (2016). https://doi.org/10.7314/apjcp.2016.17.2.835
6. Z. Wang, M.A. Jensen, J.C. Zenklusen, A practical guide to the cancer genome atlas (TCGA), in *Statistical Genomics* (Humana Press, New York, 2016), pp. 111–141. https://doi.org/10.1007/978-1-4939-3578-9_6
7. E.H. Yau, I.R. Kummetha, G. Lichinchi, R. Tang, Y. Zhang, T.M. Rana, Genome-wide CRISPR screen for essential cell growth mediators in mutant KRAS colorectal CancersGenome-wide CRISPR screen of KRAS-mutant tumor xenografts. Cancer Res. **77**(22), 6330–6339 (2017). https://doi.org/10.1158/0008-5472.CAN-17-2043
8. S. Zuo, G. Dai, X. Ren, Identification of a 6-gene signature predicting prognosis for colorectal cancer. Cancer Cell Int. **19**(1), 1–15 (2019). https://doi.org/10.1186/s12935-018-0724-7

Chapter 13
Brain Tumor Prediction Using Transfer Learning and Light-Weight Deep Learning Architectures

13.1 Introduction

In recent times, brain cancer has been recognized as one of the most deadly brain diseases. Gliomas are the most common type of brain tumor and have a high mortality rate (Bauer et al., 2013). These tumors develop in the brain or spinal cord and are classified into low-grade (LGGs) and high-grade gliomas (HGGs). High-grade gliomas are highly aggressive and typically result in a life expectancy of around two years after diagnosis. Meningiomas, another type of brain tumor, arise from the meninges, while tumors in the pituitary gland are called pituitary tumors. Figure 13.1 shows the various types of brain tumors. Several imaging techniques like MRI, CT, and PET are frequently used to examine brain tumors and other brain conditions. MRI, in particular, is a powerful tool for detecting and treating brain cancer due to its ability to provide detailed images, high contrast in soft tissues, and multi-directional imaging.

Previous approaches in brain tumor detection often faced limitations, particularly in terms of accuracy and computational efficiency. Manual methods struggled with precise tumor type identification, often leading to delayed or less-targeted treatment interventions. Computational methodologies employed in brain tumor prediction involve the utilization of techniques like machine learning, deep learning, and image processing algorithms specifically applied to Magnetic Resonance Imaging (MRI) scans. These methods are designed to automatically detect, classify, and predict brain tumors by extracting patterns and relevant features from these images. Utilizing algorithms such as Convolutional Neural Networks (CNNs), or other classification models, these computational techniques strive to achieve early and precise identification of brain tumors. CNNs have proven effective in analyzing complex image data, recognizing patterns, and facilitating classification tasks, making them valuable in medical image interpretation. However, these computational approaches have inherent limitations. Predicting brain tumors heavily relied on computationally intensive models, posing challenges in their practical deployment on devices

M. A. Wani et al., *Advances in Deep Learning, Volume 2*, Studies in Big Data 12,
https://doi.org/10.1007/978-981-96-3498-9_13

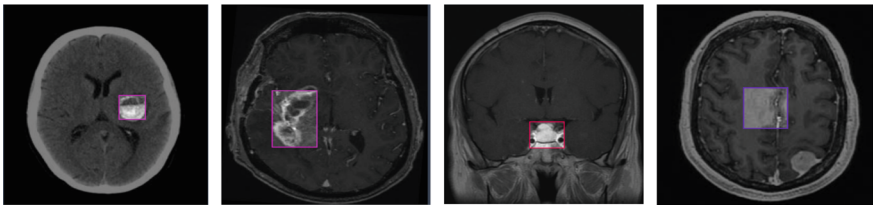


Fig. 13.1 MRI images showcasing different brain tumor types: **a** Glioma, **b** Glioma, **c** Pituitary tumor, and **d** Meningioma

with limited computational capabilities or those requiring energy efficiency. These models demanded substantial computational resources, making them less feasible for implementation on lightweight or battery-oriented devices. Therefore, the shift towards light-weight architectures effectively addresses the challenges posed by computational expenses and resource-intensive models. This transition makes brain tumor prediction models more accessible and practical for deployment in real-world medical settings, particularly on devices with limited computational resources. By integrating Transfer Learning, robust models for classifying brain tumors in MRI images can be developed. This work focuses on the design and development of light-weight architecture, MobileNet V3, for prediction of brain tumors. This fusion of advanced neural network architectures with specialized medical imaging datasets holds the potential to make substantial advancements in identifying and categorizing brain tumors, thereby significantly impacting healthcare practices.

13.2 Light-Weight Architecture

The proposed lightweight system leverages the MobileNet V3 architecture for predicting brain tumor types from MRI scans. The end-to-end framework of the system is shown in Fig. 13.2. The architecture comprises various layers: an input layer receiving the image, convolutional layers for feature extraction using efficient depth-wise separable convolutions, residual and bottleneck blocks enhancing information flow while reducing computational costs, average pooling to reduce spatial resolution, fully connected layers for conversion to probabilities, and a SoftMax layer for class probabilities.

Mobile Net is a lightweight and efficient neural network designed for mobile and resource-constrained environments. It utilizes depth-wise separable convolutions and pointwise convolutions to reduce computational costs while maintaining accuracy. The model processes input MRI scans, which are derived from the Figshare dataset consisting of 3064 T1-weighted MRI scans from 233 patients, categorized into three brain tumor types: glioma (1426 images), pituitary tumor (930 images), and meningioma (708 images). In the preprocessing phase, the MRI images undergo resizing

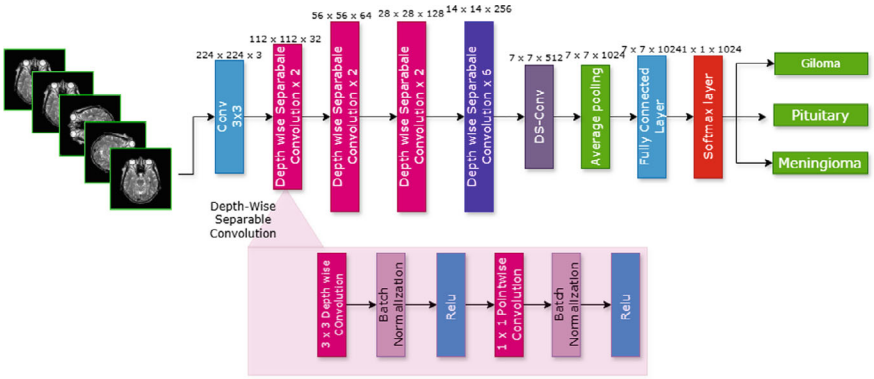


Fig. 13.2 Light-weight architecture for tumor classification

to fit the input size required by Mobile Net (224 × 224 pixels) and are normalized to ensure consistency in the input data. To enhance the dataset's diversity and robustness, data augmentation techniques such as rotation and flipping as shown in Fig. 13.3 are employed. These techniques help in addressing the class imbalance and minimizing overfitting during model training. Additionally, denoising methods like Wiener and median filtering are applied to remove common noise types (Gaussian, salt-and-pepper, and speckle noise) from the MRI scans, ensuring that the input data is clean and reliable.

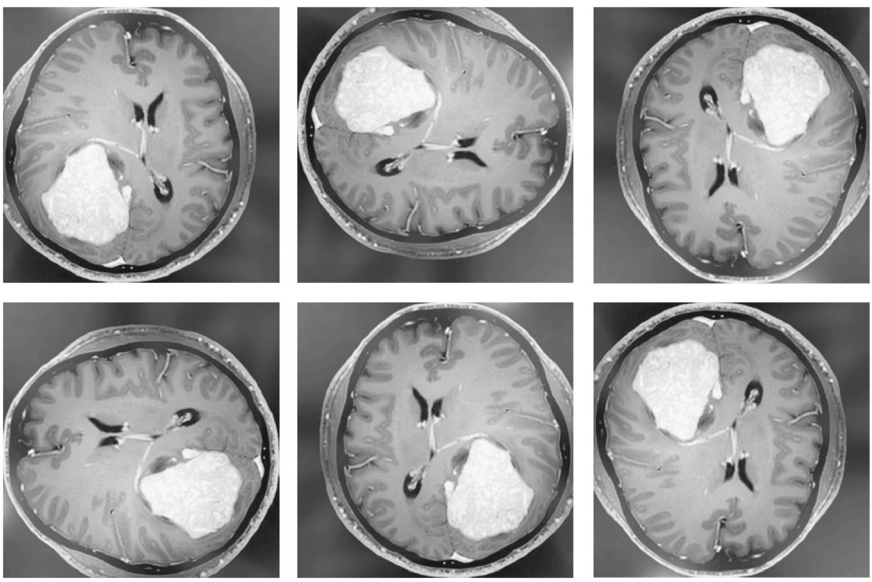


Fig. 13.3 Data Augmentation by rotations and flipping

The input images then pass through several stages of the MobileNet V3 architecture. Initially, the images are processed by a standard convolution layer, followed by multiple layers of depth-wise separable convolutions. These convolutions are essential in extracting spatial features while maintaining computational efficiency. Residual connections are integrated into the architecture to facilitate the flow of information and prevent vanishing gradients in deeper networks. Bottleneck blocks are also employed, consisting of depth-wise and pointwise convolutions, which further reduce the computational load by compressing the output channels. Stride and expansion operations are used within these blocks to balance the resolution and the number of output channels, ensuring that the model remains efficient without sacrificing accuracy. As the data flows through the network, it undergoes global average pooling to reduce the spatial dimensions and is then passed through fully connected layers that transform the feature maps into class probabilities. The final output layer utilizes a softmax activation function to classify the input MRI scans into one of three categories: glioma, pituitary tumor, or meningioma. The model's performance is evaluated using accuracy, precision, recall, and F1 score, ensuring that it effectively distinguishes between the different types of brain tumors. The evaluation metrics are calculated as follows:

$$Accuracy = \frac{True\ Positives + True\ negatives}{Total\ Instances}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F_1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

13.3 Results

Light-weight architectures, including MobileNet, ShuffleNet, EfficientNet, and Xception, offer significant advantages in terms of model size, inference speed, flexibility, and memory and energy consumption, making them ideal for real-time applications on edge devices. Although heavyweight models like AlexNet, VGG16, VGG19, and ResNet50 might achieve marginally higher accuracies on benchmarks such as ImageNet, their computational demands, extensive training times, and larger memory footprints make them less suitable for on-device deployments. The increasing efficiency and competitive accuracy of lightweight models, especially ones like EfficientNet, emphasize a shift towards optimizing performance without compromising on computational resources. Table 13.1 shows a comparison between light-weight

and heavyweight architectures in terms of number of layers in a model, number of parameters, and MAACs in an architecture.

We utilized the MobileNet V3 architecture due to its advantageous characteristics, notably its significantly reduced number of trainable parameters compared to alternative architectures such as ShuffleNet, Xception, and Inception. Additionally, MobileNet V3 boasts a remarkably compact model size, making it particularly well-suited for resource-constrained environments without compromising performance. Utilizing the MobileNet V3 architecture, we conducted classification of brain tumors using the FigShare dataset. The results (see Fig. 13.4) showcase the model's effectiveness in distinguishing between different tumor types. The predicted and actual output comparison is shown in Fig. 13.5.

The precision metric indicates the proportion of correctly identified instances of a specific tumor type out of all instances classified as that type. Meanwhile, recall represents the percentage of actual instances of a particular tumor type that were correctly identified by the model. The F-Score, a harmonic mean of precision and recall, provides a comprehensive measure of the model's overall performance in terms of both precision and recall. The performance evaluation of brain tumor

Table 13.1 Comparison of light-weight and heavyweight architectures

Light-weight architectures	Method	# Layers	# Parameters (M)	MACs
	MobileNet	28	3.5	300 M
	ShuffleNet	11	5.3	830 M
	EfficientNet	88	5.3	390 M
	Xception	71	22.8	16.5G
Heavy architectures	AlexNet	61	60	724 M
	VGG16	138	138	15.5G
	VGG19	144	143	19.6G
	ResNet 50	25.6	25.6	3.86G

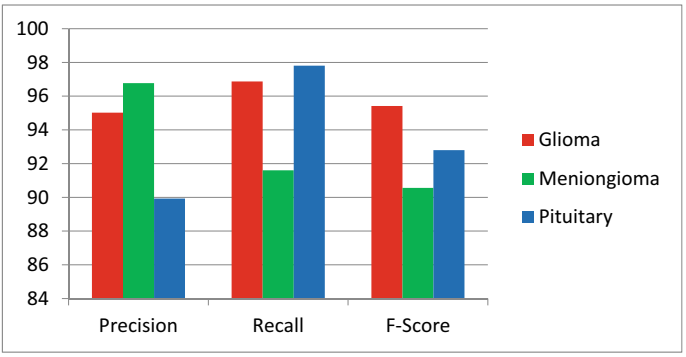


Fig. 13.4 Class-wise performance (Precision, Recall, and F-Score) of brain tumor classification

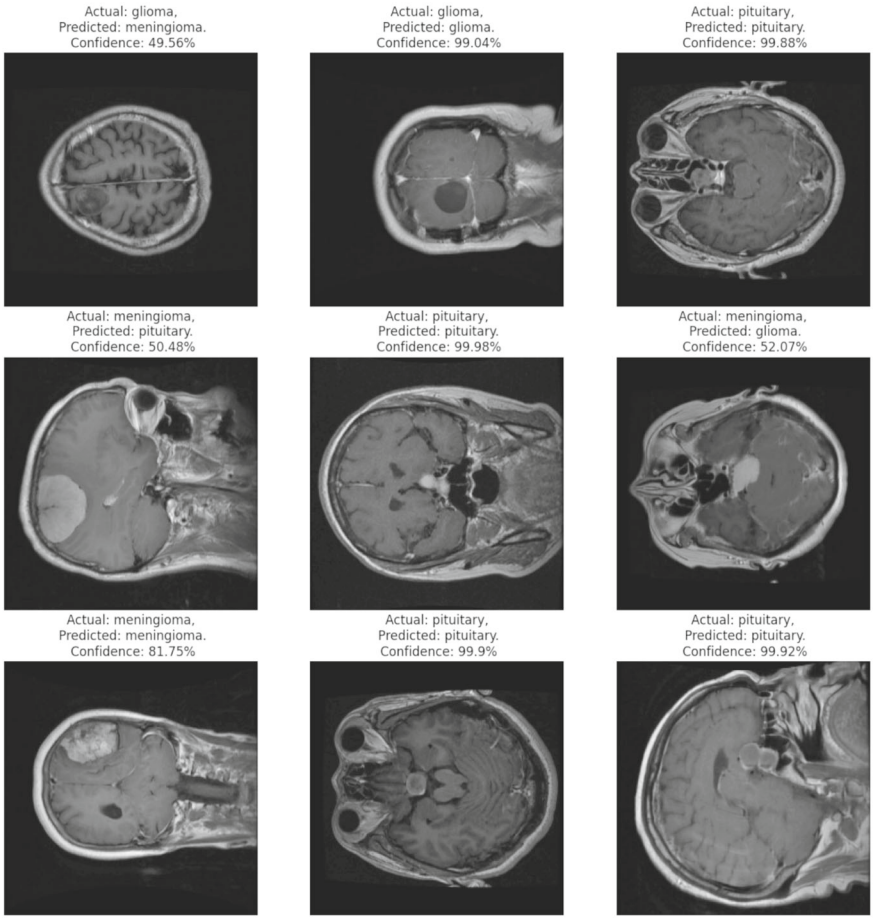


Fig. 13.5 Prediction results of the proposed model

classification using the MobileNet V3 architecture yields promising results across various tumor types. With precision scores of 95.02%, 96.77%, and 89.93% for gliomas, meningiomas, and pituitary tumors respectively, the model demonstrates high accuracy in correctly identifying instances of each tumor type. Moreover, the corresponding recall scores of 96.87, 91.6, and 97.81% highlight the model's ability to capture the majority of actual tumor instances. The balanced F-Scores of 95.42, 90.56, and 92.80% further underscore the overall effectiveness of the MobileNet V3 architecture in accurately classifying brain tumors, affirming its potential as a valuable tool in neuro-oncology for guiding treatment decisions and improving patient outcomes.

Table 13.2 Performance comparison of proposed method with existing methods

Authors	Method	Accuracy (%)
Sultan et al. (2019)	CNN	91.40
Afshar et al. (2019)	Capsule Networks	90.89
Hemanth et al. (2019)	CNN	96.41
Proposed method	MobileNet V3	96.90

The comparison of our proposed transfer learning method with state-of-the-art approaches (See Table 13.2) reveals compelling insights into the performance landscape of brain tumor classification. Sultan et al. (2019) achieved an accuracy of 91.40% using a convolutional neural network (CNN), while Afshar et al. (2019) obtained a slightly lower accuracy of 90.89% with capsule networks. Hemanth et al. (2019) reported a notable accuracy of 96.41% utilizing a CNN architecture. In contrast, our proposed transfer learning approach, utilizing MobileNet architecture, outperforms these methods with an accuracy of 96.90%. These findings underscore the effectiveness of transfer learning in leveraging pre-trained models to enhance classification accuracy, demonstrating its potential as a robust methodology for brain tumor classification tasks. The marginal improvement observed in our method highlights the significance of utilizing transfer learning techniques to achieve superior performance in medical image analysis, particularly in domains with limited labeled data and complex image features.

13.4 Challenges and Future Directions

One of the primary challenges identified in this work is the computational efficiency required for deploying brain tumor classification models on devices with limited resources. While traditional deep learning models like CNNs have proven effective in tumor detection, their heavy computational demands make them less suitable for real-time applications on mobile or edge devices. The use of light-weight architectures like MobileNet V3 is a step towards addressing this challenge, but even with these optimizations, balancing accuracy and computational efficiency remains a critical issue. The need for more efficient data pre-processing techniques, especially in managing noise and performing effective data augmentation, further complicates this task. The pre-processing stage is vital for ensuring the robustness of the model, yet it adds to the computational burden, which is already a significant challenge in resource-constrained environments.

Future work could explore the development of even more optimized architectures that can maintain high accuracy while further reducing computational requirements. This could involve experimenting with novel neural network designs or integrating more advanced transfer learning techniques to leverage existing models without extensive retraining. Another potential direction is enhancing the model's ability to handle a broader range of brain tumor types beyond the three categories currently

addressed, which would involve expanding the dataset and refining the model to ensure it can generalize across different conditions. Furthermore, continued advancements in noise reduction and data augmentation strategies could improve model robustness, particularly in dealing with the diverse quality of MRI scans encountered in clinical settings.

13.5 Summary

This work presents a robust approach to brain tumor classification using the MobileNet V3 architecture, demonstrating the feasibility of deploying lightweight models for medical image analysis on resource-constrained devices. The proposed system effectively addresses the challenges of computational efficiency and accuracy, achieving impressive results on the Figshare dataset, which consists of 3064 T1-weighted MRI scans categorized into glioma, pituitary tumor, and meningioma. Specifically, the model attained precision scores of 95.02% for gliomas, 96.77% for meningiomas, and 89.93% for pituitary tumors, with corresponding recall scores of 96.87%, 91.6%, and 97.81%, respectively. The balanced F-scores of 95.42% for gliomas, 90.56% for meningiomas, and 92.80% for pituitary tumors underscore the effectiveness of the MobileNet V3 architecture in accurately classifying these tumor types. This performance not only surpasses existing methods but also highlights the potential of transfer learning and light-weight architectures in advancing medical diagnostics. The system's high accuracy and efficiency make it a promising tool for real-time applications, particularly in clinical settings where quick and reliable decision-making is crucial.

Bibliography

1. S. Deepak, P.M. Ameer, Brain tumor classification using deep CNN features via transfer learning. Comput. Biol. Med. **111**, 103345 (2019)
2. M.A. Khan, I. Ashraf, M. Alhaisoni, R. Damaševičius, R. Scherer, A. Rehman, S.A.C. Bukhari, Multimodal brain tumor classification using deep learning and robust feature selection: a machine learning application for radiologists. Diagnostics **10**(8), 565 (2020)
3. H. Mohsen, E.S.A. El-Dahshan, E.S.M. El-Horbaty, A.B.M. Salem, Classification using deep learning neural networks for brain tumors. Futur. Comput. Inform. J. **3**(1), 68–71 (2018)
4. J.S. Paul, A.J. Plassard, B.A. Landman, D. Fabbri, Deep learning for brain tumor classification, in *Medical Imaging 2017: Biomedical Applications in Molecular, Structural, and Functional Imaging*, vol. 10137 (2017, March), pp. 253–268. SPIE
5. M.I. Sharif, M.A. Khan, M. Alhussein, K. Aurangzeb, M. Raza, A decision support system for multimodal brain tumor classification using deep learning. Complex & Intell. Syst. 1–14 (2021)