TOPOLOGY OPTIMIZATION FOR CNN USING NEUROEVOLUTION

Mr. Kevin Richard G. Operiano

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2021

การหาทอพอโลยีเหมาะที่สุดสำหรับซีเอ็นเอ็นโดยใช้วิวัฒนาการทางประสาท

นายเควิน ริชาด กาแลง โอเปอริโน

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2564

Dissertation Title	TOPOLOGY OPTIMIZATION FOR CNN USING NEUROEVOLUTION
By	Mr. Kevin Richard G. Operiano
Field of Study	Electrical Engineering
Thesis Advisor	Associate Professor Wanchalerm Pora, Ph.D.
Thesis Co-advisor	Professor Hitoshi Iba, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctor of Philosophy

.... Dean of the Faculty of Engineering
(Professor Supot Teachavorasinskun, D.Eng.)

DISSERTATION COMMITTEE

.... Chairman
(Associate Professor Supavadee Aramvith, Ph.D.)

.... Thesis Advisor
(Associate Professor Wanchalerm Pora, Ph.D.)

.... Thesis Co-advisor
(Professor Hitoshi Iba, Ph.D.)

.... Examiner
(Assistant Professor Suree Pumrin, Ph.D.)

.... Examiner
(Associate Professor Chaodit Aswakul, Ph.D.)

.... External Examiner
(Associate Professor Ekachai Leelarasmee, Ph.D.)

เควิน ริชาด กาแลง โอเปอริโน : การหาทอพอโลยีเหมาะที่สุดสำหรับซีเอ็นเอ็นโดยใช้วิวัฒนาการทางประสาท (TOPOLOGY OPTIMIZATION FOR CNN USING NEUROEVOLUTION) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : รศ.ดร. วันเฉลิม โปรา, อาจารย์ที่ปรึกษาร่วม : ศ.ดร. ฮิโตชิ อิบะ 147 หน้า.

ในช่วงไม่กี่ปีที่ผ่านมามีการพัฒนาสถาปัตยกรรมโครงข่ายประสาทเทียมให้มีความลึกและซับซ้อนขึ้นมากเพื่อเพิ่มประสิทธิภาพของมัน โครงข่ายลึกต้องการชุดข้อมูลขนาดใหญ่และทรัพยากรการคำนวณสูงมาก อย่างไรก็ตามในบางแอพปลิเคชัน เช่น การวิเคราะห์ภาพทางการแพทย์ ชุดข้อมูลมีจำนวนจำกัดและจัดหาได้ยาก ในกรณีนี้โครงข่ายลึกอาจไม่สามารถถูกฝึกได้มากเพียงพอ ซึ่งทำให้เสี่ยงต่อการที่ทำให้มันเข้ากับข้อมูลเกินไป (overfitting) นอกจากนี้ ไม่ใช่ทุกคนที่สามารถเข้าถึงทรัพยากรการคำนวณขั้นสูงได้ การออกแบบโครงข่ายขนาดเล็กแต่มีประสิทธิภาพเทียบเคียงกับโครงข่ายลึกนั้นต้องใช้ความเชี่ยวชาญและความพยายามลองผิดลองถูกอย่างมาก วิทยานิพนธ์นี้จึงขอนำเสนอให้ปรับปรุงวิธีวิวัฒนาการทางประสาทเพื่อค้นหาสถาปัตยกรรมโครงข่ายประสาทเทียมที่เหมาะสมที่สุดโดยอัตโนมัติสำหรับชุดข้อมูลที่กำหนด วิวัฒนาการทางประสาทเป็นวิธีที่ได้รับแรงบันดาลใจจากการคัดเลือกโดยธรรมชาติ และเคยถูกนำไปใช้กับโครงข่ายประสาทเทียมเพื่อเพิ่มประสิทธิภาพสถาปัตยกรรมโดยไม่มีข้อจำกัดด้านปริมาณข้อมูลแล้ว ด้วยการปรับปรุงอย่างพิถีพิถัน วิวัฒนาการทางประสาทสามารถค้นหาสถาปัตยกรรมโครงข่ายประสาทเทียมขนาดเล็กที่เทียบเท่ากับสถาปัตยกรรมเชิงลึกได้ การทดลองที่ได้ดำเนินการยืนยันว่าการใช้วิวัฒนาการทางระบบประสาทที่ปรับปรุงขึ้นหลากหลายรูปแบบสามารถบรรลุความแม่นยำเทียบเท่ากับสถาปัตยกรรมวิวัฒนาการทางประสาททั่วไป สถาปัตยกรรมวิวัฒนาการทางประสาทคงตัว และ RestNet-34 (91.59%, 91.00% and 89.25% ตามลำดับ) จากการทดลอง YOLOv3 ที่ได้รับการวิวัฒนาการมีความลึกเพียง 47 ชั้น มีความแมนยำ 63.8% ในขนะที่ YOLOv3 ดั้งเดิมมีความลึก 106 ชั้น แต่มีความแม่นยำเพียง 52.9% นอกจากนี้ สรุปได้ว่าวิวัฒนาการทางประสาทที่นำเสนอสามารถบรรลุวัตถุประสงค์ทั้งหมดของวิทยานิพนธ์ฉบับนี้ เนื่องจากช่วยในการสร้างสถาปัตยกรรมทางเลือกให้กับโครงข่ายลึก แต่ไม่สูญเสียประสิทธิภาพ

ภาควิชา ......วิศวกรรมไฟฟ้า...... ลายมือชื่อนิสิต ....
สาขาวิชา .....วิศวกรรมไฟฟ้า..... ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก ....
ปีการศึกษา ....2564.......... ลายมือชื่ออาจารย์ที่ปรึกษาร่วม ....

6071451721: MAJOR ELECTRICAL ENGINEERING
KEYWORDS: NEUROEVOLUTION / CONVOLUTIONAL NEURAL NETWORKS / IMAGE CLASSIFICATION / OBJECT DETECTION / TRANSFERABILITY

KEVIN RICHARD G. OPERIANO : TOPOLOGY OPTIMIZATION FOR CNN USING NEUROEVOLUTION. ADVISOR : ASSOC. PROF. WANCHALERM PORA, Ph.D., THESIS COADVISOR : PROFESSOR HITOSHI IBA, Ph.D., 147 pp.

In the recent years, the architecture of the convolutional neural networks has become much deeper and more complex to improve their performance. Consequently, they require large datasets and a considerable amount of computational resources. However, in some applications such as medical imaging analysis, datasets are scarce and difficult to collect. In these cases, deep networks cannot be trained enough, which makes them susceptible to overfitting. Moreover, not all institutions have access to abundant computational resources. Designing a small network that performs as well as a deep network requires expertise and a great effort. Neuroevolution is therefore proposed to automatically discover an optimal convolutional neural network architecture for a given dataset. Neuroevolution is a method inspired by natural selection and previously applied in artificial neural networks to optimize their architecture. With meticulous implementation, neuroevolution can find small convolutional neural network architectures that are on par with deep architectures. The experiments conducted confirm that different proposed neuroevolution implementations can achieve accuracies comparable to those of deep architectures as observed in the accuracies of the generic neuroevolution, steady-state neuroevolution, and ResNet-34 (91.59%, 91.00% and 89.25% respectively). In addition, the neuroevolution performed in a specific object detection application with a limited dataset (i.e., dangerous objects X-ray) has demonstrated that it can find architectures that have modest depths but have performances similar to the deep network. In an experiment, the YOLOv3 with neuroevolution backbone has 47 blocks and 63.8% accuracy, whereas the original YOLOv3 has 106 blocks and 52.9% accuracy. Conclusively, the proposed neuroevolution methods can achieve all the objectives of this dissertation as they effectively aid in creating alternative architectures to deep networks but without performance loss.

Department : ....Electrical Engineering.... Student's Signature ....
Field of Study : ...Electrical Engineering... Advisor's Signature ....
Academic Year : ....2021........... Co-advisor's signature ....

Acknowledgements

I would like to express my deepest gratitude to the almighty God above who unwaveringly sustains me right from the beginning, especially in these trying times that challenged the whole of mankind. I thought I would give up somewhere along the way and go home mentally broken. However, my constant communication with You through prayers and meditations has made me strong, enough to endure the hardships even for one day at a time. And now, here I am, still alive and well, and about to finish my degree unbelievably. If I may share something, I would like to say that it is important to grow our spirituality along with our intelligence because it gives worth to the works of our hands and allows us to overcome even the unimaginable trials in life. It sums up in the words I strive to live by, *ora et labora*, which means prayer and work.

I also would like to thank my Mom and Dad for being my light and pillar. Words cannot express how grateful I am for everything that you have done for me. I dedicate my Ph.D. degree to you. I would like to thank my siblings Erika and Allen for being there. Your presence and amusing small talks gave me the strength to strive to become a better big brother to you. I also acknowledge my relatives and friends who constantly pray for me because that is the only and most important thing we can do given the circumstances.

I would like to give my sincere appreciation to my professors, Aj. Wanchalerm who supported me tirelessly from the start of my Ph.D., and Iba-sensei who provided me insights into the amazing world of evolution. I would like to thank my friends here at Chulalongkorn University and the University of Tokyo for making this chapter of my life enjoyable and memorable.

Last but not the least, I would like to give my gratitude to AUN/SEED-Net, the Department of Electrical Engineering, and the International School of Engineering for the scholarship and trust given.

Contents

List of Tables

List of Figures

Figure Page

CHAPTER I

INTRODUCTION

1.1 Motivation

In recent years, the ability of neural networks to learn is being realized owing to the developments in technology that were not present when neural network was first conceptualized. Applications of neural networks can be found from spam detection in emails, the projection of house prices, to recommender systems in music or video streaming services. The one that caught the imagination of people and made them think that neural networks are truly learning in the literal sense of the word, is the application of convolutional neural network (CNN). CNN is a special type of neural network that is specifically designed to handle images. The simplest application of CNN is image classification, where it tries to put a label on the picture it has "seen". It was developed in the 1990s but did not live to its full potential due to the limitation in the computing resources (LeCun et al., 1998). In 2012, when the computing power was finally ready, CNN has been improved and proposed again (Krizhevsky et al., 2012). Immediately, it was able to outperform all the leading image classification methods at the time. In four years since its resurgence, it was able to surpass human-level performance in image classification (He et al., 2016). From that point onward, the capabilities of neural networks are utilized in many different applications. In one example, CNN is used in facial recognition to unlock electronic devices. In another example, CNN is used in autonomous driving, which is arguably one of the most difficult tasks to program due to the sheer number of variables.

As researchers try to push the limits of CNN capability, the architecture of CNNs continue to grow in depth and complexity. Consequently, they require a large amount of data and computational power to train properly. Big companies such as Google LLC have access to massive amounts of resources to perform large-scale experiments whereas normal institutions such as universities do not. Using a deep CNN architecture to train and use for an application takes a significant amount of time. Furthermore, specialized applications such as tumor detection have a limited amount of dataset images. A deep network trained on a small dataset has several redundant weights that lead to problems such as overfitting, which is the network perfectly identifying the whole training dataset but fails to generalize to datasets it has not seen. Moreover, the deep architectures cannot be deployed into devices that have limited memory (e.g., surveillance cameras). A small network should address the problem but hand-engineering a small network that performs on par with deep networks is not a simple task. The way neural networks learn is not yet fully understood, which makes it difficult to develop exactly a high-performing small architecture. Hence, hand-engineering a network requires numerous trial-and-error and expertise.

During the development of neural networks in the 1980s, there are different methods proposed to train its weights. One of the methods, inspired by natural selection, is called neuroevolution. Unlike the mathematical approaches to network training such as backpropagation, neuroevolution uses a population of candidate solutions, where it mutates the best or fittest solutions repeatedly for several generations to arrive at the optimal solution. Since neuroevolution does not compute anything, it can optimize any parameter of the neural network including the architecture. In 2002, a method called neuroevolution of augmenting topologies (NEAT) was able to successfully optimize the neural network weights and architecture together and subsequently achieved the state-of-the-art performance on the double pole balancing without velocity task, which was a very difficult benchmark (Stanley and Miikkulainen, 2002). Neuroevolution can effectively optimize the architecture of neural networks. Therefore, neuroevolution can also potentially discover small CNN architectures that have performances close to those of the conventional deep CNN networks.

In this study, neuroevolution methods that discover optimal CNN architectures are explored. The techniques proposed are inspired by the intuitions in the NEAT paper (Stanley and Miikkulainen, 2002). Moreover, one of the advantages of using a conventional deep network is transfer learning. The weights of the deep networks trained with large-scale datasets are available online. Using the trained weights (e.g., as pretraining weights) typically leads to training and performance boost. However, a customized network architecture developed by neuroevolution cannot utilize the trained weights because its architecture is different. Thus, various techniques to utilize the trained weights on a neuroevolution-produced network are experimented. In addition, alternative ways to transfer the learning of a trained network to the neuroevolution-produced network are emphasized. Finally, the ability of neuroevolution to optimize the architecture of a CNN for specific applications is examined.

1.2 Objectives

1. Combine CNN and Neuroevolution to optimize the weights and architecture automatically.
2. Employ transfer learning intuition to reduce CNN-Neuroevolution training time.
3. Apply the created algorithm to a specific problem. (e.g. classification of microscopic images)

1.3 Scope of Work

1. Produce a novel Convolutional Neural Network Architecture developed using transfer learning intuition and neuroevolution techniques.
2. Use good practices in Deep Learning to improve the accuracy and reduce the computational time.
    (a) Achieve an accuracy of ± 5% compared to CGP-CNN algorithm, a leading Neuroevolution-CNN combination using the CIFAR-10 dataset.
    (b) Achieve 70% training time when compared to that of the CGP-CNN algorithm.
    (c) Achieve at least 85% accuracy from the specific application of the algorithm

1.4 Summary and Structure of the Thesis

The rest of the dissertation is organized as follows. The next chapter reviews the related work and theories in artificial neural networks and convolutional neural networks. Moreover, the datasets and data augmentations commonly used in CNN are discussed. Methods of learning from a trained network are also explored in this chapter. Furthermore, neuroevolution and neuroevolution approaches to CNN are examined. In Chapter 3, the CNN experiments on specialized applications are described. Here, methods are proposed to improve the training and utilization of the CNN. In Chapter 4, different neuroevolution approaches to CNN, methods to utilize the learning of a trained network, and neuroevolution application methods are proposed. Chapter 5 describes the experiments that evaluate the effectiveness of the approaches and methods proposed in the previous chapter. Specific applications of the neuroevolution-produced CNN are demonstrated in Chapter 6. Finally, the last chapter discusses the limitations and future directions of the study and then concludes the dissertation.

CHAPTER II

BACKGROUND KNOWLEDGE AND RELATED WORK

In this chapter, the literature and theories from machine learning methods to neuroevolution approaches to CNN are discussed. In Section 2.1, machine learning methods are introduced. Section 2.2 describes a brief history of the artificial neural network development and also the theory that accompanies it. The development of a special type of neural network called convolutional neural network in the 1990s and its resurgence in 2012 is discussed in Section 2.3. Additionally, the basic building blocks and the operation of CNN are examined. Different applications of CNN are also reviewed in this section. Furthermore, this section analyzes the vulnerability of CNN. In Section 2.4, the datasets and data augmentations, which are fundamental components of CNN training are discussed. The techniques that utilize a trained CNN to effectively train a different network are reviewed in Section 2.5. Evolutionary algorithms which optimize the connection weights and architectures of the artificial neural network are explored in Section 2.6. Finally, the application of the evolutionary algorithms to CNN is examined in Section 2.7.

2.1 Machine Learning

Machine learning is a significant contributor to the information age. It is particularly ubiquitous in digital devices (e.g., computers, smartphones) that it has become part of everyday life. Early machine learning applications are seen in spam detection in emails, price predictions, etc. Afterward, it has developed to handle very complicated tasks such as autonomous driving and natural language processing. Machine learning is first defined as the study of giving computers the ability to learn without any explicit programming (Samuel, 1959). In another definition, machine learning is the study of algorithms that automatically improve with experience and data usage (Mitchell, 1997). In the early developments of machine learning, the classical machine learning algorithms are thoroughly explored, which include linear regression, logistic regression, k-nearest neighbors (k-NN), decision trees, support vector machines (SVM), etc. In linear re-

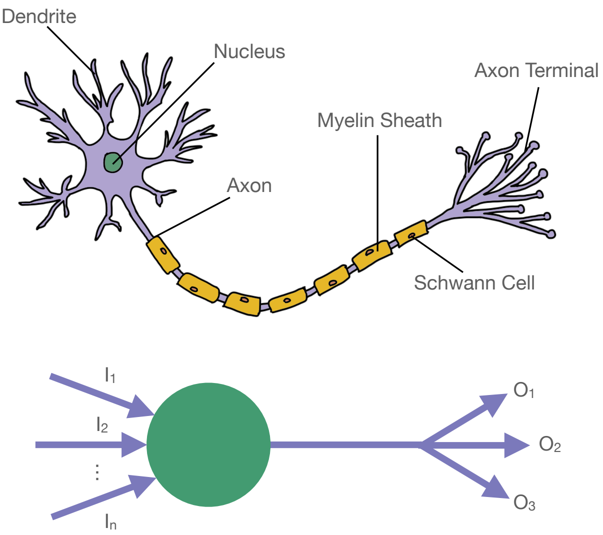


Figure 2.1: Comparison of a neuron and a neural network node.

gression, it uses a linear function to fit datasets (e.g., house prices) and predict values given inputs (Kenney and Keeping, 1962). Logistic regression is a simple classification algorithm that predicts the label of an input using binary (0 or 1) outputs (Nelder and Wedderburn, 1972). The k-NN uses local approximation for classification and regression applications (Altman, 1992). Decision tree is based on a tree structure that also solves classification and regression problems (Quinlan, 1986). SVM uses kernels to derive a new hyperplane for an entire training data and linearly separate them into their respective labels (Boser et al., 1992). However, there is an algorithm under machine learning that has pushed the boundaries in the field. This immensely popular algorithm is called neural network. Neural network is a model inspired by the neurons inside the human brain (Rosenblatt, 1957; Minsky and Papert, 1969). Combined with the backpropagation training method (Rumelhart et al., 1986), a huge amount of data, and powerful computing machines, neural network enables a wide variety of applications ranging from simple face detection to being the core in autonomous driving, which is very sophisticated due to the sheer number of parameters (Yurtsever et al., 2020).

2.2 Artificial Neural Network

Artificial neural network (ANN), fully connected neural network, or simply neural network is a machine learning algorithm whose architecture is inspired by neuron connections in the human brain as shown in Fig. 2.1. It is a supervised learning algorithm that

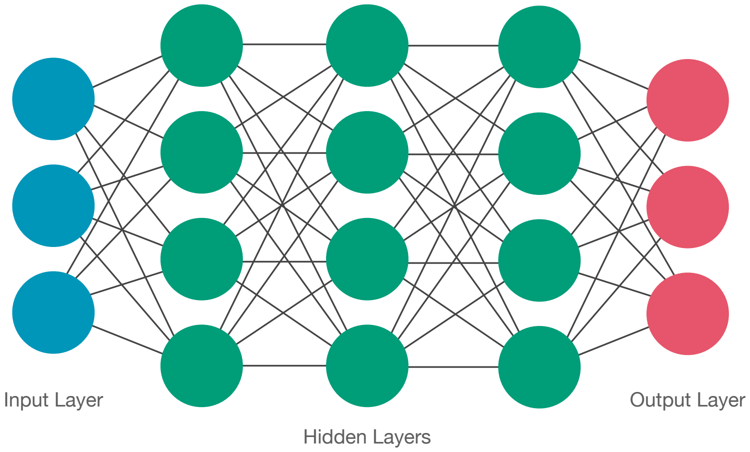


Figure 2.2: Artificial Neural Network Architecture Example.

needs an abundant amount of data to fully utilize its capabilities. With the development of powerful computing machines and the availability of massive amounts of data in the past decade, neural network has achieved tremendous success in the many complex applications (e.g., language translation) where classical machine learning algorithms have failed. A neural network is composed of an input layer, hidden layers, and an output layer. The input layer consists of nodes that correspond to the number of inputs. The number of layers and nodes in the hidden layers depends on the design of the architecture. Similar to the input layer, the number of nodes in the output layer corresponds to the output required by the problem. In every layer, the nodes are all connected to the nodes of the next layer as depicted in Fig. 2.2. Essentially, each node or neuron in a neural network represents a mathematical function by computing the weighted sum of its input. A larger weight implies more impact on the neuron output. After the weighted sum is obtained, it is fed to a nonlinear function commonly known as the activation function (e.g., Sigmoid, ReLU) to model complex functions while controlling the numerical representation (i.e., limit to computable values) (Hinton et al., 2006). The network learns by optimizing the weights of the nodes using backpropagation with respect to the network cost (Rumelhart et al., 1986). Owing to the sequence of complex functions of all the nodes in every layer, the neural network can handle difficult and complicated applications.

2.2.1 Developments in ANN

Although the monumental progress in neural network is recently being realized in the last decade, the idea itself is as old as other machine learning algorithms. In 1958, Frank Rosenblatt published a report describing the primitive form of neural network called perceptron (Rosenblatt, 1958). With the aim of understanding the human brain, Rosenblatt experimented on neural network with one or two trainable layers only. He trained the network by adjusting the input weights of the neurons depending on their effects on the outputs. The weights that contribute to the correct neural network classification were increased whereas the other weights were decreased. However, due to the simplicity of its network architecture, perceptron could not model a complex real-world application that it was supposed to do (Minsky and Papert, 1969). Aside from the computational limitation at the time, the simple hill-climbing algorithm used to train perceptrons could not scale to deep networks. In 1986, Rumelhart et al. (1986) introduced a method called backpropagation, which could train deep networks effectively. Backpropagation uses partial derivatives to compute the gradients of the error function starting from the output layer propagating backward until the second layer. The gradients from the previous layer are reused to compute the gradients of the current layer. Using the gradients to assess the contribution of the weights to the correct and wrong answer, the weights are nudged accordingly to push the network toward the correct answer. The introduction of backpropagation has allowed deep neural network to develop good internal representations of datasets similar to the hand-engineered features that required expertise to design. However, the problem of restricted computational resources persisted and thereby limited the usability of the deep networks at the time. In 2012, when the computing power of low-cost processors had been enough, a deep neural network called AlexNet delivered a breakthrough performance in a renowned image classification competition (Krizhevsky et al., 2012). The deep neural network combined with a large dataset and computing power through parallel processing has greatly reduced the error rate to 15.3%; whereas the second-best method is approximately 10% apart. Since then, neural networks have been utilized in a wide variety of applications. Different variations of neural networks are employed to tackle different problems (e.g., convolutional neural network for images and recurrent neural network for natural language processing).

2.2.2 Implementation

Artificial neural network computes the prediction on an input by forward propagation. To demonstrate a simple forward propagation, consider an architecture composed of an input layer, a hidden layer with one node, and an output layer as illustrated in

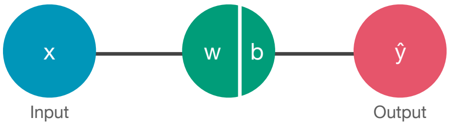


Figure 2.3: Simple Artificial Neural Network Architecture.

Fig. 2.3. The prediction is computed by multiplying the input x with the weight w and then adding the bias b. The result is modified by an activation function σ. Formally, the prediction $\hat{y}$ is calculated as

$$\hat{y} = \sigma(wx + b), \tag{2.1}$$

where the activation function σ can be a sigmoid function or a rectified linear unit (ReLU). In recent applications, the ReLU is favored due to its ability to decrease the training time (Nair and Hinton, 2010). The sigmoid function and the ReLU function are defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

and

$$\sigma(z) = max(0, z) \tag{2.3}$$

respectively. Using gradient descent, the network is optimized by minimizing the cost function or the loss function

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}), \tag{2.4}$$

where m is the dataset size.

To train the simple neural network shown in Fig. 2.3 or minimize the cost function, the backpropagation algorithm is employed. The concept behind it is to adjust the weights and biases of the network that heavily influence the prediction of the network. The derivatives of the loss function with respect to the weights and biases are computed to be able to adjust the weights and biases such that the loss decreases. Before explaining backpropagation, as a shorthand, the equation that is fed to the activation function is represented as

$$z = wx + b \tag{2.5}$$

and the prediction as

$$\hat{y} = a = \sigma(z). \tag{2.6}$$

Normally, the prediction is not always activated but for the simplicity of the example, it is activated. The loss function for an input example is also rewritten as

$$L(a, y) = -(y \log(a)) + (1 - y) \log(1 - a)). \tag{2.7}$$

Through the equations (2.5), (2.6) and (2.7), the required derivatives

$$\frac{\partial L(a, y)}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial a}{\partial z} \frac{\partial L(a, y)}{\partial a} \tag{2.8}$$

and

$$\frac{\partial L(a, y)}{\partial b} = \frac{\partial z}{\partial b} \frac{\partial a}{\partial z} \frac{\partial L(a, y)}{\partial a} \tag{2.9}$$

can be calculated using chain rule. The derivative of the loss $L(a, y)$ with respect to a is

$$da = \frac{\partial L(a, y)}{\partial a} = -\frac{y}{a} + \frac{1 - y}{1 - a}, \tag{2.10}$$

where da is used for shorthand. The derivative of a with respect to z is computed as

$$\frac{\partial a}{\partial z} = a(1 - a), \tag{2.11}$$

and using this value, the derivative of the loss $L(a, y)$ with respect to z is calculated as

$$dz = \frac{\partial L(a, y)}{\partial z} = \frac{\partial L(a, y)}{\partial a} \cdot \frac{\partial a}{\partial z} = a - y, \tag{2.12}$$

where dz is employed as shorthand. Combining dz with $\frac{\partial z}{\partial w}$ and $\frac{\partial z}{\partial b}$, the derivative of the loss $L(a, y)$ with respect to w is

$$dw = \frac{\partial L(a, y)}{\partial w} = x \cdot dz \tag{2.13}$$

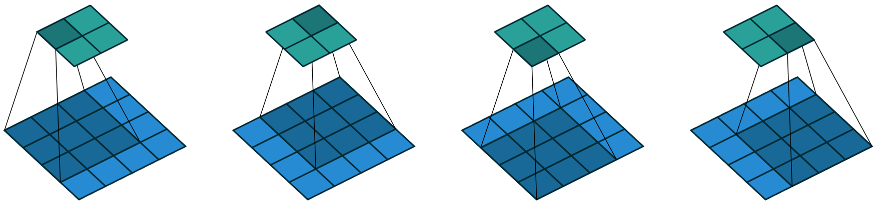


Figure 2.4: A 3 × 3 filter sweeping across a 4 × 4 image resulting in a 2 × 2 output (Dumoulin and Visin, 2016).

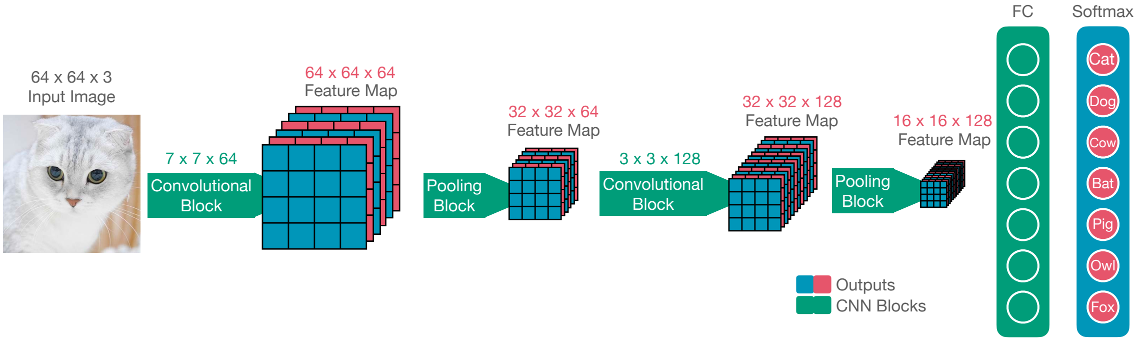


Figure 2.5: Convolutional Neural Network Architecture Example.

and the derivative of the loss $L(a, y)$ with respect to b is

$$db = \frac{\partial L(a,y)}{\partial b} = dz, \tag{2.14}$$

where dw and db are their shorthands respectively. The weight and bias are updated by subtracting the dw and db to the their current values. The respective computations are defined as

$$w = w - \alpha dw \tag{2.15}$$

and

$$b = b - \alpha db, \tag{2.16}$$

where α is the learning rate. The backpropagation algorithm runs for many epochs until the weight and bias of the network converge.

2.3 Convolutional Neural Network

The convolutional neural network (CNN) is a type of neural network that specializes in image applications. Image applications are problematic to ANNs because ANNs take an image as per pixel input. For example, a small 200 × 200 RGB image is equivalent to $200 \times 200 \times 3 = 120{,}000$ input nodes. When the input nodes are multiplied with the first hidden layer, which may consists of a relatively low number of 1,000 nodes, the number of parameters suddenly becomes 120,000,000 (without the bias term). The computational cost of the ANN in this case is considerably expensive. As a solution, CNN uses filters or kernels as its learnable parameters instead of node weights. In an image, a pixel typically has high correlations with its neighboring pixels and this can be exploited by sharing the parameters of the pixels through sweeping a filter across the image as shown in Fig. 2.4 (LeCun et al., 1998). A layer in CNN is composed of a group of filters called the *convolutional block*. By sharing the features among image pixels using convolutional blocks instead of one-to-one connections, the computational cost drops dramatically. To illustrate the difference in parameter count, a common first convolutional block may have a filter size of 7 × 7 with 64 channels (i.e., the number of filters). The number of parameters in the first layer is $7 \times 7 \times 3$ (RGB channels) $\times$ 64, which is only 9,408 (excluding the bias term) compared to 120,000,000. Aside from the convolutional block, another basic block used in CNN is called the *pooling block* or *subsampling*. The pooling blocks provide translational invariance, which makes the network robust to small spatial differences in the image by decreasing the image resolution. Effectively, the network can see larger portions of the image due to the larger receptive field of the network (through decreased image resolution). Moreover, this allows the network to develop more complex representations of the image as the depth of the network increases. A simple CNN architecture is exhibited in Fig. 2.5. In this figure, the CNN architecture consists of two sets of convolutional block-pooling block combinations and a fully connected (FC) layer with the softmax at the end.

Visualizing the results on each layer shows how the CNN architecture learns as depicted in Fig. 2.6. In the first layer, the network learns the basic structure of the images such as horizontal edges and vertical edges. The filters in this layer function as edge detectors (e.g., Sobel X, Sobel Y). On the succeeding layer, the network starts to learn complex structures such as circular and rectangular shapes, which may have been

Figure 2.6: Visualization of the CNN features on every layer (Zeiler and Fergus, 2014).

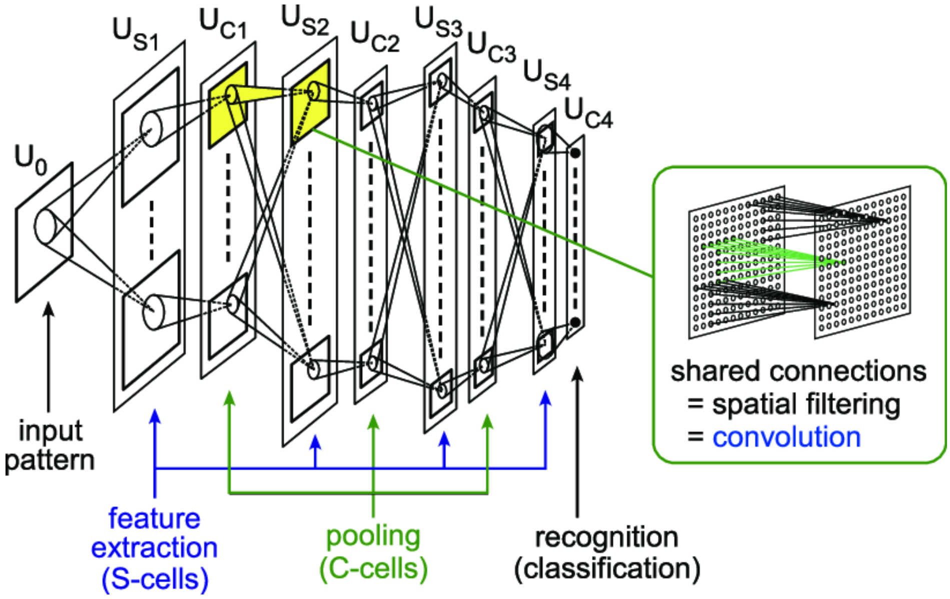


Figure 2.7: Neocognitron Architecture (Fukushima, 1980).

extracted from wheels, balls, etc. Toward the last layers, it can now detect more complicated features such as dog faces and flowers. Thus, the CNN has a structured operation, in which it learns from the basic structures of an image to complicated structures (Zeiler and Fergus, 2014).

2.3.1 Developments in CNN

The roots of CNN can be traced back to the neocognitron. In 1980, Kunihiko Fukushima proposed the neocognitron, which is inspired by the visual nervous system found in vertebrates. The neocognitron structure, as seen in Fig. 2.7, is composed of alternating simple cells (S-cells or lower order hypercomplex cells) and complex cells (C-cells or higher order hypercomplex cells), which mimic the processing that occurs in the biological simple cells and complex cells. The purpose of the S-cells, similar to a convolutional block, is to perform feature extraction, whereas the C-cells, similar to a pooling block, provide tolerance to position changes. Together, the local features extracted by the S-cells (e.g., cat leg) are integrated by the C-cells to form global features (e.g., cat) (Fukushima, 1980).

The original implementation of the CNN, where its name is coined, is the LeNet developed by Yann Lecun and his team between 1989 to 1998. It is experimented with

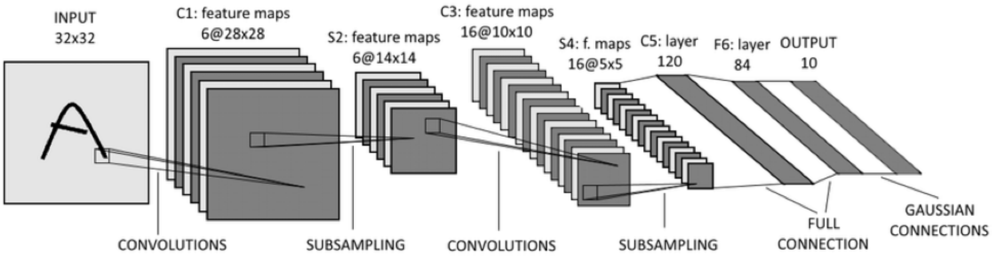


Figure 2.8: LeNet-5 Architecture (LeCun et al., 1998).

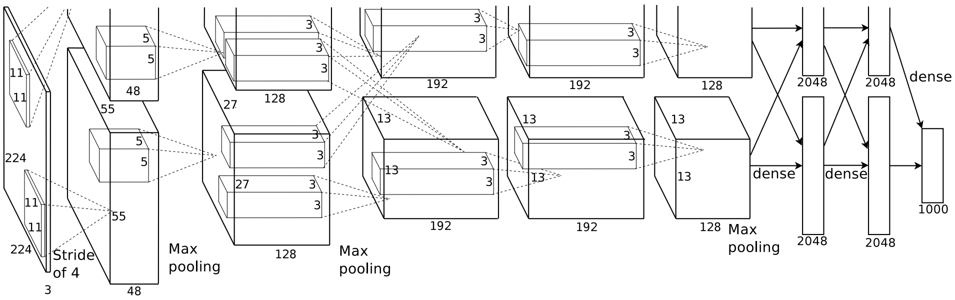


Figure 2.9: AlexNet Architecture (Krizhevsky et al., 2012).

the handwritten single-digit dataset commonly known as the MNIST dataset. In this work, the convolutional blocks and pooling blocks (subsampling) are proposed as the basic building blocks of CNN. The LeNet architecture, as shown in Fig. 2.8, consists of Conv-Pool-Conv-Pool-FC-FC blocks. Each convolutional block has a filter size of 5×5, and the pooling type used then is average pooling. The total number of parameters is 60,000 (LeCun et al., 1998).

The potential of CNN is publicly realized from the work of Alex Krizhevsky on his AlexNet in 2012. It marks the beginning of CNN superiority over the traditional image processing techniques in various image-related applications. AlexNet is a reinforced CNN architecture with more layers as illustrated in Fig. 2.9, and has a massive 60 million parameters. In that year, it is used as an entry to the ImageNet Competition, where it shatters the object recognition records by logging in 15.3% error rate, which is lower than that of the second best by almost 11% (Krizhevsky et al., 2012). Since then, enhanced CNN variations have dominated the competition in the ensuing years and subsequently reduced the error rate further to 3.57%, at which the CNN exceeds human-level performance (He et al., 2016).

Aside from the huge amount of training parameters of the AlexNet architecture,

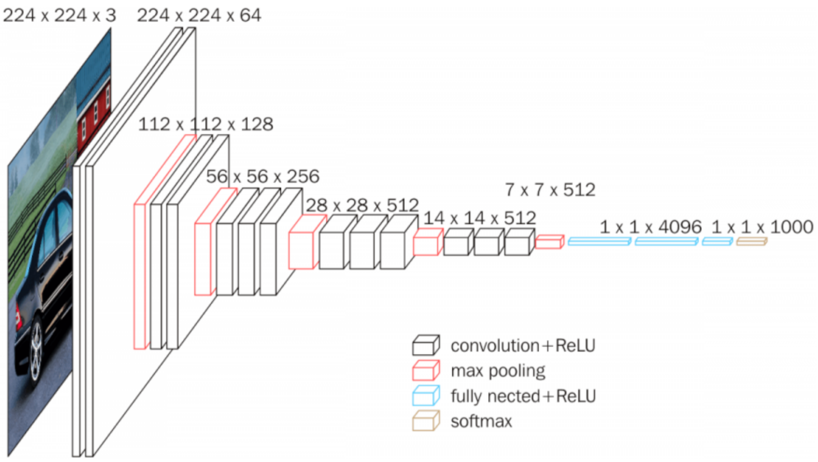


Figure 2.10: VGG Architecture (Simonyan and Zisserman, 2014b).

which is mainly credited for its remarkable performance, there are also several good practices that Krizhevsky et al. (2012) introduced to improve the performance of AlexNet. Their key contributions are the following: First, they employ the ReLU (rectified linear unit) (Nair and Hinton, 2010) as the activation function instead of *tanh*, which is the standard at the time. Using ReLU, the AlexNet can reach 25% error rate six times faster than when it uses *tanh*. Second, they implement the training process on multiple graphics processing units (GPU) when the memory is only 3GB, since the training set contains 1.2 million images. As a result, the top-1 and top-5 error rates are reduced by 1.7% and 1.2% respectively compared to using only one GPU. Third, they use overlapping pooling, which has a stride smaller than the filter size, instead of the regular pooling, which has a stride equal to or larger than the filter size. Overlapping pooling reduces the top-1 and top-5 error rates by 0.4% and 0.3% respectively. Fourth, to solve the problem of overfitting, they introduce data augmentation and dropout. Data augmentation extends the training set by mirroring the images, translating the images, and altering the image intensities using PCA (principal component analysis). It reduces the top-1 error rate by more than 1%. Dropout, on the other hand, *turns off* the neurons in the FC layers randomly with a probability of 0.5. It helps eliminate the reliance of the network on strong neurons and regulates the contribution of all the neurons, thereby reducing overfitting and training time. Lastly, other techniques such as momentum and decaying learning rate are likewise used to make the network converge faster.

After the success of AlexNet, several works on CNN are developed to obtain a better performance than the AlexNet. In 2014, one of the main networks developed is called

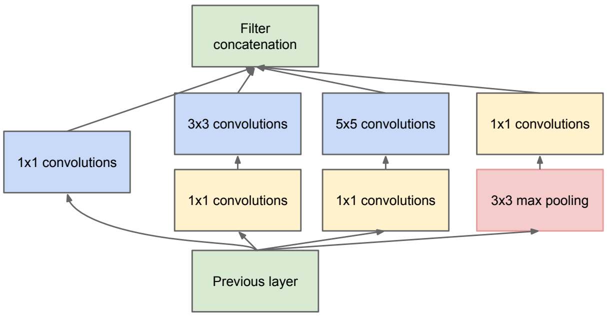


Figure 2.11: Inception Module (Szegedy et al., 2015).

VGG network (VGG-Net). The objective of VGG-Net is to significantly increase the depth of the network and reduce the training time. VGG-Net has 138 million parameters, which is twice the size of AlexNet. While still following the best practices established in the AlexNet implementation, the key difference in VGG-Net is the use of small 3 × 3 filters throughout the architecture instead of the varying filter sizes (11 × 11, 5 × 5, and 3 × 3). Whilst the filters are small, the number of channels is increased considerably to replicate the receptive field of AlexNet. The VGG-Net architecture is shown in Fig. 2.10. The simplicity and uniformity of VGG-Net are its novel contribution. In terms of performance, it yields a top-5 error rate of 7.32% in the ImageNet competition, earning them the first runner-up (Simonyan and Zisserman, 2014b).

Another main CNN developed in 2014 is the GoogLeNet network. It is even deeper than the VGG-Net but it employs a module called *inception* to require far fewer parameters (4 million). The inception module, as shown in Fig. 2.11, combines convolutional blocks with different filter sizes to avoid deciding on a specific size. It simply allows the training process to decide which parameters are to be utilized. The outputs from the different convolutional blocks are combined by concatenating them together, thereby increasing the channel size. Since after the concatenation the output size becomes large, it is regulated by using a 1 × 1 convolution (Lin et al., 2013), which is the critical operation for keeping the number of parameters low. The 1 × 1 convolution manages the dimensionality by reducing the channel size and consequently reduces the computational cost on the 3 × 3 and 5 × 5 convolutions. Furthermore, it also attenuates redundant filters. The GoogLeNet architecture is composed of stacked inception modules as shown in Fig. 2.12. It also uses auxiliary classifiers during training, as seen in the additional softmax layers, to ensure that the layers in the middle are learning properly. In terms of

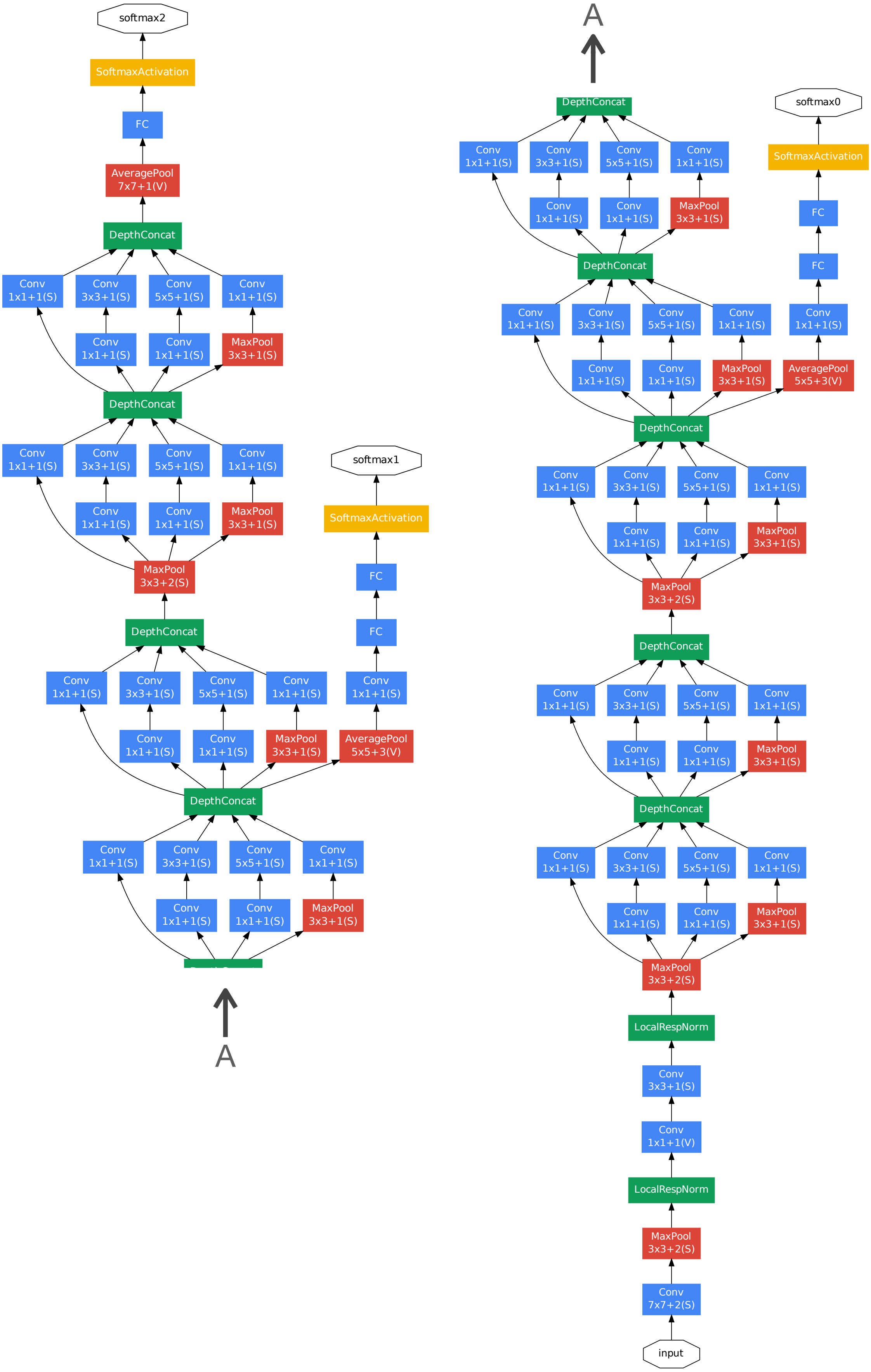


Figure 2.12: GoogLeNet Architecture made of stacked Inception Modules (Szegedy et al., 2015).

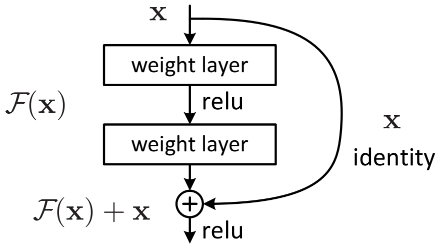


Figure 2.13: Residual Block (He et al., 2016).

performance, it achieved a top-5 error rate of 6.67%, which beats the VGG-Net, and wins the ImageNet competition in that year.

The network that paves way for a very deep CNN is the residual neural network (ResNet). Similar to GoogLeNet, which breaks the convention of simply stacking convolutional and pooling blocks, ResNet also presents a key innovation in the architecture through the introduction of *skip connections*. The obstruction that prevents the network from becoming very deep is the so-called *vanishing gradient*. As the gradients are backpropagated to the earlier layers of a deep network, the gradients become smaller and smaller due to the repeated gradient approximation. Consequently, the vanishing gradient saturates or even degrades the performance of the network. As a solution, ResNet uses skip connections or identity shortcuts as shown in Fig. 2.13 to fast track the input x over the two weight layers $F(x)$. The main idea is that if the two weight layers in the middle cannot learn any good features, they can at least become zero, and thus, the output becomes simply x. Applying this idea to a network implies that for every additional residual block to the network, the output of the network can always revert to the output when there is no additional residual block if the additional residual block cannot add anything of value. The ResNet architecture is shown in Fig. 2.14. ResNet is an important innovation in CNN because it allows networks to become very deep (e.g., 150 blocks) through skip connections. There are no apparent downsides to having very deep networks aside from the high computational cost. As for the performance of ResNet, it topped the 2015 ImageNet competition with the top-5 error rate of 3.57%, which surpasses the human-level performance.

Building on the success of ResNet, the next CNN architecture development called DenseNet utilizes skip connections even further. DenseNet uses a dense block, as depicted in Fig. 2.15, which is composed of convolutional blocks. In a dense block, every layer is

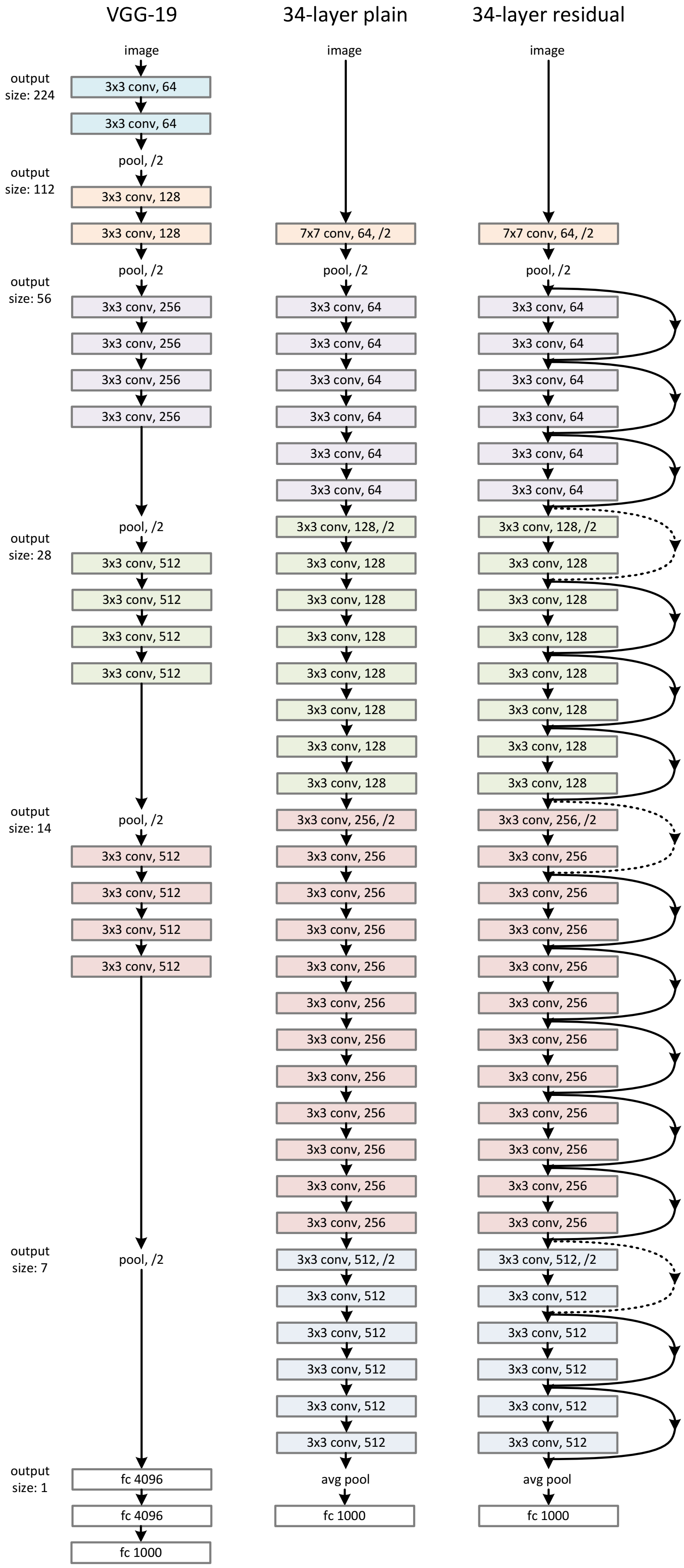


Figure 2.14: ResNet Architecture (He et al., 2016).

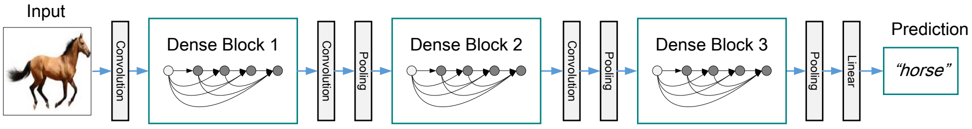


Figure 2.15: DenseNet Architecture (Huang et al., 2017).

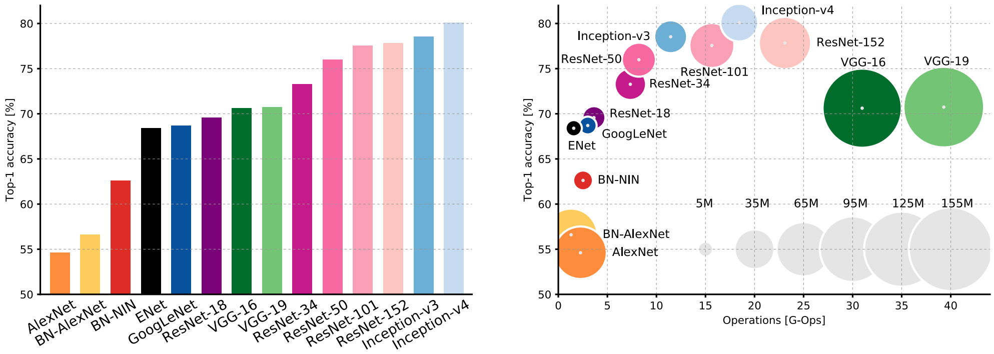


Figure 2.16: Different CNN networks and their performances on ImageNet dataset. The size of the circle on the right graph indicates network parameters size (Canziani et al., 2016).

connected to all its preceding layers and obtains its input through the concatenation of the ouputs of preceding layers. The concatenated input provides each layer with the *collective knowledge* of the preceding layers. The connections on every layer allow the network to have fewer channels, and thereby, lower the computational cost and total parameters size. About the performance, DenseNet has outperformed the ResNet in the CIFAR-10 dataset with 3.6% to 6.41% respectively while having less than half of the parameters of ResNet (Huang et al., 2017).

The networks discussed are tested in the same environment to objectively compare their performances (Canziani et al., 2016). The summary of the discussed networks performances is illustrated in Fig. 2.16.

2.3.2 Implementation

Despite the various changes and innovations in the CNN architecture, the basic components of CNN remain the same. These components are a convolutional block and a pooling block. In between the connection of these blocks, an activation function operation (usually ReLU) and normalization are performed as shown in Fig. 2.17. A simple CNN architecture can have two sets of convolutional-pooling block combinations, an FC layer, and a softmax, wherein the output of the network is turned into a probability of classes

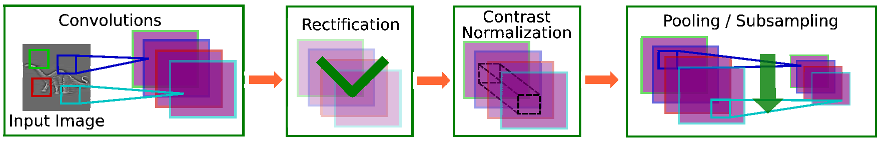


Figure 2.17: Typical Components of CNN (Jarrett et al., 2009)

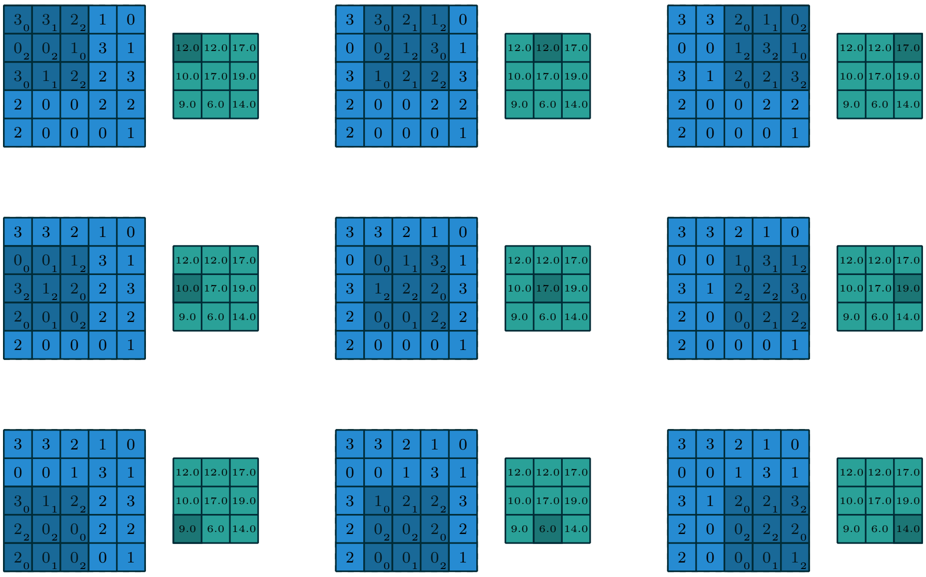
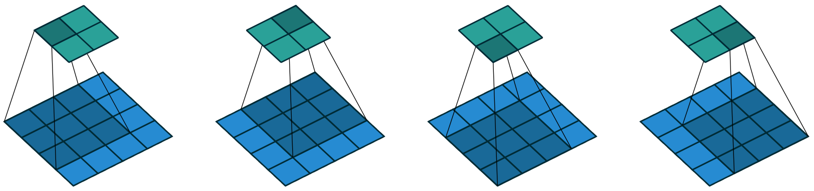


Figure 2.18: Convolution operation. The numbers on the rightmost bottom corner of the 3 × 3 dark blue box are the filter values. The blue box and the green box represent the input and output respectively. The output is obtained by multiplying the overlapping filter values and image pixel values and then summing the products (Dumoulin and Visin, 2016).

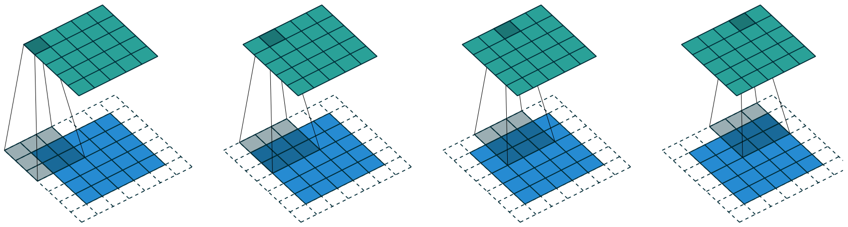
or predictions as shown in Fig. 2.5.

2.3.2.1 Convolutional Block

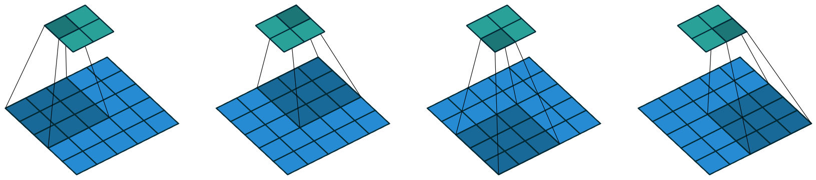
The correlation of image pixels and their neighboring pixels is exploited to reduce the parameters of neural network by using convolution filters or filters instead of the one-to-one connections as the learnable parameters or weights. To perform the convolution operation, the filter is first placed on the top leftmost corner of an image as shown in Fig. 2.18 and multiplying the overlapping image pixels values and filter values. Then, the products are summed to obtain the output. The process continues by sliding the filter one pixel at a time until the whole image pixels are passed. Typically, a convolutional block contains a lot of filters to extract different features of the input. The number of filters in a convolutional block is called channels. The filter dimension depends on the channels of the input. For example, an input image usually has 3 channels. Therefore, the



a) Input: 4x4, Filter: 3x3, Padding: valid, Stride: 1, Output: 2x2



b) Input: 5x5, Filter: 3x3, Padding: same, Stride: 1, Output: 5x5



c) Input: 5x5, Filter: 3x3, Padding: valid, Stride: 2, Output: 2x2

Figure 2.19: Different padding and stride combinations (Dumoulin and Visin, 2016).

filter dimension is $f \times f \times 3$, where f is the filter size of choice. As observed in Fig. 2.18, the size of the output decreases after performing the convolution. If the network is deep, this becomes a problem because the output continues to shrink until it cannot be convolved. As a solution, padding is used to extend the image and avoid a decrease in size after a convolution operation. This type of padding is called *same* padding. If there is no padding used, it is called *valid*. In very deep CNN architectures, same padding is always used with convolutions. Another important detail in the convolution operation is the *stride*. As mentioned earlier, the filter is slid by a pixel by default but this can be adjusted according to preference. For example, the filter can slide every 2 pixels (stride of 2), which effectively reduces the output size in half. Different paddings and strided convolution are shown in Fig. 2.19. In summary, the basic convolutional block settings that need to be configured are the filter size, number of channels, padding, and stride.

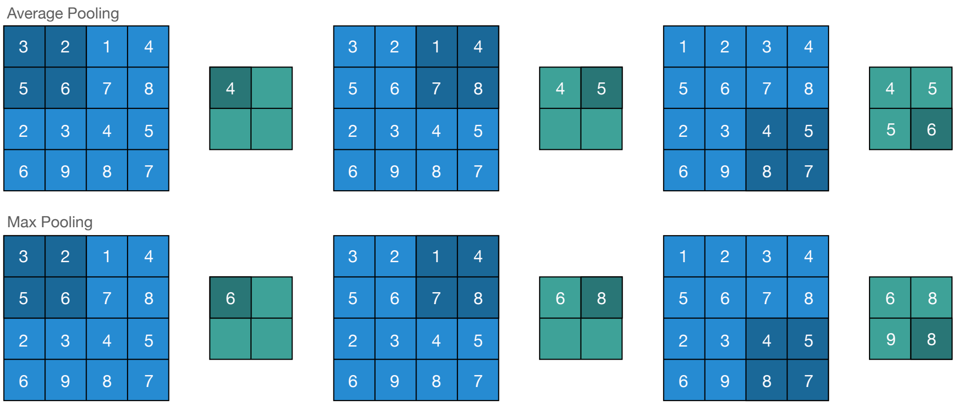


Figure 2.20: Average Pooling and Max Pooling.

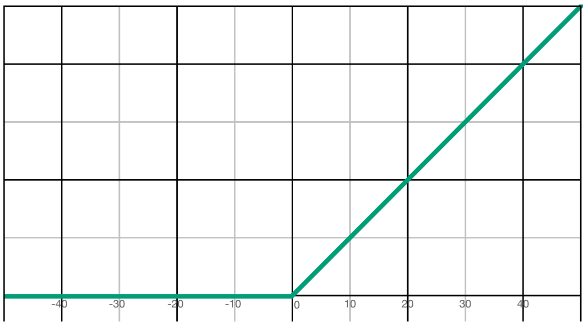


Figure 2.21: ReLU Graph.

2.3.2.2 Pooling Block

The pooling or subsampling increases the receptive field of the CNN by reducing the dimensions of the input. It makes the network robust to small differences in the positions. There are two types of common pooling operation namely: the average pooling and max pooling. Similar to a convolutional block, pooling uses a filter that slides throughout the image pixels. By convention, the pooling filter size is 2×2 that slides with the stride of 2. The stride of 2 reduces the output size to half of the input size. Instead of multiplying the filter parameters with the overlapped area of the image, pooling simply takes the average of the image values in the overlapped area for the average pooling or takes the largest number for the max pooling as shown in Fig. 2.20. Since the filter does not need to learn anything, the pooling block does not have learnable parameters. In modern CNN implementations, max pooling is generally favored over the average pooling because with max pooling, the feature that has the highest contribution is extracted (Krizhevsky et al., 2012).

2.3.2.3 Rectified Linear Unit

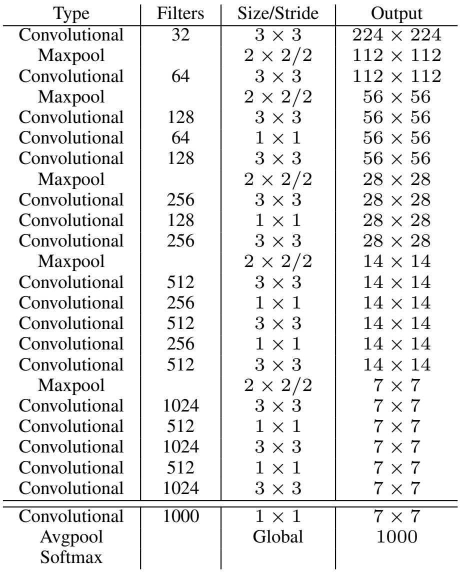
After the convolution operation is performed, the output is passed through an activation function. The conventional activation function used in CNN is the rectified linear unit (ReLU) (Nair and Hinton, 2010). The ReLU graph is illustrated in Fig. 2.21 and the operation is define as $\max(0, x)$, where x is the input. The ReLU function outputs the same value as the input as long as the input value is positive. However, it replaces negative input values with 0. The advantages of ReLU are that it reduces vanishing gradients, increases sparsity in representations (due to the removal of negative values), which helps in dense representations, and reduces computational time (Krizhevsky et al., 2012).

2.3.2.4 Batch Normalization

To speed up the training process, the training set is normalized according to their variances to widen the contours of the learning problem and thereby, ease the optimization for algorithms such as gradient descent. The hidden layers are also normalized to improve the training but the shuffling of datasets every training epoch changes the distribution of the hidden neurons, which causes the internal covariate shift. Consider a network that is trained with binary classification to detect the presence of a dog. For instance, two consecutive batches have different representations of a dog. The first batch contains the faces of a dog and the second batch contains the full body of a dog. If the two batches are fed to the network, the feature space of the hidden layer for every batch will have a considerable shift in the distribution called covariate shift. This slows down the training because the hidden layers need to learn different distributions every time until it converges. To avoid the large changes in distribution, the output in the hidden layers are normalized using *batch normalization* (Ioffe and Szegedy, 2015). Batch normalization is normally applied to the output of the hidden layers before the activation function. It is implemented using the Algorithm 1. In the algorithm, the learnable parameters γ and β are used to change the mean of the distribution and learn the appropriate values that fit the hidden layer distribution. As a result, batch normalization lowers the covariate shift of the hidden layer distribution by fixing the mean and variance. Despite the changes in the distribution as the input changes, the mean and variance stay the same, which stabilizes the output values. It also helps lessen the effect of the changes in the previous

Algorithm 1 Batch Normalizing Transform applied to the output x over a mini-batch. (Ioffe and Szegedy, 2015)

```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
    Parameters to be learned: $\gamma, \beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$    ▷ mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$    ▷ mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$    ▷ normalize
$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$    ▷ scale and shift
```



Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 2.22: YOLOv2 Architecture (Darknet-19) (Redmon and Farhadi, 2017).

hidden layers, and thus, allows faster and stable training.

2.3.3 Applications

2.3.3.1 YOLO Object Detection

The architectures previously discussed are all applied to image classification. However, there are also other significant works in the development of CNN for other applications such as object detection. Object detection locates objects in an image (e.g., cars, bikes) and places bounding boxes and labels on them. One of the important CNNs developed for object detection is called YOLO (you only look once) (Redmon and Farhadi, 2017). YOLO combines the bounding box regression and classification in the same network, which makes it fast and suitable for real-time processing. The YOLOv2, which is

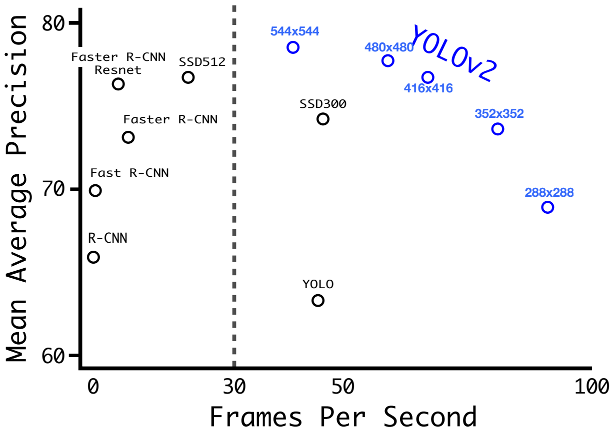


Figure 2.23: YOLOv2 Performance on PASCAL VOC 2007 Dataset (Redmon and Farhadi, 2017).

a notable version of the network, has achieved 76.8% mAP (mean average precision) on PASCAL VOC 2007 dataset at 67 FPS (frames per second). When the FPS is reduced to 40, its mAP increases to 78.6%, which is better than its contemporary networks. The YOLOv2 has an architecture called Darknet-19 for its feature extraction. Darknet-19 has 19 convolutional blocks with 5 max pooling scattered in the middle as illustrated in Fig. 2.22. The improvements of YOLOv2 over its predecessor are the following: First, it uses convolutions with anchor boxes that improve the bounding box predictions. Second, it employs 224 × 224 images for training the feature extraction network on image classification, and then, 448 × 448 images for fine-tuning the network on object detection, all of which makes the training easier and raises the mAP by 4%. Third, it uses batch normalization on all convolutional blocks. Fourth, it utilizes dimension clusters, which take advantage of the bounding box patterns among different problem domains to create better bounding box sizes that fit the detected object. With these improvements, the YOLOv2 outperforms other object detection networks as reported in Fig. 2.23. The example image object detections are shown in Fig. 2.24. However, YOLOv2 has a problem detecting small objects that are close to each other. The YOLOv3 addresses the limitations of YOLOv2 by applying the developments in CNN for image classification. The most distinct change in YOLOv3 is the use of skip connections. Its feature extraction network, Darknet-53, has 53 convolutional blocks that employ skip connections. It is considerably deeper than Darknet-19. Consequently, it solves the problem with small object detection and once more improves the performance. The YOLOv3 architecture is shown in Fig. 2.25 and its performance comparison to other object detection networks is

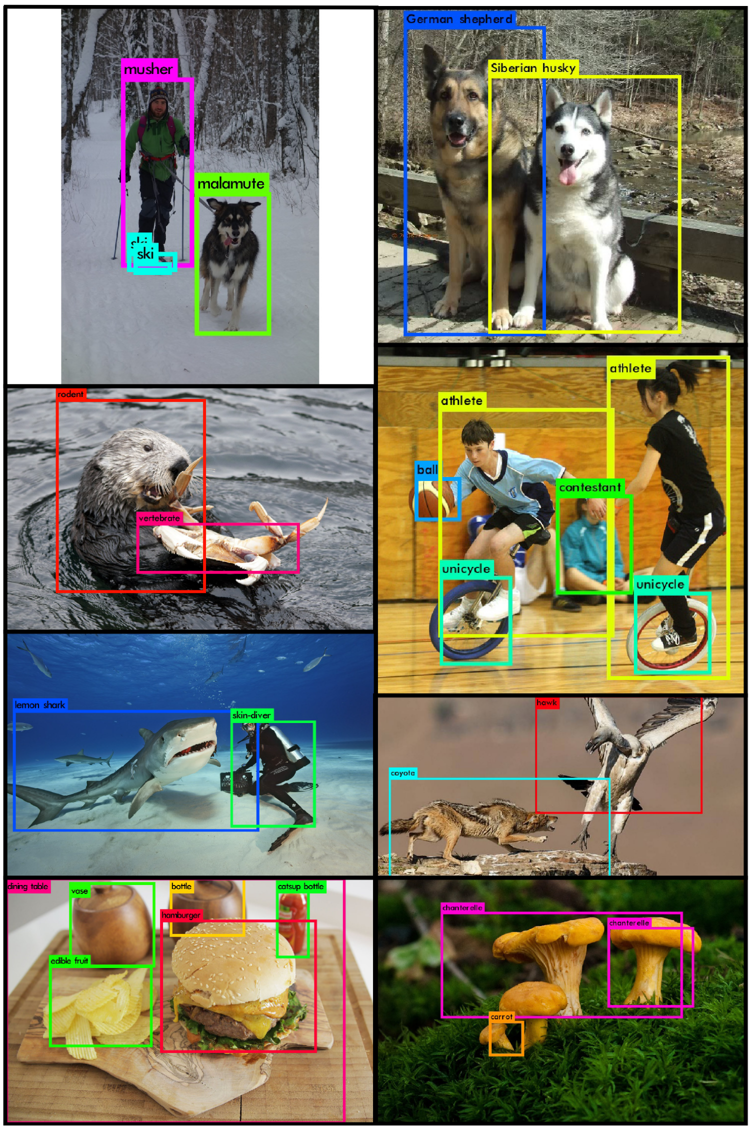
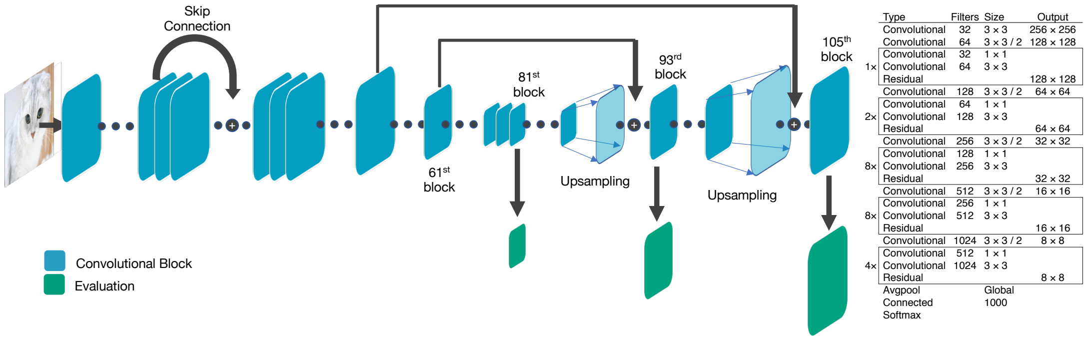


Figure 2.24: Examples of YOLOv2 Object Detection (Redmon and Farhadi, 2017).

shown in Fig. 2.26.

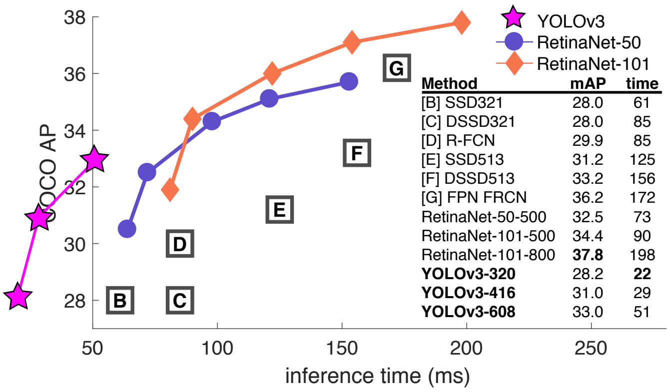
2.3.3.2 Novel Applications of CNN

In addition to object detection, CNN can also be applied to novel applications. One of the novel applications of CNN is to create a painting. The ability of the weights of CNN to characterize an image can be exploited to extract the key attributes of an image. Subsequently, the weights can be employed to *reimagine* an image into the so-called "style" of a previously seen image. This technique is called neural style transfer and an example is exhibited in Fig. 2.27 (Gatys et al., 2016). Moreover, the CNN can be configured



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.25: YOLOv3 Architecture (Darknet-53) (Redmon and Farhadi, 2018).



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 2.26: YOLOv3 Performance on COCO dataset (Redmon and Farhadi, 2018).

to have two networks competing to outperform each other. This is known as generative adversarial networks (GANs), in which one network tries to classify images correctly called the discriminator network, and another network tries to fool the discriminator network by creating fake image examples called the generator network (Goodfellow et al., 2020). In one of its study using celebrity faces as the dataset, the generator network becomes exceptionally well at fooling the discriminator network that it can be tweaked to produce faces that do not exist but looks genuine (Karras et al., 2017). The realistic faces are shown in Fig. 2.28. Another study also used GANs to transfer the facial movements from a source video to a target video, which is exhibited in Fig 2.29 (Kim et al., 2018). Aside from the facial movements, body postures can also be transferred between the source video and the target video. In a different study, the dance moves from a source video are mimicked by a target video (Chan et al., 2019). The sample images can be seen in Fig. 2.30. Finally, in addition to image recognition, CNN can also be applied in action recognition (i.e., video recognition). One of the implementations of action recognition is by employing two CNNs that extract different information from a video. The evaluation

Figure 2.27: Neural Style Transfer Examples (Gatys et al., 2016).

of each network is combined to produce a prediction. In the work of Simonyan and Zisserman (2014a), they used the first network to classify still frames (spatial recognition stream) and the second network to classify the motion between the frames using optical flow (temporal recognition stream). The network architecture is shown in Fig. 2.31.

2.3.4 Adversarial Attacks

Despite the immense success of the CNN, it has been discovered that it has a vital weak point in its learning. A subtly and deliberately perturbed image, which looks normal to the human eyes, can cause the network to catastrophically mislabel it with high confidence. The image depicted in Fig. 2.32 shows that when the panda image on the left is added with the calculated image perturbation in the middle, the resultant image or *adversarial example* can fool the network with high probability. Although the

Figure 2.28: GAN produced faces (top) and closest real face images (bottom) (Karras et al., 2017).

image in the middle looks similar to a noise image, it is actually composed of carefully calculated values designed to maximize the error of the network. Therefore, it causes the network to mislabel the adversarial example with high confidence (Goodfellow et al., 2015). The susceptibility of CNN to adversarial examples indicates that despite the CNN exceeding human-level performance in image classification, it is not using the same method as humans to evaluate images.

One of the methods to produce adversarial examples (i.e., images that fool the network) is using the fast gradient sign method (FGSM). In FGSM, the gradients of the loss with respect to the input are used to slightly perturb the image to the direction of the gradients, which effectively increases the error (Goodfellow et al., 2015). It is a simple adversarial attack method that can generate adversarial examples quickly. Consequently, there are techniques developed that can resist this kind of attack (e.g., image filtering techniques) (Das et al., 2017). However, this adversarial attack can be reinforced by repeating the perturbation on the input image, which is called projected gradient descent (PGD) (Madry et al., 2018). Thus, PGD creates strong adversarial examples where many adversarial defenses struggle to fully defend the network. A peculiar and critical characteristic of the adversarial examples is that these can be used to fool other CNNs without any additional perturbations, also known as transferability. Transferability of adversarial examples poses several potential concerns to systems using CNN (e.g., pedestrian detection) because direct access to these systems is not necessary to compromise

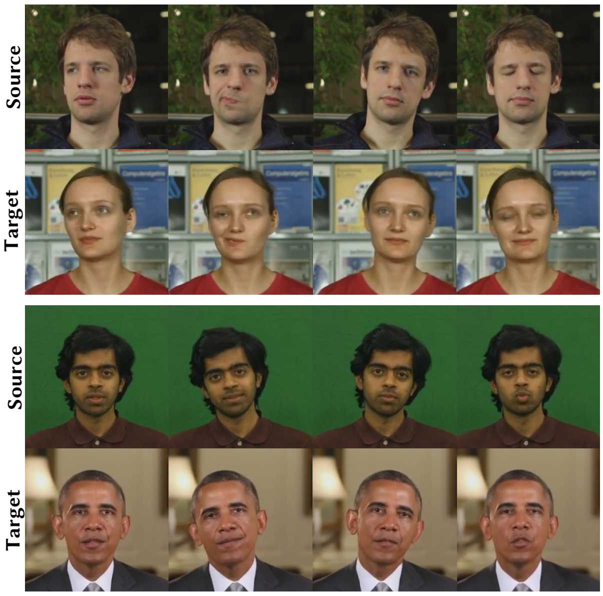


Figure 2.29: Facial expressions and head movements transfer (Kim et al., 2018).

them. Several studies are done to understand and try to prevent transferability (Szegedy et al., 2013; Biggio et al., 2013; Goodfellow et al., 2015; Papernot et al., 2016; Liu et al., 2016; Papernot et al., 2017; Moosavi-Dezfooli et al., 2017; Tramèr et al., 2017; Dong et al., 2018; Demontis et al., 2019). The characteristics of networks with high transferability between them are the following: First, networks tend to converge into similar functions when trained with the same dataset despite their differences in the architecture, parameters, hyperparameters settings, etc (Goodfellow et al., 2015; Moosavi-Dezfooli et al., 2017; Tramèr et al., 2017). Second, complex networks are more susceptible to transferability than simple networks and the skip connections can be employed to increase transferability (Demontis et al., 2019; Wu et al., 2019). Third, the networks with high transferability of adversarial examples exhibit aligned gradients (Liu et al., 2016; Demontis et al., 2019). Lastly, the extent of transferability is correlated to the similarity of network architectures. Networks with high architectural similarity tend to have high transferability (Papernot et al., 2016; Tramèr et al., 2018).

There are many proposed solutions to address the problem of adversarial attacks. From the input perspective, the input images are filtered to remove the effects of the perturbation caused by the adversarial attacks. This line of techniques includes, JPEG compression (Das et al., 2017), bit squeezing (Xu et al., 2017), bilateral filtering (Xie et al., 2019), etc. Another proposed solution is to include adversarial examples in the training

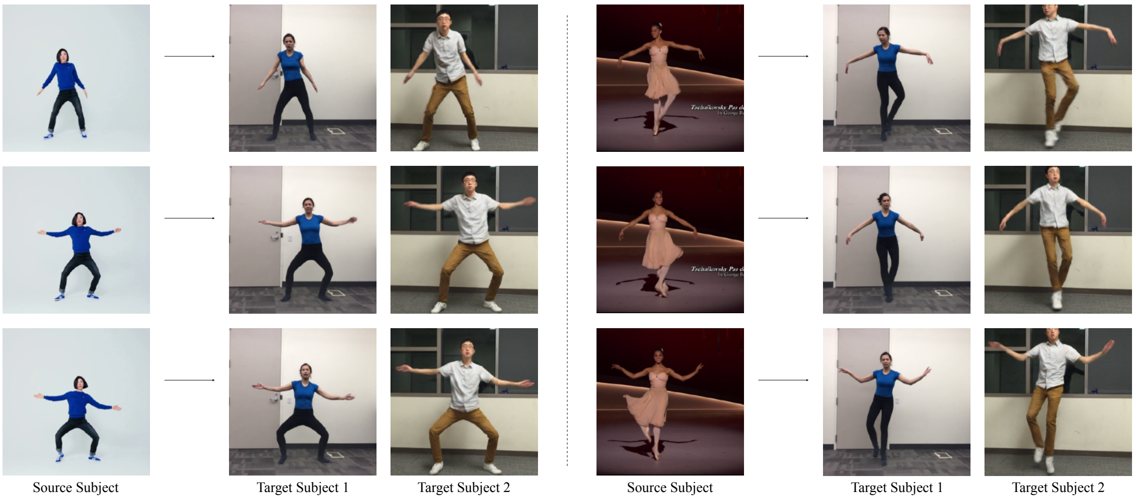


Figure 2.30: Body movements transfer (Chan et al., 2019).

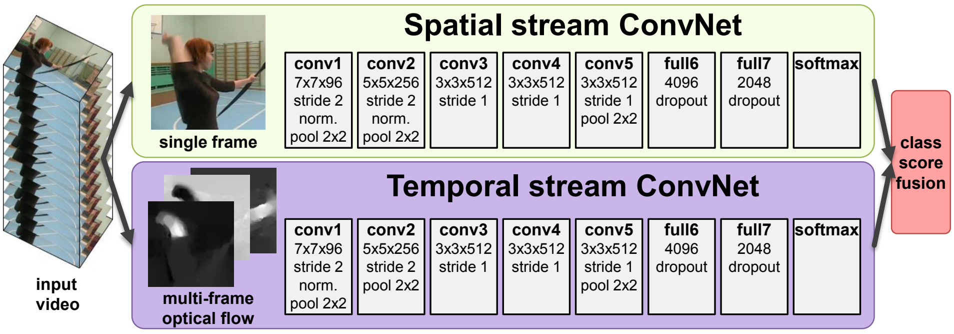


Figure 2.31: Two-stream Architecture for Video Classification (Simonyan and Zisserman, 2014a).

set of the network, known as adversarial training (Goodfellow et al., 2015). The notable adversarial training methods are free adversarial training (Shafahi et al., 2019) and fast adversarial training (Wong et al., 2019). Adversarial training works well in making the network robust to adversarial examples but the additional training cost (i.e., adversarial examples generation and additional training images) is the main drawback. Furthermore, there are various proposed solutions to tackle specifically the transferability of adversarial examples. Since the aligned gradients of networks indicate high transferability, some studies suggest reversing the direction of the gradients (Kariyappa and Qureshi, 2019; Jalwana et al., 2020). Moreover, evolving an architecture that is robust to transferability is also recently introduced by a handful of studies. Devaguptapu et al. (2020) proposed to use differentiable architecture search (DARTS) (Liu et al., 2018a) to find architectures that are robust to transferability. Kotyan and Vargas (2020) used neuroevolution to search for robust networks. To make the search effective and less exhaustive, several

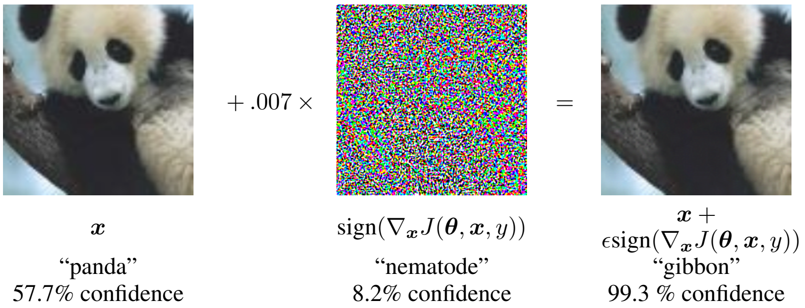


Figure 2.32: Adversarial Attack Example. A network that classifies the panda image as a panda can be fooled by adding the calculated image perturbation in the middle to the panda image. The output image still resembles a panda to the human eyes but causes the network to mislabel it as a gibbon with high confidence (Goodfellow et al., 2015).

works incorporated adversarial training in the architecture search. Liu and Jin (2021) utilized DARTS with adversarial training, in which they found more robust networks than without adversarial training. Other studies employed oneshot neural architecture search (Cai et al., 2019) with adversarial training to find a family of robust networks (Guo et al., 2020; Xie et al., 2021).

2.4 Datasets and Data Augmentations

Datasets are important components of the deep neural network. As deep neural network continues to grow in depth, the size of the datasets also increases to effectively train the network parameters or weights. With sufficient dataset images, a deep neural network can highly perform tasks such as image recognition, object detection. There are several conventional datasets used to benchmark the performance of a network (e.g., CIFAR-10, ImageNet). These kinds of datasets have large sizes because they can be easily collected. For example, ImageNet is a compilation of normal images that can be taken from everyday life situations (e.g., a walking dog) (Russakovsky et al., 2015). Conversely, there are datasets that are obtained from specialized applications and thus have limited datasets. For example, X-ray of cancer tumor has limited datasets because it requires a specific machine to collect data and positive examples are relatively scarce. To extend a limited dataset, a technique called *data augmentation* is employed. An example of data augmentation is horizontally flipping an image, which creates a mirrored version of the image as a new example. This technique can also be applied to conventional datasets to further improve the performance of a network. Krizhevsky et al. (2012) employed horizontal flip in their work on AlexNet, which shattered the image classification records

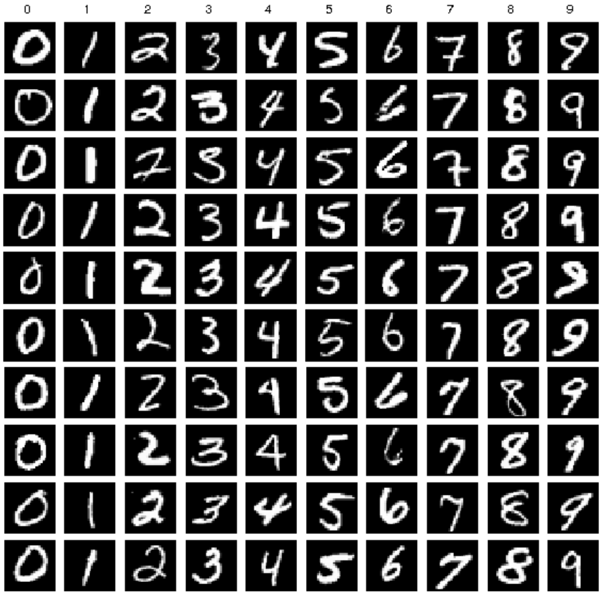


Figure 2.33: MNIST Dataset Examples (Lim et al., 2016).

in the ImageNet competition at the time.

2.4.1 Common Datasets and Specialized Dataset

The common datasets utilized in convolutional neural network (CNN) are introduced here. The datasets are MNIST, CIFAR-10, FMNIST, and KMNIST, which are used to train a network for image classification. In addition, a specialized dataset on dangerous objects X-ray is also introduced, which is used for object detection.

2.4.1.1 MNIST Dataset

The MNIST dataset consists of handwritten single-digit images as illustrated in Fig. 2.33. It has 10 classes corresponding to each 0-9 digits. Each image is black and white and the size is 28×28 pixels. The training set has 60,000 images and the test set has 10,000 images (LeCun et al., 1998).

2.4.1.2 CIFAR-10 Dataset

The CIFAR-10 dataset is comprised of natural images as shown in Fig. 2.34. The dataset has 10 classes, which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each image is colored with the size of 32×32 pixels. The training set

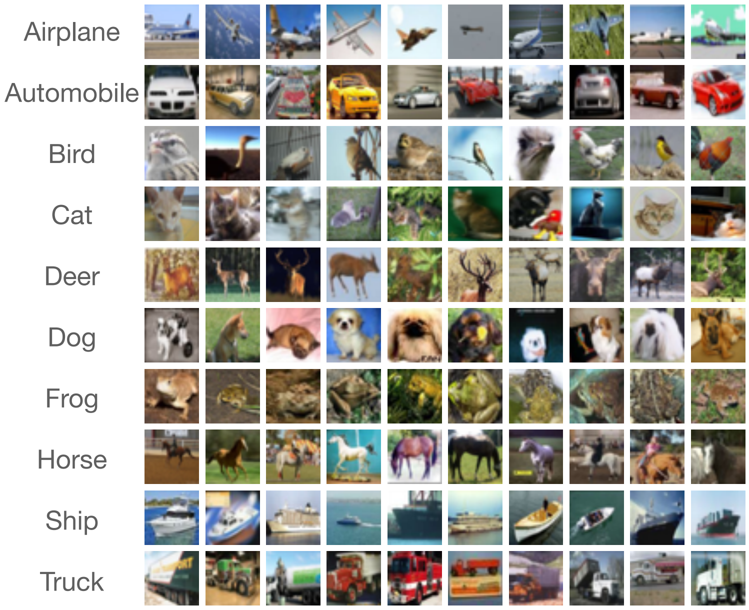


Figure 2.34: CIFAR-10 Dataset Examples (Krizhevsky et al., 2009).

has 50,000 images, whereas the test set has 10,000 images (Krizhevsky et al., 2009).

2.4.1.3 FMNIST Dataset

The Fashion-MNIST (FMNIST) dataset is created to become an alternative to the MNIST dataset, which has become a simple dataset for deep neural network. Most deep networks can easily achieve 99% accuracy on the MNIST test set and FMNIST tries to provide a harder dataset. The example images are shown in Fig. 2.35. Similar to MNIST, it has 10 classes namely: t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. Each image is grayscale with the size of 28 × 28 pixels. The dataset has 60,000 training images and 10,000 test images (Xiao et al., 2017).

2.4.1.4 KMNIST Dataset

The Kuzushiji-MNIST (KMNIST) contains old Japanese character writing as depicted in Fig. 2.36. The dataset contains 10 classes with each class representing a hiragana character as seen on the leftmost column in Fig. 2.36. Each image has black and white colors with the size of 28 × 28 pixels. This dataset also has 60,000 training images and 10,000 test images (Clanuwat et al., 2018).

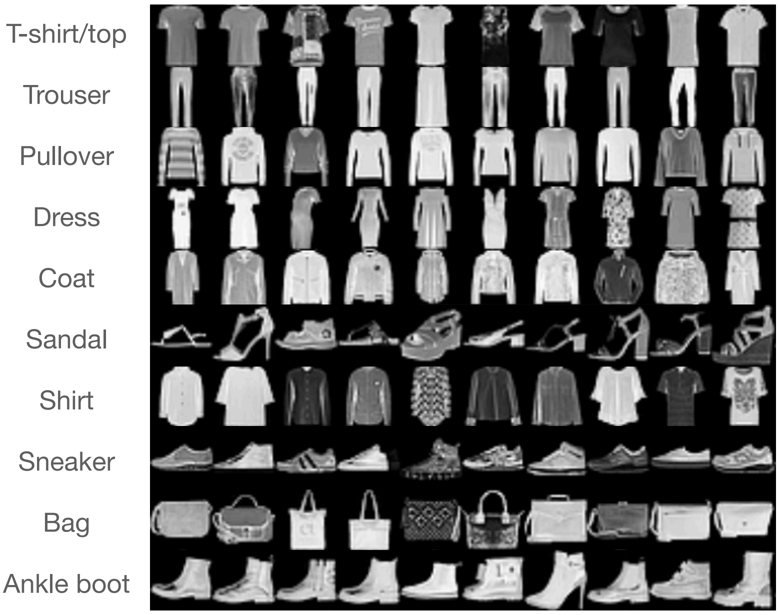


Figure 2.35: FMNIST Dataset Examples (Xiao et al., 2017).

2.4.1.5 Dangerous Objects X-ray Dataset

The dangerous objects X-ray dataset contains the X-ray of banned objects in airports. The dataset classes are scissors, knives, and bottles. Every image is grayscale with an average size of 600 × 600 pixels on the length and width. As seen in Fig. 2.37, an image may contain an object, two overlapping objects, or multiple overlapping objects. Since this dataset is for object detection, the bounding box and class information of each object in an image are recorded in the labels. The dataset has 662 training set images and 442 test set images. Furthermore, to increase the limited dataset size, there are also synthesized dangerous objects X-ray images, which are shown in Fig. 2.38. The synthesized images are created by applying traditional image processing techniques on the raw X-ray images and combining the objects extracted. The synthesized images provide an additional 3010 images to the training set (Zou et al., 2018).

2.4.2 Data Augmentations

In 2012, AlexNet incited the resurgence of interest in CNNs when it accomplished new records in image classification whilst beating traditional image processing techniques. It credited data preparation as one of the reasons for its success. In data preparation, data augmentations are applied to the training images (e.g., mirroring, cropping, etc.)

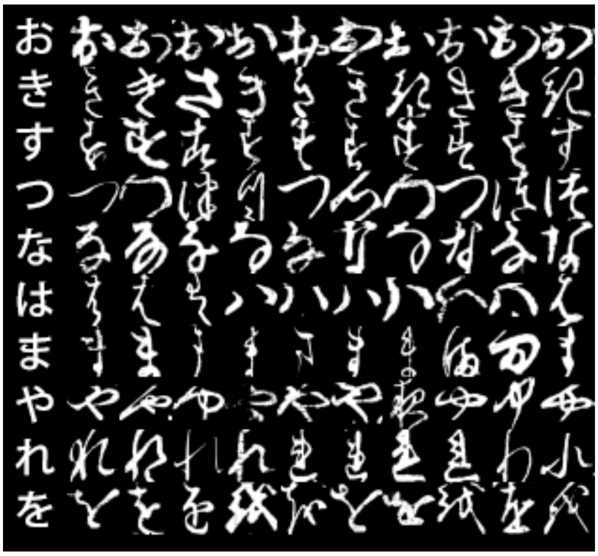
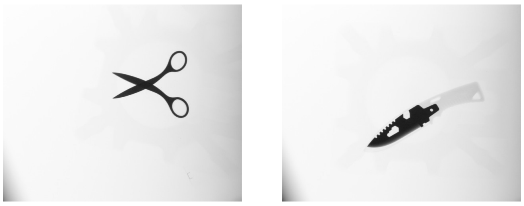


Figure 2.36: KMNIST Dataset Examples (Clanuwat et al., 2018).

to make the network invariant to image perturbations. By applying simple geometric transformations to the images, the small changes in the images can significantly extend the training dataset, and thereby, increase network accuracy. A deep network can overfit a dataset due to the sheer number of its weights but data augmentations can reduce overfitting by altering the input image in every epoch (Krizhevsky et al., 2012). Therefore, the network is forced to learn only the important features of a dataset. Furthermore, data augmentations are invaluable to specialized and limited datasets such as dangerous objects X-ray, which is discussed in Section 2.4.1.5. Through data augmentations, more information can be extracted from a small dataset.

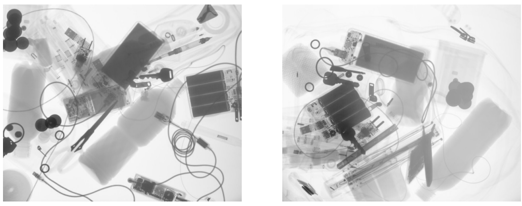
2.4.2.1 Training Set Data Augmentations

Typically, data augmentations are applied to the training images during the network training process. It creates myriads of image variations through simple image perturbations (e.g., flipping, rotation, etc.) to extract more information from the dataset (Esteva et al., 2017). In addition, the majority of data augmentations are developed for the image classification (Shorten and Khoshgoftaar, 2019). The data augmentations that modify the shape of the image are called *geometric transformation.* The examples of geometric transformation, as shown in Fig. 2.39, are horizontal flip, rotation, cropping, scaling, etc. In contrast, there are data augmentations that adjust the RGB channels of an image to highlight or accentuate a particular image feature. These are called *color space trans-*



a) Single object

b) Two overlapping objects



c) Multiple overlapping objects

Figure 2.37: Raw Dangerous Objects X-ray Image Examples (Zou et al., 2018).

formation or *photometric augmentation*. The examples of these augmentations are white balance, histogram equalization, etc. as exhibited in Fig. 2.40 (Wu et al., 2015; Mikołajczyk and Grochowski, 2018). Finally, there are data augmentations that introduce noise to the image to make the network robust. Examples of these data augmentations are random erasing and cutout regularization, which patch an image with a random block— usually black— to remove information from the image (DeVries and Taylor, 2017; Zhong et al., 2020). Through the image occlusions, these augmentations help the network to focus also on other parts of an image before deciding on a label. These augmentations are shown in Fig. 2.41.

Recent advances in data augmentation focus on combining two or more images to create a new image. In one study, two images are crudely combined by taking the per-pixel average of two images. Unexpectedly, this has resulted in an accuracy increase in CIFAR-10 benchmark, although the scientific basis is still ongoing research (Inoue, 2018). A more intuitive approach is to combine images using alpha blending. In alpha blending, an image is turned translucent to allow it to be overlaid on top of another image. In this

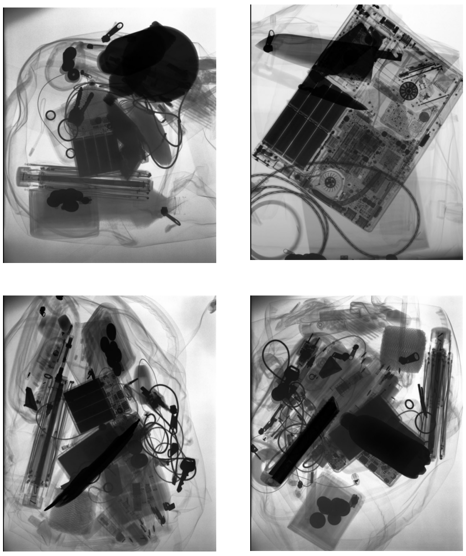


Figure 2.38: Synthesized Dangerous Objects X-ray Image Examples (Zou et al., 2018).

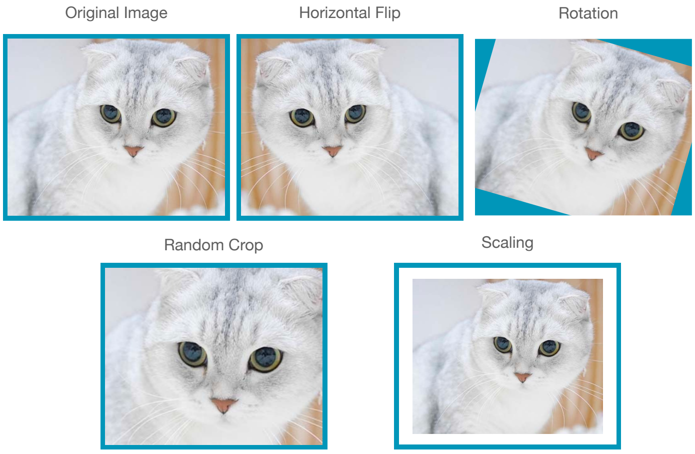


Figure 2.39: Geometric Transformation Examples.

way, the images are still visually understandable. This technique has also increased the network performance (Zhang et al., 2017). Various methods to mix two images are further explored in the work of Summers and Dinneen (2019), which is shown in Fig. 2.42. One of their best image integration was able to reduce the CIFAR-10 error by 1.6%. Interestingly, Takahashi et al. (2020) have combined 4 different images by cropping each image with a specific size such that the 4 images can be arranged to form an image. Consequently, the label for the produced image is a soft label with class values depending on the ratio of each image in the integrated image. The soft label forces the network to concentrate on 4 different areas of an image on every image example. As a result, the network focuses on vital portions of an image and remarkably increases performance. The example image

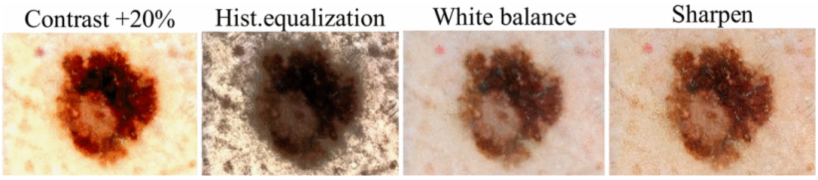


Figure 2.40: Color Space Transformation Examples (Mikołajczyk and Grochowski, 2018).

Figure 2.41: Random Erasing Examples.

is shown in Fig. 2.43.

2.4.2.2 Test-time Augmentations

Although not very common, data augmentations can also be applied to the test set images before network evaluation. This procedure is called test-time augmentation. Test-time augmentations allow a more robust network prediction by showing multiple viewpoints of an image rather than only an unperturbed image. In statistics, test-time augmentations are compared to ensemble learning, in which multiple models create a consensus on the final prediction. In contrast, deep neural network uses an image that is augmented to form an ensemble of viewpoints. The evaluation of the network on all the augmented images is averaged to obtain the final network prediction. Radosavovic et al. (2018) implemented this approach to label unlabeled images from the internet and use these pseudo-labels to train a semi-supervised network. In addition, test-time augmentations are used to apply the 10-crop testing procedure on AlexNet and ResNet (Krizhevsky et al., 2012; He et al., 2016). The 10-crop testing procedure augments a test image by taking an image crop on each image corner, center, and also the equivalent on the image horizontal flip. The average of the predictions on all the 10 augmented images is used as the final prediction. Moreover, test-time augmentations can also measure the robustness of a network, in which the drop in accuracy of the network indicates its sensitivity to image perturbations (Minh et al., 2018). Furthermore, test-time augmentations using geometric transformations are found to be effective in classifying skin lesion (Matsunaga

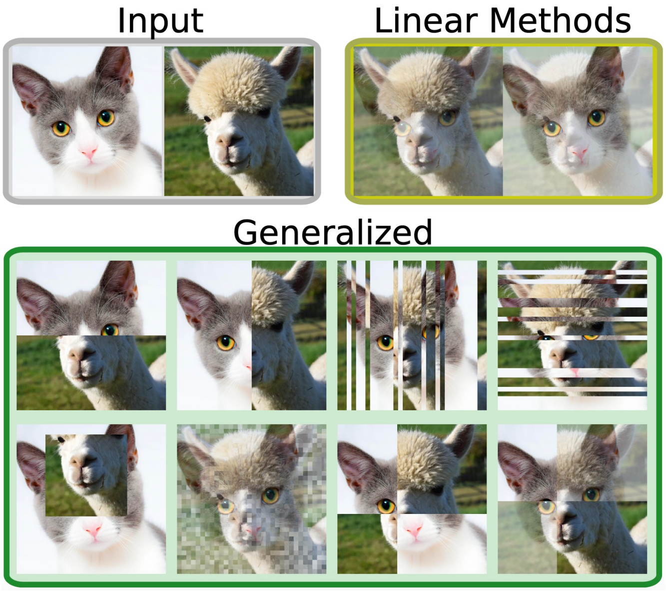


Figure 2.42: Image Mixing Examples. Linear methods shows alpha blending. Generalized shows different image mixing methods (Summers and Dinneen, 2019).

et al., 2017), and these are more effective when combined with data augmentation in the training phase (Perez et al., 2018).

2.5 Learning from trained Networks

2.5.1 Transfer Learning

Transfer learning is a method in deep neural network in which the learning of a network on a particular task is repurposed for another task. The motivation of transfer learning is attributed to the hierarchical nature of representations in CNN. From the first layer to the last layer of a network, the features extracted grow in complexity. Particularly, the initial layers act as low-level feature extractors such as edge detectors, whereas the final layers detect complex features such as faces (Zeiler and Fergus, 2014). This learning behavior is common to CNNs, which allows the reuse of the trained weights to another application as initialization. As a result, the network has faster convergence and higher performance improvement, especially when the new dataset is similar to the previous dataset the network is trained on. The transfer learning can exhibit three potential benefits, which are a higher start, higher slope, and higher asymptote as shown in Fig. 2.44 (Torrey and Shavlik, 2010). A higher start implies a higher starting accuracy, a higher slope implies a steeper rate of network improvement, and a higher asymptote

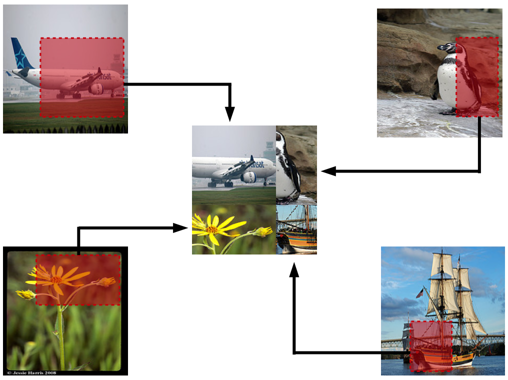


Figure 2.43: Four images are combined with varying ratio sizes to form an image (Takahashi et al., 2020).

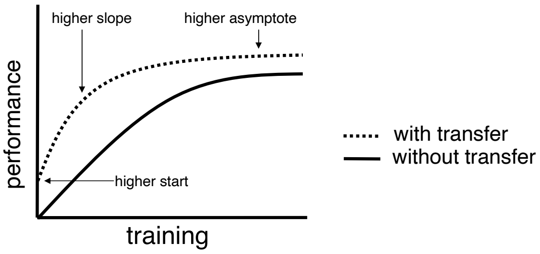


Figure 2.44: Potential benefits of transfer learning (Torrey and Shavlik, 2010).

implies a higher convergence value.

There are two methods to apply transfer learning to a network. The first method is *feature extraction*, in which the trained network is utilized as a fixed feature extractor. In feature extraction, the weights of a trained network are frozen and only the last layer is used to train the network on a new dataset. This method is particularly effective when the previous dataset and the new dataset share features (e.g., wolf dataset and dog dataset) because only the last layer, which differentiates the datasets, is needed to be retrained. In addition, this technique is also useful for small datasets (e.g., medical X-rays). Using the learning of a network on large-scale and challenging datasets can help small datasets compensate for the lack of numbers in training a deep network. The second method is *fine-tuning*. Fine-tuning is similar to feature extraction but the main difference is the retraining of trained network weights. Depending on the size and similarity of the new dataset to the previous dataset, the network layers to be retrained can be adjusted. If the new dataset is small, the first few layers, which extract generic features, can be fixed to avoid overfitting. However, when the new dataset is large, the whole network can be retrained,

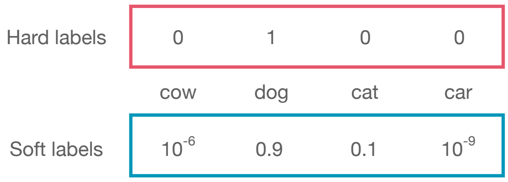


Figure 2.45: Difference between the hard and soft labels in knowledge distillation (Liu et al., 2018b).

in which the trained weights are utilized as initialization. Compared to initializing the network with random weights, the trained weights can accelerate the network convergence and improve performance. This is attributed to the minimal adjustments needed in the network weights since these are already tuned as opposed to random weights.

Transfer learning is a powerful technique to facilitate the training of a deep network. The similarity of datasets in CNN makes it a standard procedure to train networks. Although the efficiency of transfer learning decreases as the new task becomes more and more different than the initial task, it is still beneficial to initialize the network with trained weights rather than random weights. Finally, in some tasks, fine-tuning a network does not hamper the performance of the network on its initial task (Yosinski et al., 2014; Sharif Razavian et al., 2014).

2.5.2 Knowledge Distillation

Very deep networks have achieved remarkable success in various fields such as computer vision. However, the complexity that accompanies very deep networks has hindered the deployment of these networks to real-life applications, particularly on small devices such as surveillance cameras. To address this kind of issue, a model compression method is proposed to transfer the information from a large model to a small model without substantial accuracy drop by training the small model to imitate the behavior of the large model (Buciluǎ et al., 2006). The idea of using a large model to teach a small model is later known as *knowledge distillation* in the deep neural network literature (Hinton et al., 2015). The concept of knowledge distillation is to train a student network to imitate how the teacher network evaluates an input to gain performance comparable to the teacher network. The supervision signal from a teacher network, which is referred to as *knowledge*, guides the student network in imitating the teacher network. For example, in an image classification task, the logits (i.e., final layer output) of a teacher network contain

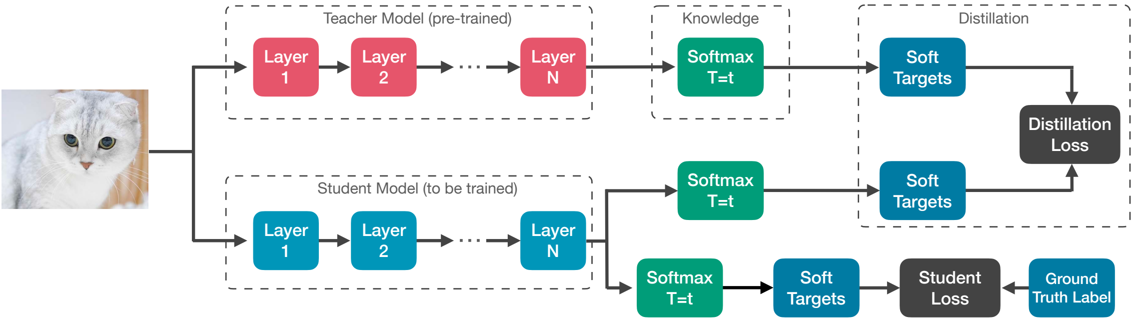


Figure 2.46: Vanilla Knowledge Distillation Process (Gou et al., 2021).

knowledge that is not found in the training dataset. In a dataset set, a dog image has a one-hot label of one for the dog class and zero for the other classes. However, in the teacher evaluation, the dog class may have a high probability and low probability for the cat class but the probability for the car class is many times lower than the cat. This difference in labeling is shown in Fig. 2.45. In another example, the teacher evaluation on handwritten digit 6 may be closer to digit 8 than digit 1. This classes relationship learned by the teacher is known as the *dark knowledge*.

A typical knowledge distillation set-up consists of a large and trained teacher network and a small network. To share the knowledge learned by the teacher network to the student network, during the student network training, an input image is fed to both the teacher and student networks. Their respective logits are compared and the calculated difference becomes the distillation loss. Furthermore, the student still calculates the cross-entropy loss on its evaluation and the ground truth label, which is now referred to as the student loss. The new loss of the student network is the sum of the student loss and distillation loss. The summary of the training process is shown in Fig. 2.46. Adding the distillation loss to the typical cross-entropy loss encourages the student to not only learn the correct labels of the images but also learn how the images classes relate to other classes. As a result, the student network can perform better with teacher network knowledge than simply relying on the dataset labels.

2.6 Evolutionary Computation

2.6.1 Genetic Algorithm

Genetic algorithm is a metaheuristic approach to search for optimized solutions to problems, inspired by natural selection. It uses a population of potential solutions that undergo a series of selection, crossover, and mutation operators for many generations to search for the fittest individual or optimal solution (Mitchell, 1998). A simple genetic algorithm process can be summarized by the following steps:

Step 1. Randomly generate a population of potential solutions (to a problem)

Step 2. Calculate the fitness of each individual in the population

Step 3. Select the individuals based on their fitness

Step 4. Apply the genetic operators to the selected individuals to produce children

Step 5. Replace the individuals in the population with the children

Step 6. Repeat the process from Step 2 for the next generation until the final generation count is reached

In Step 1, the individuals in a population are generated using GTYPE (genotype or chromosomes), which carries the genetic codes of a solution, and PTYPE (phenotype), which is the physical manifestation of the genotype (Iba et al., 2009). For example, the GTYPE can represent the number of nodes and layers in a neural network and PTYPE can represent the actual neural network with its connections and weights. In Step 2, fitness is a measure of the capability of an individual to address the main problem (e.g., accuracy). The fitness of each individual is computed to recognize individuals that can potentially bring the population closer to the optimal solution. In Step 3, based on the fitness, the individuals are selected to undergo genetic operators that will form the new population. There are many ways to perform selection but the common techniques are roulette-wheel selection and tournament selection. The roulette-wheel selection, also known as the fitness-proportionate selection, is the simplest selection method. It creates a roulette where the probability of each individual to be chosen is proportional to its

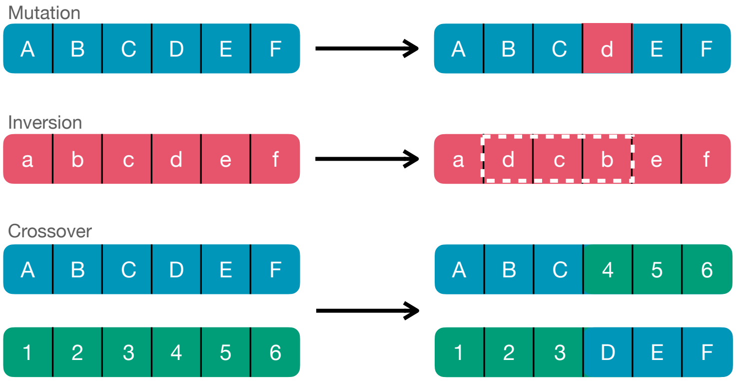


Figure 2.47: Genetic Operators (Iba et al., 2009).

fitness. The probability of each individual being selected is computed as

$$p_i = \frac{f_i}{\sum f}, \tag{2.17}$$

where f_i is the fitness of the i-th individual. The roulette-wheel is "spun" and the individual, which is represented by the area where the "ball" lands, is selected. On the other hand, tournament selection holds a tourney for S_t number of randomly chosen individuals that will compete using their fitnesses. The S_t is the size of each tourney and the number of tourneys is equivalent to the number of individuals. The "winner" of each tourney becomes the selected individual. Other selection methods include the elite strategy, in which the best individual in the current population is carried over to the next generation without mutations. The concept behind this is to guarantee that the next population does not incur performance degradation due to mutations. Another selection method is the steady-state selection where only a certain number of low-performing individuals are replaced by the children.

In Step 4, each selected individual is applied with a genetic operator, which can be mutation, inversion, or crossover. The genetic operators are inspired by biological reproduction. Mutation and inversion have originated from asexual reproduction, whereas crossover has originated from sexual reproduction (Fontanari and Meir, 1990; Collins and Jefferson, 1992). Although the implementation of genetic operators depends on the properties of individuals, the concept of genetic operators can simply be illustrated on a GTYPE. For instance, the GTYPE of every individual in a population is represented by a six-character string. As shown in Fig. 2.47, mutation changes one of the characters into another kind, inversion swaps two characters inside the string, and crossover uses two individuals which exchange or “cross” a certain subset of the string (Iba et al., 2009). After each selected individual is evolved, they replace the current individuals in the population for the next generation as in Step 5. The process from Step 2 repeats until the final generation count is reached or the termination criterion is met.

2.6.2 Neuroevolution

Neuroevolution is the evolution of artificial neural network using genetic algorithm. In the 1980s, backpropagation was a popular algorithm to train artificial neural network. However, despite the success of backpropagation, it is vulnerable to getting trapped in a local minimum of an error function. Moreover, it cannot find the global minimum of a multimodal and/or nondifferentiable error functions (Hertz et al., 2018). To address this issue, a small group of researchers adopts the genetic algorithm to optimize the connection weights. This is called fixed-topology neuroevolution, where the architecture is preconceived and only the connection weights are modified. Evolving the connection weights as a method of network training has two phases. First is the representation of the connection weights, which can be binary representation (Holland et al., 1992; Goldberg and Holland, 1988) or real-number representation (Montana et al., 1989; Bartlett and Downs, 1990). Second is the evolutionary process in which the genetic operators or search operators (e.g., mutation, crossover) are used in conjunction with the representation scheme (Yao, 1999). Evolutionary training is an attractive approach because it can work better on complex, multimodal, and nondifferentiable surfaces. Furthermore, it can be applied directly to different kinds of ANN (e.g., recurrent ANN), eliminating the need to develop a training method. In some studies, evolutionary training is even combined with backpropagation to generate better results. The genetic algorithm is utilized to locate good regions in space (near-optimal initial connection weights) and then backpropagation performs the local search by fine-tuning the initial weights (Belew et al., 1991; Lee, 1996; Topchy and Lebedko, 1997).

Although the connection weights can be trained well using neuroevolution, the capability of an ANN to process information depends on its architecture or topology. Most of the topology designing process relies on human expertise and experience. However, the topology can be designed using neuroevolution. Automatic topology design has two types. The first type is the constructive algorithm, where it starts with the smallest

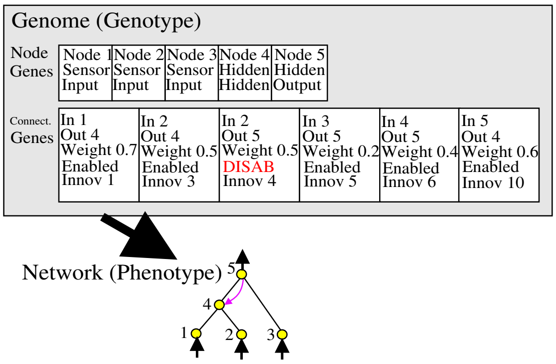


Figure 2.48: NEAT Genetic Encoding (Stanley and Miikkulainen, 2002).

network configuration (e.g., minimum layer, minimum nodes) and gradually makes it complex. On the other hand, the second type is the destructive algorithm where it does the opposite. It starts with the most complex networks and gradually removes the unnecessary components. Similar to the connection weights evolution, architecture evolution has two phases. The first phase is architecture genotype representation. When all the details of the architecture (number of hidden layers, connection, etc.) are encoded, it is called direct encoding (Whitley et al., 1990; Miller et al., 1989). However, if some details are omitted (e.g., connection details) and left for the training process to decide, it is called indirect encoding (Kitano, 1990; Harp et al., 1989). The second phase is the genetic operators used to evolve the architecture. Evolving the architectures has shown to be effective in applications such as supervised training (Chen et al., 1993) but it has difficulties in evaluating the fitness correctly due to the absence of weights information and subsequently makes the evolution inefficient (Angeline et al., 1994). As a result, there are several works published on evolving the topology and weights simultaneously (Koza and Rice, 1991; Angeline et al., 1994; Yao and Liu, 1998, 1997).

The advantage of evolving both the topology and weights over a fixed topology and evolving weights is not yet firmly established. Gruau et al. (1996) have argued that evolving the topology saves time and effort on applying trial-and-error on topology design in fixed-topology neuroevolution. They supported this argument by solving the hardest pole-balancing benchmark at the time using their proposed method called cellular encoding. However, the same benchmark was later solved by a fixed-topology method called enforced subpopulations that is 5 times faster. Therefore, the argument for evolving architecture has been nullified (Gomez et al., 1999).

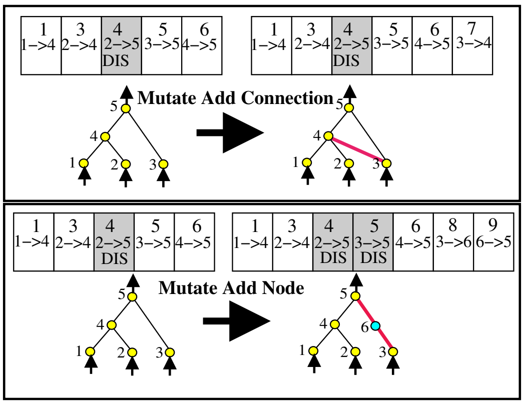


Figure 2.49: Structural Mutations in NEAT (Stanley and Miikkulainen, 2002).

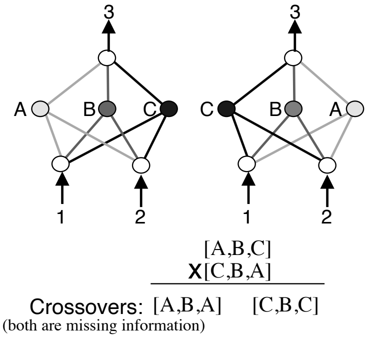


Figure 2.50: Crossover problem in topology evolving algorithms. It is problematic due to the missing information when two network parts are recombined (Stanley and Miikkulainen, 2002).

A seminal work that established the importance of topology and weights evolution is proposed by Stanley and Miikkulainen (2002) called neuroevolution of augmenting topologies (NEAT). The authors asserted that if the evolution of the structure along with the connection weights is properly designed, then it can significantly enhance the performance of the network. In one of the NEAT performance evaluations, it has outperformed the leading fixed-topology neuroevolution in the standard double pole balancing without velocity task, which due to its difficulty, only two algorithms were able to solve it at the time. Specifically, NEAT is 25 times faster than cellular encoding and 5 times faster than enforced subpopulations. NEAT has three crucial aspects that are responsible for its success. First is tracking the genes of a network through *historical marking.* Every network individual in NEAT is encoded using direct encoding as shown in Fig. 2.48 and recorded in the historical marking. The mutations applied on the networks are *add node* and *add connection* as shown in Fig. 2.49. Historical marking also allows better crossover of two networks, which is previously omitted in topology evolving algorithms due to the detri-

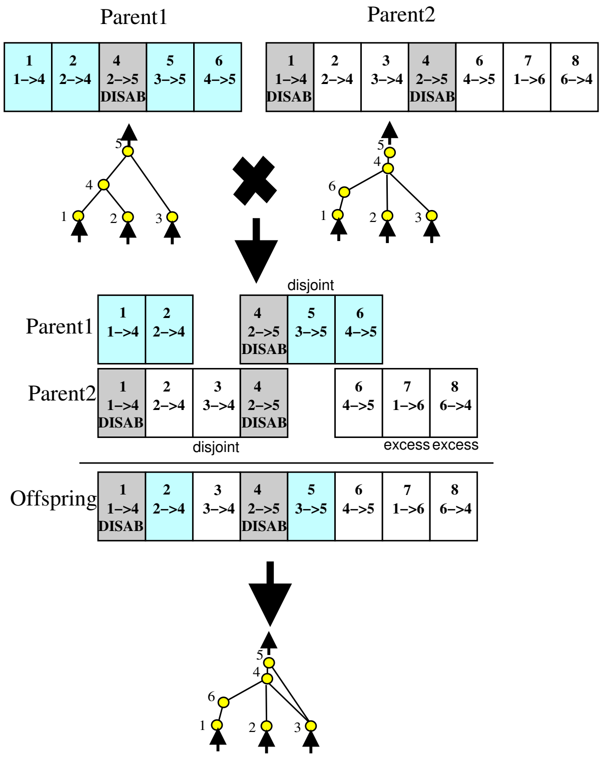


Figure 2.51: Crossover proposed in NEAT. The networks are essentially combined using the historical marking to prevent loss in information (Stanley and Miikkulainen, 2002).

mental effect of recombining parts of two networks as illustrated in Fig. 2.50. In contrast to common crossover techniques, the crossover proposed in NEAT essentially combines the architecture of two networks to avoid loss in information as shown in Fig. 2.51. Second, since historical marking can divide a population into species based on topological similarity, NEAT proposes speciation to protect diversity. In speciation, individuals with similar topologies compete within their own niche instead of the whole population. In this way, the emerging structures are protected from being eliminated by the reigning structure immediately and allow them to develop before competing with other niches in the population. Lastly, it minimizes the dimensionality of networks by initializing the population with the most basic network structure—no hidden nodes. Typically, neuroevolution algorithms initialize the population with random topologies (Angeline et al., 1994; Gruau et al., 1996) but it is not known whether the random topologies are necessary to produce good networks. Moreover, these are costly to optimize since more connections in a network correspond to more dimensions to be searched. However, by initializing from the most basic network structure, NEAT avoids optimizing and building from unnecessary random structures and only maintains structures that are relevant to the problem. In addition, by

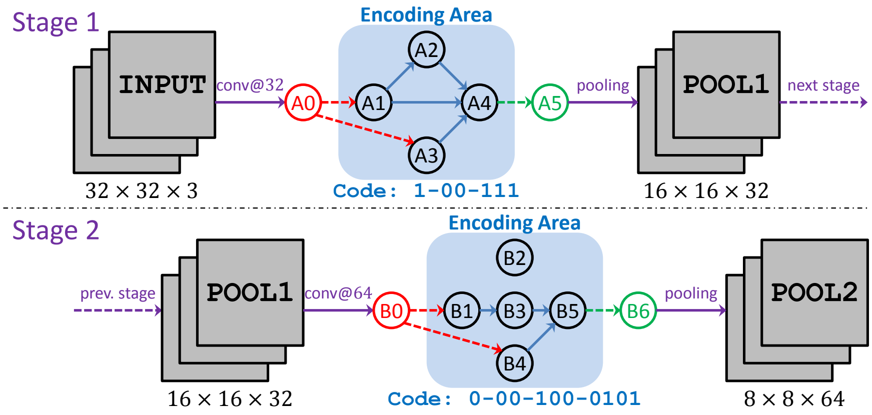


Figure 2.52: Genetic CNN fixed-length binary string encoding (Xie and Yuille, 2017).

searching through a small number of weight dimensions, NEAT also significantly reduces the required number of generations to arrive at a solution.

2.7 Toward CNN and Neuroevolution Combination

The convolutional neural network (CNN) architecture continually grows in depth to improve its performance. However, it is not known whether the hand-engineered conventional CNN architectures are optimal for a given dataset or application. Since researchers have demonstrated the ability of neuroevolution to optimize the connection weights and architecture of ANNs, different studies are also recently proposed to apply neuroevolution to CNN. However, unlike the neuroevolution methods for ANN, the proposed neuroevolution methods for CNN are typically concerned with architecture optimization only and leave the weights optimization to backpropagation. One of the reasons for this is the architecture depth of CNNs. The deep CNN architectures make it impractical to optimize the weight using neuroevolution. Moreover, the training methods (e.g., momentum, weight decay) developed for CNN allow better weights optimization. Instead, the focus of neuroevolution is to evolve CNN architectures, which have complex parts (e.g., convolutional block, pooling block, skip connections) unlike ANN with simple nodes. As discussed in Section 2.3, the CNN architecture plays a crucial role in the performance of the network.

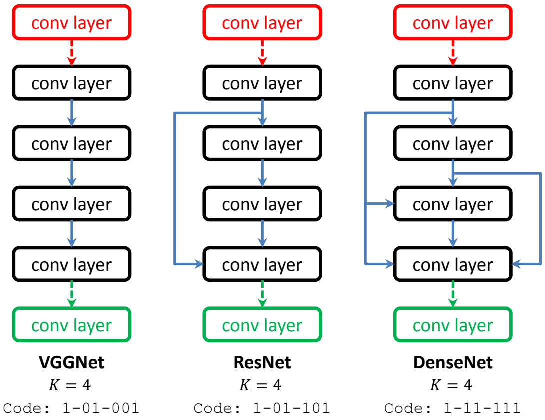


Figure 2.53: Fixed-length binary string encoding of VGGNet, ResNet, and DenseNet (Xie and Yuille, 2017).

2.7.1 Developments in Neuroevolution Approaches to CNN

Xie and Yuille (2017) proposed genetic CNN to automatically design the network architecture. To facilitate the evolution, CNN architectures are represented by a fixed-length binary string as shown in Fig. 2.52. The authors divided a CNN architecture into stages where each consists of convolutional blocks and pooling blocks combination. As seen in Fig. 2.52, each node in the encoding area corresponds to a convolutional block. If a block has multiple inputs, the sum of those inputs is taken. Furthermore, each convolutional block is followed by batch normalization and ReLU. The first and last nodes are not encoded. The binary string only encodes the connections in the intermediate nodes, in which 1 indicates a connection and 0 indicates no connection. In the stage 1, the binary string $1-00-111$ can be interpreted as $A1 \rightarrow A2, A1 \rightarrow A3, A2 \rightarrow A3, A1 \rightarrow A4, A2 \rightarrow A4, A3 \rightarrow A4$. This type of encoding can also represent the building blocks of conventional CNN architectures, which are shown in Fig. 2.53. After encoding the network, the binary string is subjected to genetic operators (e.g., mutation) to evolve. Afterward, the binary string is converted back to a network and trained normally. The example network architectures produced by genetic CNN can be seen in Fig. 2.54. The best networks produced have performed an error rate of 7.10% and 29.03% on the CIFAR-10 and CIFAR-100 datasets respectively. On the CIFAR-10 dataset, the total execution time took 17 GPU-days using 50 generations and 20 individuals.

Another approach to applying neuroevolution to CNN is proposed by Suganuma et al. (2017) called CGP-CNN. Instead of binary encoding, the authors used Cartesian

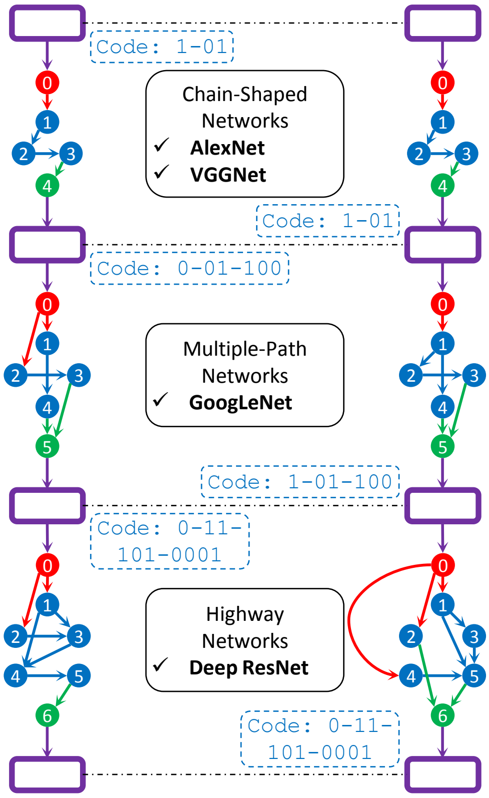


Figure 2.54: Two structures learned independently by genetic CNN (Xie and Yuille, 2017).

genetic programming (CGP) to represent the network and subsequently evolve them. The example genotype and phenotype of a network are shown in Fig. 2.55. The blocks used in evolution are convolutional block, resblock (resembling a ResNet block), max pooling, and average pooling. In addition, the authors introduced two types of skip connections, which are concatenation and summation, unlike genetic CNN that has summation only. The concatenation, which resembles GoogLeNet and DenseNet operations, takes two inputs and simply stacks them. On the other hand, summation, which resembles the ResNet operation, sums the input together. If the input sizes are different, the smaller input is padded with zeroes and then summed with the larger input. There are two main configurations tested. The first configuration, called ConvSet, uses convolutional block and all the pooling and skip connection blocks. The second configuration, called ResSet, is the same as ConvSet but replaces the convolutional block with the resblock. The produced network architectures for each configuration are exhibited in Fig. 2.56. The best evolved networks have achieved an error rate of 6.75% and 5.98% using the ConvSet and ResSet configurations respectively on the CIFAR-10 dataset. The network produced by the ResSet configuration took 14 days to complete the evolution process.

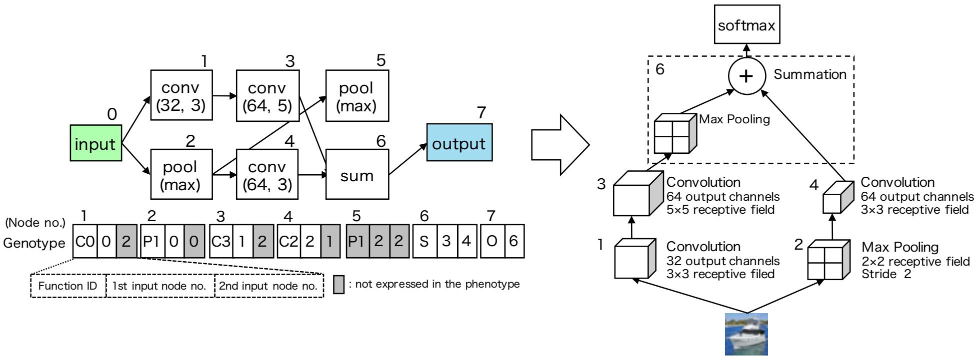


Figure 2.55: Example genotype (left) and phenotype (right) of a network using CGP. The phenotype corresponds to the CNN network architecture (Suganuma et al., 2017).

The neuroevolution approaches to CNN discussed previously have taken significant time and resources to produce their respective best network. However, there is a proposed neuroevolution that prioritizes the limited resources to evolve CNN architectures. This method is called aggressive genetic programming (GP) (Li et al., 2018). In this method, the CNN architectures are encoded using an acyclic graph, in which a node represents a block (e.g., convolution, pooling, FC, concatenation, etc.). The main difference in this method is the use of aggressive selection as shown in Fig. 2.57. In aggressive selection, the fittest individuals are taken and only they produce children networks by applying different mutations. It is different from tournament selection, where individuals with average fitness can be selected. The reason behind this approach is to avoid developing individuals that may be replaced along the course of evolution since the aim is to save resources. However, the diversity of the population suffers in this strategy. To alleviate this problem, the authors complemented the aggressive selection with numerous mutations. In addition to adding blocks to the network, altering the blocks and removing the blocks are also included as separate mutations. The mutation process can be seen in Fig. 2.58. The networks discovered for CIFAR-10 and CIFAR-100 datasets are shown in Fig. 2.59. The best networks produced have achieved 90.52% and 68.04% accuracy on CIFAR-10 and CIFAR-100 datasets, which took 72 GPU-hours and 184 GPU-hours respectively to complete the evolution.

The neuroevolution for CNN architectures is typically used for image classification. However, the architectures developed by neuroevolution can also specialize in other applications. In one study, the YOLOv2 architecture is evolved to improve the object detection performance on dangerous objects X-ray (Tsukada et al., 2020). Another study

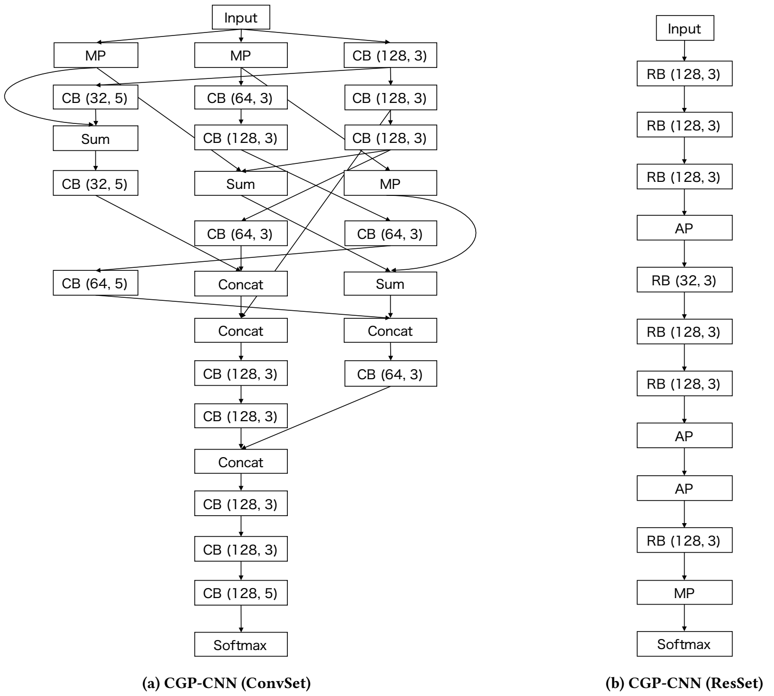


Figure 2.56: The networks produced by the CGP-CNN on two configurations (Suganuma et al., 2017).

proposed to use neuroevolution to replace the feature extraction network of the YOLOv3 with an optimized architecture (Operiano et al., 2020). Moreover, a separate study proposed to utilize neuroevolution to search for CNN architectures that are robust to the transferability of adversarial examples (Kotyan and Vargas, 2020).

2.8 Summary

In summary, this chapter starts with the discussion about ANN as an algorithm inspired by the biological neurons in the brain that automatically learns to produce a desired output without explicit programming. The potential of ANNs are visually realized in the applications of CNN, a special type of neural network designed for image processing. The developments in CNN such as training techniques, architecture changes, are carefully reviewed in this chapter. In addition, the common datasets used for benchmarking CNNs and data augmentations that extend the information provided by datasets are explored. Furthermore, since training a deep CNN network requires a significant amount of data and computational resources, taking advantage of the learning of a trained network to effectively train a different network is examined in this chapter. CNN architectures continue to grow deeper to increase their performance but they need large datasets and compu-

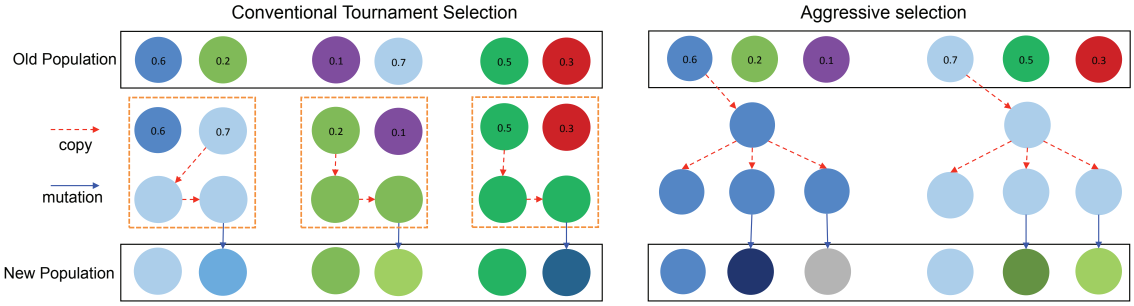


Figure 2.57: Comparison of tournament selection and aggressive selection. The aggressive selection takes the fittest individuals in the population and these individuals are mutated differently to produce children. (Li et al., 2018).

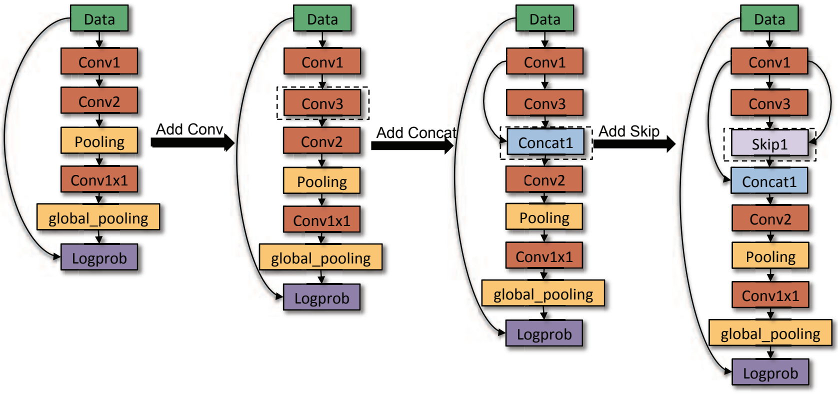


Figure 2.58: CNN mutation process of Aggressive GP (Li et al., 2018).

tational power to train them properly. Therefore, evolutionary methods that discover optimal networks which perform similar to deep networks are also explored. The evolutionary methods for ANN are first surveyed, and then the relatively novel evolutionary approaches to CNN are examined.

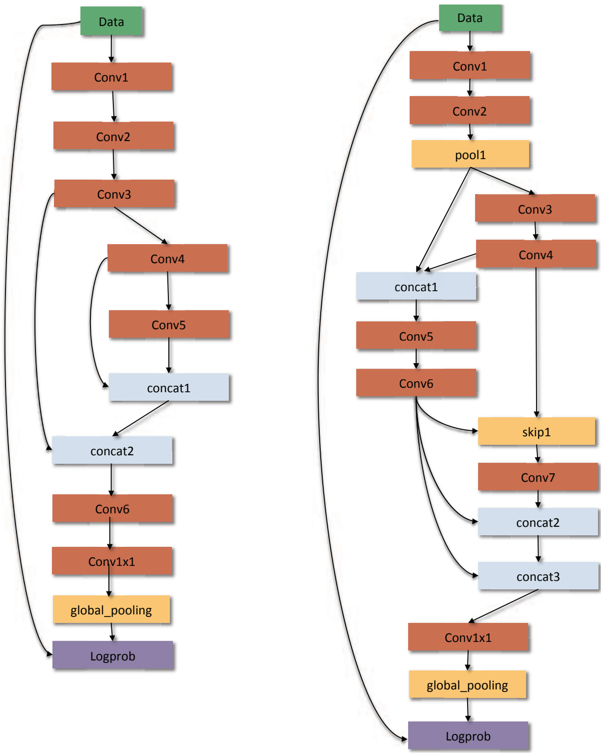


Figure 2.59: CNN architectures discovered by Aggressive GP for CIFAR-10 (left) and CIFAR-100 (right) datasets (Li et al., 2018).

CHAPTER III

CNN DATA AUGMENTATIONS AND APPLICATIONS

Convolutional neural networks (CNN) is a machine learning technique that achieves phenomenal performance in applications such as image classification and object detection. However, the performance of a CNN varies greatly depending on the tuning of hyper-parameters (e.g., learning rate and data preparation) as demonstrated in the AlexNet training method (Krizhevsky et al., 2012). In Section 3.1, a data augmentation method is proposed to improve the performance of a CNN that is trained on a small dataset. Furthermore, CNN can be combined with different algorithms to create a specialized application. In Section 3.2, a CNN is combined with Haar Cascades to detect and recognize alphabet hand sign language.

3.1 Pre- and Post-training Data Augmentations

CNN usually relies on huge datasets to learn the proper features of an object from an image and generalize well on images it has not seen before. However, there are specialized applications that have limited datasets. For example, dangerous objects X-ray dataset has limited images due to the difficulty in data collection and reluctance of X-ray companies to share data which can jeopardize public safety. To address the dataset limitation, several methods are proposed to extract more information from the dataset such as data synthesis (Zou et al., 2018), in which a new set of images are created by cutting and combing existing images, and data augmentation (Krizhevsky et al., 2012), in which the image is slightly modified to create new image examples (e.g., image mirroring).

In this section, there is a total of six novel data augmentations proposed to help the CNN increase its accuracy or mean average precision (mAP) on object detection. The dataset used is the dangerous object X-ray dataset as shown in Section 2.4.1.5, which contains prohibited items such as knives, scissors, and bottles. The first two data augmentations are the *simple objects addition* and *quadrant addition*, which are both applied on the images before training and collectively denoted as Pre-training data augmentations (Pre-TDA). The last four data augmentations are the *quadrant computation*, *zoom*

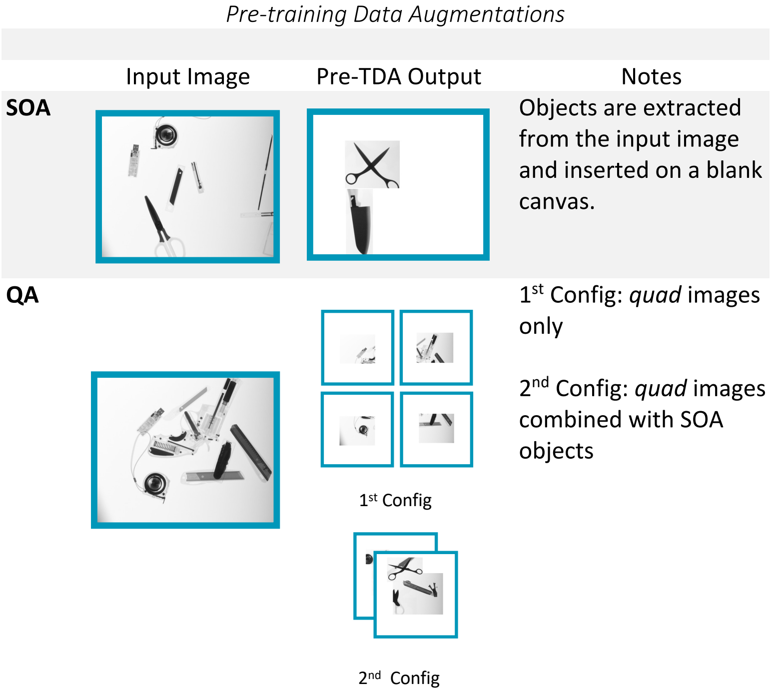


Figure 3.1: The Pre-TDA are shown here. In simple objects addition, an image is inserted with a random object from the training set to become a new image example. In quadrant addition, an image is divided into quadrants to produce new images. Moreover, a random object can also be inserted into a quadrant image to make a complex new image.

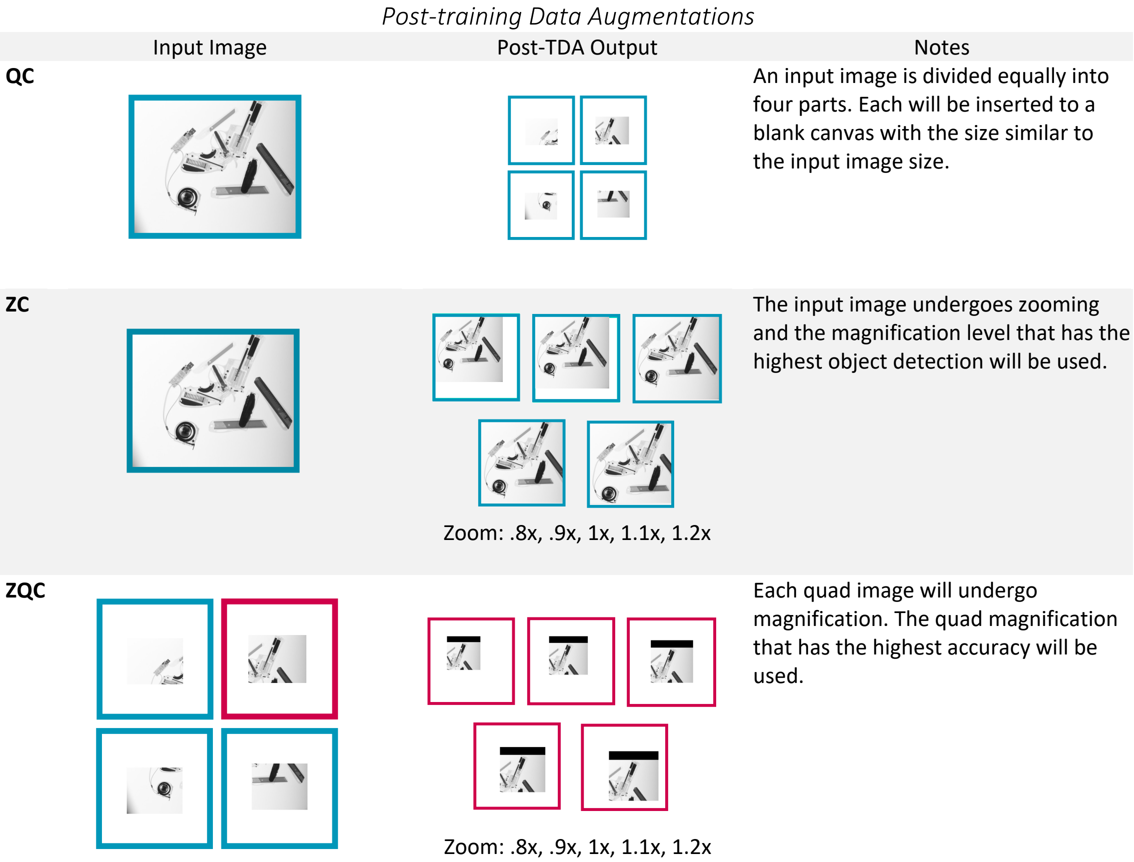
computation, *zoomed quad computation*, and *quad + zoom computation*, which are applied to the images during test-time. These four augmentations are collectively denoted as Post-training data augmentations (Post-TDA).

Simple objects addition (SOA) is a data augmentation that automatically combines objects from different images into a blank image canvas arbitrarily. Since the objects are from X-ray images, the objects can overlap. The overlapping area is dealt with by applying the darkest pixel between the two images. In the second data augmentation, which is quadrant addition (QA), an image is divided into four parts or quadrants. Although there is a possibility that an object inside the image is cut into multiple parts, the piece of that object in a quadrant will retain its label as long as the area of the part is more than 35% of the total area of the whole object. Subsequently, the quadrant, which is one-fourth of the image is placed on a blank image canvas to become a new image (denoted as quad image). Moreover, the quad image can be populated with the extracted objects from SOA. The intuition of the QA data augmentation is to break the pattern of object arrangements. Since the X-ray dataset has limited images, most of the images tend to look similar and differ only with some displaced objects. This similarity makes the network associate the objects together and the detection rate drops when they are separated. Furthermore, the division helps the network detect a partially represented object. Both the SOA and QA data augmentations are shown in Fig. 3.1.

In the Post-TDA, the augmentations are applied to the test set images during network evaluation. This helps increase the confidence of a network during the prediction by showing many viewpoints of the same image to the network. Although the test set images are typically untouched to have a fair measure of the network performance, the goal of Post-TDA is not to increase the network performance unreasonably and compare it to other networks. Instead, the aim of Post-TDA is to extend the network capability to detect objects that is clearly bounded by the limited dataset. An increase in the network performance is important in real-life applications such as in dangerous objects X-ray detection.

The first Post-TDA is the quadrant computation (QC). Similar to QA, the input image is divided into quadrants and form quad images. The image and its corresponding quad images are fed to the network to find dangerous objects. The quad images are support images, which provide additional viewpoints for the network in case it fails to detect all the dangerous objects in the uncut image. To test the effectiveness of the QC data augmentation, two datasets of comparison are utilized. The first dataset is the original test set and the second dataset is the combination of the original test set images and quad images. To fill the second test set, each image in the test set and its corresponding quad images, as a group, are compared. If the original test image has higher accuracy, which means the network properly identifies the objects, it will be added to the second dataset. However, if the quad images have higher accuracy collectively, which means the network identifies the objects better when divided into quadrants, then these images are added to the second set. The process repeats until all the images in the test set are processed. Afterward, the complete second set is evaluated and compared to the performance of the first dataset. The process described is summarized by equations (3.1) and (3.2).

Let I_i be an input image, $Q(I_i)$ as the corresponding quad images of I_i, O as the output dataset of the QC procedure, and $AP(I_i)$ as the accuracy or mean average



Post-training Data Augmentations

	Input Image	Post-TDA Output	Notes
QC			An input image is divided equally into four parts. Each will be inserted to a blank canvas with the size similar to the input image size.
ZC		Zoom: .8x, .9x, 1x, 1.1x, 1.2x	The input image undergoes zooming and the magnification level that has the highest object detection will be used.
ZQC		Zoom: .8x, .9x, 1x, 1.1x, 1.2x	Each quad image will undergo magnification. The quad magnification that has the highest accuracy will be used.

Figure 3.2: Different Post-TDA are shown here. In quadrant computation, the input image is divided into quadrants as support images. In zoom computation, the image is zoomed into 4 levels to obtain the optimal magnification. In zoomed quad computation, the quad images are zoomed into 4 levels to obtain the optimal magnification of each quad image.

precision (mAP) function. Given a set of test images $\{I_1, I_2, \ldots, I_N\}$, O (i.e., second set) is constructed as

$$O := \bigcup_{i=1}^{N} S_i, \tag{3.1}$$

where

$$S_i := \begin{cases} \{I_i\} & \text{if } AP(I_i) \geq AP(Q(I_i)) \\ Q(I_i) & \text{if } AP(I_i) < AP(Q(I_i)) \end{cases}. \tag{3.2}$$

The example images are shown in Fig. 3.2 and the process can be visualized in Fig. 3.3.

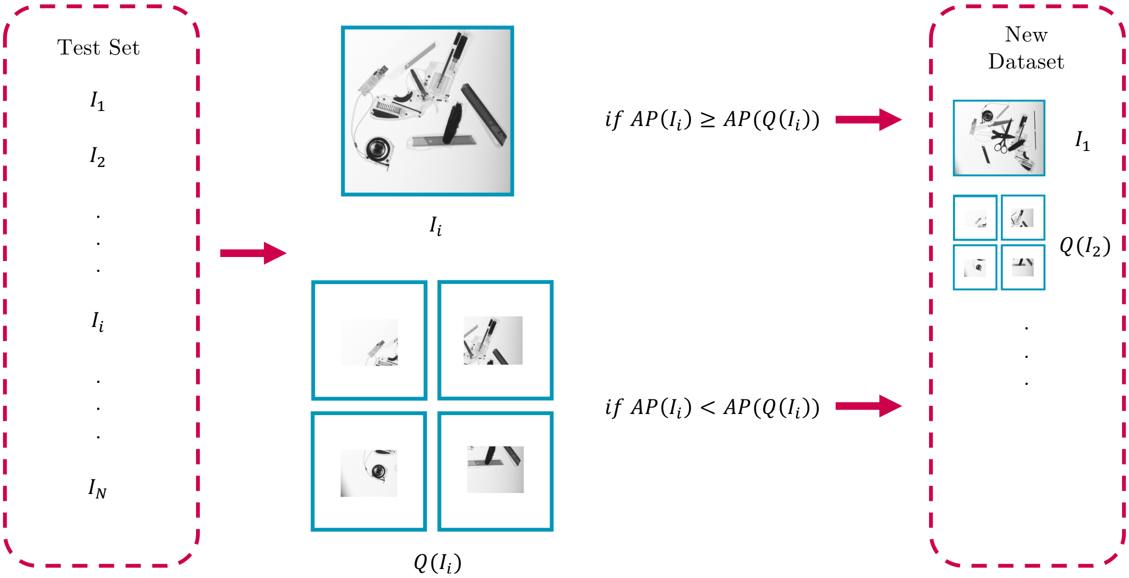


Figure 3.3: To fill a new dataset with the combination of the original and quad images, each image in the test set is compared with its corresponding quad images through their accuracy. If the original test image has higher accuracy than the corresponding quad images collectively, the original image will be added to the dataset. Otherwise, the quad images will be added to the dataset instead of the original image.

The second Post -TDA is zoom computation (ZC). In this data augmentation, the image is magnified to different magnification sizes to let the network find objects in the size that matches its learned features. If the network detects an object in a different magnification than the original image size, it implies that there is a slight mismatch between the training set and the test set. Although the network is trained, it is not object size invariant enough due to a limited dataset. Similar to QC, the effectiveness of ZC is quantified by comparing two datasets, which are comprised of first, the original test set images, and second, the combination of original and zoomed test set images. Before filling the second set, each image in the test set is magnified into four magnification levels (i.e., .8x, .9x, 1.1x, and 1.2x). Together with the original image (i.e., 1x), there are five images that are evaluated by the network. For each test image, the magnification level that has the highest accuracy is added to the second set. After the second set is completed, it is evaluated by the network as a whole dataset and compared with the first dataset. The ZC images are shown in Fig. 3.2. The process encapsulated by equations (3.3) and (3.4). The I_i and $AP(I_i)$ definitions are the identical to the QC equations. Let $Z(I_i)$ be the zoomed images of I_i (i.e., four magnification levels and original size), and O as the output dataset of ZC method. Given a set of test images $\{I_1, I_2, \ldots, I_N\}$, O is constructed as

$$O := \bigcup_{i=1}^{N} S_i, \tag{3.3}$$

where

$$S_i := \{\text{argmax}_{\bar{I} \in Z(I_i)} \text{AP}(\bar{I})\}. \tag{3.4}$$

The third Post-TDA is the zoomed quad computation (ZQC), which is the combination of the previous methods. In ZQC, the images are divided into quadrants, and the quad images are presented in four magnification levels as in ZC. Hence, the quad images can give focus to certain portions of an image and the corresponding magnification can give a wider object size representation. Similar to QC, the images produced by ZQC are used as support images during application. Again, there are two datasets of comparison as explained in the previous methods. To fill the second set, each test set image is divided into quadrants. Then, every quad image is subjected to multiple magnification levels, and the size that has the highest accuracy is used. As a result, the quad images can have different magnification levels. Afterward, the original test set image is compared to its corresponding size-optimized quad images to fill the second set similar to QC. The ZQC images are shown in Fig. 3.2. The ZQC process is summarized in equations (3.5)– (3.7), in which equation (3.6) compares the test images and their corresponding zoomed quad images, and equation (3.7) obtains the best magnification level of each quad image.

The I_i, $Q(I_i)$, $AP(I_i)$ are defined in the previous methods. Let $Z(q)$ be the five magnifications of a quad image similar to the ZC, T_i as the selected zoomed quad images with the highest accuracies, and O as the output dataset of ZQC. Given a set of test images $\{I_1, I_2, \ldots, I_N\}$, O is produced as

$$O := \bigcup_{i=1}^{N} S_i, \tag{3.5}$$

where

$$S_i := \begin{cases} \{I_i\} & \text{if } AP(I_i) \geq AP(T_i) \\ T_i & \text{if } AP(I_i) < AP(T_i) \end{cases} \tag{3.6}$$

and

$$T_i := \{\text{argmax}_{\bar{I} \in Z(q)} AP(\bar{I}) \mid q \in Q(I_i)\}. \tag{3.7}$$

The final Post-TDA is the quad and zoom computation (QZC). QZC is the combination of zoom computation and zoomed quad computation. The ZC outputs the best magnification for each test set image, whereas ZQC outputs the best magnification for the corresponding quad images. Essentially, QZC combines the advantages of ZC and ZQC to produce an optimal dataset configuration. As performed in the previous Post-TDA methods, the QZC employs two datasets to measure its effectiveness. To create the second set, the output dataset of ZC on a given test set is compared to the output dataset of ZQC on the same test set. Whichever has the higher accuracy between the image and its corresponding quad images collectively are added to the second set as shown in Fig. 3.3. It should be noted that ZQC uses the original test images and not their best-magnified image counterpart. The QZC process is summarized in equations (3.8) – (3.11). Equation (3.9) evaluates the images produced by equations (3.10) and (3.11) and takes the best image representation. Equation (3.10) produces the best magnification of test images and equation (3.11) produces the best magnification of their corresponding quad images.

The I_i, $Q(I_i)$, $AP(I_i)$, $Z(I_i)$, and $Z(q)$ are defined in the previous methods. Let G_i be the selected magnification of an input test image similar to ZC output, and T_i be the corresponding selected zoomed quad images identical to ZQC output. Given a set of test images $\{I_1, I_2, \ldots, I_N\}$, the output dataset of QZC experiment O is constructed as

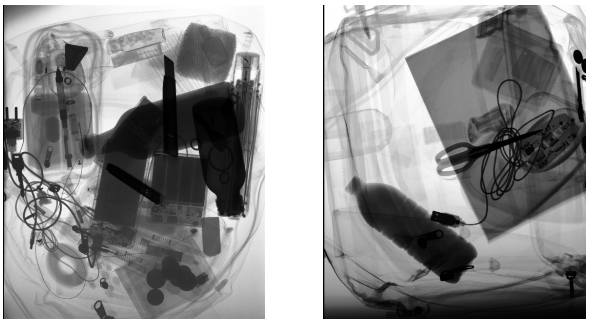
$$O := \bigcup_{i=1}^{N} S_i, \tag{3.8}$$

where

$$S_i := \begin{cases} G_i & \text{if } AP(G_i) \geq AP(T_i) \\ T_i & \text{if } AP(G_i) < AP(T_i) \end{cases}, \tag{3.9}$$

$$G_i := \{\text{argmax}_{\bar{K} \in Z(I_i)} \text{AP}(\bar{K})\}, \tag{3.10}$$

Raw Images



Synthesized Images

Figure 3.4: Raw and synthesized images of dangerous objects X-ray (Zou et al., 2018).

and

$$T_i := \{\text{argmax}_{\bar{I} \in Z(q)} AP(\bar{I}) \mid q \in Q(I_i)\}. \tag{3.11}$$

3.1.1 Pre-TDA and Post-TDA Experiments

There are six experiments conducted to evaluate each data augmentation proposed. The dataset utilized in the experiments is the dangerous objects X-ray as discussed in Section 2.4.1.5. From the dataset paper (Zou et al., 2018), three experiment results are obtained as benchmarks. The first experiment used a raw X-ray dataset, which has 662 training images and 442 test images, on the YOLOv2. Conversely, the second and third experiments employed synthesized images (i.e., images created from processing the training images), which are 331 and 3010 images respectively in addition to the raw images. The example raw and synthesized images are shown in Fig. 3.4. However, since the network used in the experiments is YOLOv3, two additional benchmarks are produced. The first YOLOv3 benchmark uses the raw dataset only, whereas the second YOLOv3 benchmark uses the raw dataset with additional 500 synthesized images. The benchmarks are summarized in Table 3.1.

Table 3.1: The benchmark for the dangerous objects X-ray. The accuracies on YOLOv2 are obtained from the results in the dangerous objects X-ray paper, whereas the YOLOv3 results are produced.

Benchmark	Details	Accuracy
YOLOv2	Raw Dataset (662 Train Set, 442 Test Set)	84.20%
YOLOv2+Synth1	Raw Dataset + 331 Synthesized Train Set	84.91%
YOLOv2+Synth2	Raw Dataset + 3010 Synthesized Train Set	86.42%
YOLOv3	Raw Dataset (662 Train Set, 442 Test Set)	84.66%
YOLOv3+Synth1	Raw Dataset + 500 Synthesized Train Set	86.21%

Table 3.2: The benchmarks are compared with the SOA results. The SOA has three configurations, which are raw dataset with 1-2 objects, raw dataset with 3-4 objects, and raw dataset with 3-4 objects and class balancing. Adding SOA with class balancing improves the performance of the network.

Benchmark	Accuracy
YOLOv2	84.20%
YOLOv2+Synth1	84.91%
YOLOv2+Synth2	**86.42%**
YOLOv3	84.66%
YOLOv3+Synth1	86.21%
SOA	Accuracy
Raw + SOA 1-2	83.19%
Raw + SOA 3-4	86.41%
Raw + SOA 3-4 with CB	**87.50%**

3.1.1.1 SOA Experiment

In the SOA experiment, the objects are extracted from the training set and randomly added to the training images to create new images automatically. There are three configurations of SOA used. The first configuration adds 1-2 objects randomly to images, whereas the second and third configurations add 3-4 objects. However, the third configuration adds objects with class balancing (i.e., using more objects from classes that have low accuracies). For each configuration, 500 images are generated and added to the raw dataset as dataset extension during training.

The experiment results in Table 3.2 show that adding more objects (i.e., SOA 3-4)

Table 3.3: The benchmarks are compared with the QA results. The QA has two configurations, which are raw dataset with quad images, and raw dataset with quad images and SOA objects. Extending the dataset with quad images improves the performance of the network.

Benchmark	Accuracy
YOLOv2	84.20%
YOLOv2+Synth1	84.91%
YOLOv2+Synth2	**86.42%**
YOLOv3	84.66%
YOLOv3+Synth1	86.21%
QA	Accuracy
Raw + QA	86.70%
Raw + QA + SOA	**87.00%**

to the training images reinforces the object features to the network, which results in high accuracies. Moreover, if the objects are consciously added through class balancing, the network can improve its learning on weak classes. For example, the network has low accuracies on the scissors and knife classes because they look similar. Supplying SOA objects more on scissors and knives improves network performance. The difference between SOA and synthesizing images is that synthesizing images requires expertise and image processing to combine images, in which the synthesized images can look different than the raw images. Hence, even adding approximately 3000 synthesized images (YOLOv2+Synth2) may not significantly improve the network performance. The results show that SOA can be a substitute for synthesizing images.

3.1.1.2 QA Experiment

In the QA experiment, the quad images of the training images are collected, and then, 500 quad images are randomly selected as dataset extension. The QA has two configurations, which are quad images only and quad images combined with SOA objects. As shown in Table 3.3, adding quad images to the training set slightly increases the network accuracy over the YOLOv3+Synth1. Moreover, adding SOA objects to the quad images slightly increases the network accuracy even more than quad images only. The QA results are comparable to those of the SOA, which implies that although the different features are highlighted between the QA and SOA, the outcome is still similar. The SOA

Table 3.4: The QC data augmentation method is applied on the QA trained networks because QC utilizes quad images. Then, the results are compared with the YOLOv3+Synth1 and the QA trained networks. Using quad images as additional viewpoints for the network improves its ability to detect objects.

Benchmark	Accuracy
YOLOv3+Synth1	86.21%
Raw + QA	86.70%
Raw + QA + SOA	**87.00%**
QC	Accuracy
Raw + QA	**88.26%**
Raw + QA + SOA	87.40%

helps the network learn by reinforcing whole object representations, whereas QA helps the network learn by teaching cropped object representations.

3.1.1.3 QC Experiment

The QC data augmentation is evaluated as described in Section 3.1. Since QC is a Post-TDA, it utilizes trained networks that it will try to improve the performance. For the QC experiment, the networks employed are the QA trained networks because these are trained with quad images. The results in Table 3.4 demonstrate that using quad images of every test image as additional viewpoints for the network improves the performance of the network. Despite the network being trained with limited images, the network learning can be extended by providing different viewpoints of the same image. Notably, the improvement in the network trained by the second configuration of QA has a small improvement after using QC. This is attributed to the additional SOA objects that the network has to learn aside from the cropped object representation.

3.1.1.4 ZC Experiment

The ZC data augmentation is evaluated as described in Section 3.1. Since ZC does not utilize image division as in QC, any trained network can be utilized with ZC. To test the effectiveness of the ZC method, the YOLOv3, best SOA network (from Section 3.1.1.1), and best QA network (from Section 3.1.1.2) are applied with ZC. As shown in Table 3.5, obtaining the optimal magnification of test images significantly improves the

Table 3.5: The ZC method is applied on the YOLOv3, best SOA network, and best QA network. The results are compared with the same networks to confirm whether ZC improves the performance of the networks. The results show that ZC can provide the object size that fits the features learned by the network, thereby improving the network performance.

Benchmark	Accuracy
YOLOv3	84.86%
SOA (Best)	**87.50%**
QA (Best)	87.00%
ZC	Accuracy
YOLOv3	88.72%
SOA (Best)	88.98%
QA (Best)	**89.02%**

Table 3.6: The ZQC data augmentation is applied to QA trained networks because ZQC utilizes quad images. The results are compared with the YOLOv3+Synth1, best SOA network, and best QA network. Providing the appropriate size of the quad images using ZQC improves the performance of the network.

Benchmark	Accuracy
YOLOv3+Synth1	86.21%
QA (Best)	87.00%
QC (Best)	**88.26%**
ZQC	Accuracy
Raw + Quad	**89.91%**
Raw + Quad + SOA	88.86%

performance of all the networks tested. Furthermore, the results imply that the networks have learned the object features properly but due to the small dataset, the object size variation is not well represented. Therefore, ZC can help networks trained with limited datasets to identify objects that have sizes different from the dataset.

3.1.1.5 ZQC Experiment

The performance of ZQC is evaluated as depicted in Section 3.1. In contrast to ZC, the ZQC subjects the quad images to different magnification instead of the original image only. The networks from the QA experiment are used to apply ZQC because ZQC

Table 3.7: The QZC data augmentation method is applied to networks trained by QA because it utilizes quad images. Since QZC is the combination of the Post-TDA methods, the QZC results are compared with all the best Post-TDA networks. The results show that the QZC can provide a wide representation of an image, and therefore achieve significant improvement in the network performance compared to previous Post-TDA methods.

Benchmark	Accuracy
YOLOv3+Synth1	86.21%
QC (Best)	87.40%
ZC (Best)	**89.02%**
ZQC (Best)	88.86%
QZC	Accuracy
Raw + Quad	**91.25%**
Raw + Quad + SOA	90.39%

utilizes quad images. The baselines are the YOLOv3+Synth1, best QA network, and best QC network. As shown in Table 3.6, the ZQC method improves the performance of the networks. When the QA results are compared with ZQC results, the accuracy increase is consistent. Therefore, obtaining the proper quad image size can prevent possible object size mismatches, which results in better network detection.

3.1.1.6 QZC Experiment

The performance of QZC data augmentation is evaluated as depicted in Section 3.1. The QZC is the culmination of all the Post-TDA methods, in which a test image is represented in multiple ways. The best magnification of a test image and the best magnification of its corresponding quad images are used to help the network detect the objects in the image. The QZC results are compared to YOLOv3+Synth1 and all the previous Post-TDA methods. The results in Table 3.7 demonstrate a significant improvement provided by the QZC method. The wide variety of viewpoints in QZC has helped the network detect objects more accurately. In particular, when the QZC result is compared with the YOLOv3+Synth1 result, there is a remarkable increase of 5% in accuracy. These improvements cannot simply be obtained by training a network with a limited dataset regardless of the number of epochs and hyperparameters tuning. Therefore, the Post-TDA methods are helpful to the network during the application, especially the QZC method.

Table 3.8: The summary of experiments result. Among the data augmentations applied on the training set, the SOA data augmentation yields the best accuracy. On the other hand, among the data augmentations applied on test-time, the QZC, which is the combination of all Post-TDA methods, yields the best accuracy.

	Pre-TDA			
	YOLOv2+Synth2	YOLOv3+Synth1	SOA (Best)	QA (Best)
Accuracy	86.42%	86.21%	**87.50%**	87.00%
	Post-TDA			
	QC (Best)	ZC (Best)	ZQC (Best)	QZC (Best)
Accuracy	88.26%	89.02%	89.91%	**91.25%**

3.1.2 Experiments Summary

There are different ways to extract more information from a limited dataset. In the original paper of the dangerous objects X-ray, the authors used synthesized images to extend the image count. However, synthesizing images requires knowledge in image processing and time to design new images. Moreover, the synthesized images may look different from the raw image as seen in Fig. 3.4, which results in a small increase in accuracy even with substantially more images (e.g., YOLOv2+Synth2). The data augmentations in the Pre-TDA are proposed to draw out more information from a limited dataset. It uses the bounding box information from the labels of the training set to automatically obtain the objects in an image and attach them to the training set images randomly to create new images. Since the objects extracted do not undergo image processing, the produced images match the training set images. As shown in Table 3.8, even a simple SOA method can outperform networks with synthesized images. Therefore, the proposed Pre-TDA methods can help networks increase their accuracy during training.

Networks that are trained on limited datasets can only reach a certain accuracy ceiling. After reaching that certain accuracy, even with network modification, hyper-parameters tuning, longer training times, or data augmentations, the accuracy cannot increase substantially. The cause of the problem is the small dataset, in which the feature that can be extracted are finite. It is one of the reasons why the success of deep neural networks is predicated on a deep network and massive dataset. Since in specialized applications, the dataset is usually limited, Post-TDA are proposed. Post-TDA are data

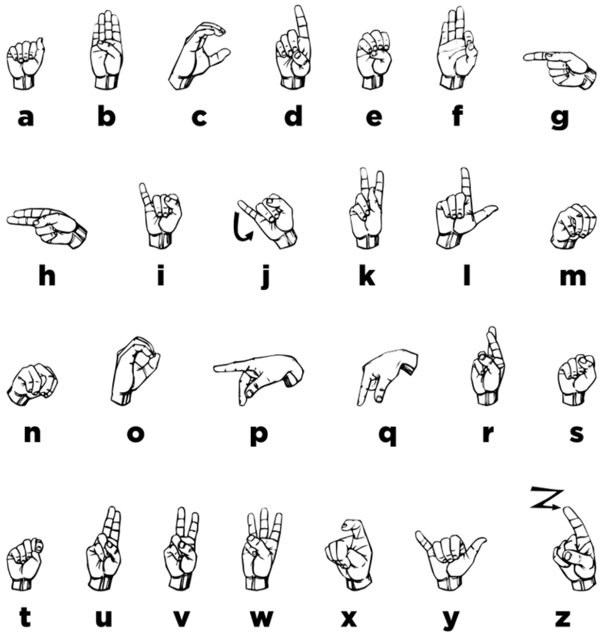


Figure 3.5: American Hand Sign Language (Alphabet) (Abner, 2014).

augmentations that are applied during test-time. It aims to provide the network with multiple viewpoints of the test image to increase its chances of detecting objects. Essentially, the Post-TDA try to match the image to the features learned by the network. In applications such as baggage checks, it is important to detect dangerous objects, and Post-TDA increase that possibility. As seen in Table 3.8, the QZC has achieved 91.25%, which is a substantial 5% increase from the YOLOv3 benchmark. This increase cannot be achieved by training on the limited dataset alone. Hence, the proposed Post-TDA methods can significantly help networks trained on limited datasets extend their capability.

3.2 Hand Sign Language Detection and Recognition

Alphabet hand sign language, as shown in Fig. 3.5, is used by hearing-impaired individuals to communicate words that do not have a gesture such as proper names. Developments in computer vision have made it possible to locate and identify hand signals in an image or stream of images (i.e., a video). In this section, a method is proposed to detect the location of a hand in an image, and subsequently identify it. The relative quickness of Haar Cascades compared to a deep CNN is utilized to localize a hand in an image, and then, the ability of CNN to classify images with high accuracy is utilized to identify the letter that the cropped hand image represents.

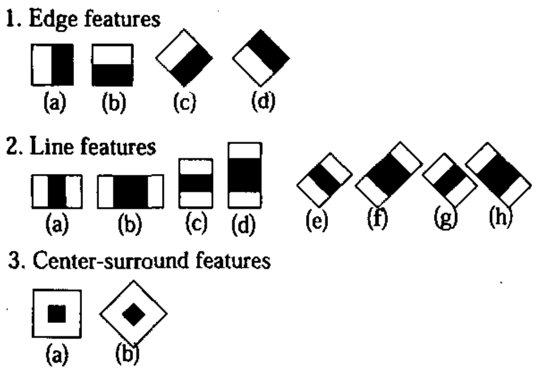


Figure 3.6: Haar-like Features (Lienhart and Maydt, 2002).

Figure 3.7: Haar Cascades implemented on hand sign language.

To detect the hand in an image, a Haar Classifier algorithm is employed (Viola and Jones, 2001). A Haar Classifier uses a set of cascade functions, which are trained using numerous positive images (i.e., images with the target object) and negative images (images without the target object), to detect the target objects in images. During training, Haar-like features are collected to extract features, which are calculated on adjacent rectangular regions at a distinct location in a detection window. The sum of pixel intensities in each region and the difference between the sums are consecutively calculated. These features measure the difference in intensity between the desired regions. The Haar-like features are shown in Fig. 3.6. Integral images are also employed to extract features from large images. Afterward, Adaboost training is implemented to train the classifier to use the features that best detect the target objects. Finally, a series of weak classifiers are combined using cascading classifiers to create a strong classifier. The Haar Cascades implemented on the hand sign language detection can be seen in Fig. 3.7.

After the hand region is extracted from the image, it is fed to the CNN to identify the letter it represents. The CNN architecture is made up of two consecutive convolution-

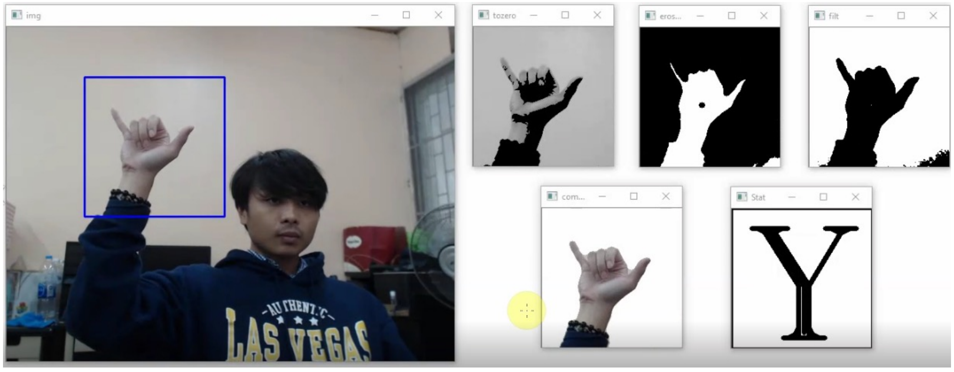


Figure 3.8: Hand Sign Language Detection and Recognition System.

pooling blocks, followed by a fully connected layer and a softmax layer. The network architecture is small because there are limited training images. The dataset consists of 200 images per letter in the English alphabet, in which two persons contributed exactly half of the dataset. The letter J and Z are excluded because they need hand motions to represent the letter. A total of 4800 images are divided into 4320 training images and 480 test images. Each image has a size of 200 × 200 pixels. The training epoch is set to 50, and the learning rate is 0.009. The optimizer used is adam optimizer.

The overall hand sign language detection and recognition system is summarized in the following steps:

Step 1. The video coming from the webcam is treated as a stream of images

Step 2. Haar Cascades is used to extract the hand region from the image.

Step 3. The hand image is pre-processed using thresholding and morphological operators to remove the background.

Step 4. The pre-processed hand image is fed to the CNN for classification.

Step 5. The output letter is displayed on the screen.

The process is shown in Fig. 3.8.

The test set results are displayed in Fig. 3.9. The total accuracy on the images from the first person and second person are 95.42% and 83.33% respectively. The combined accuracy is 89.38%. Most of the letters are correctly classified by the CNN but it has some

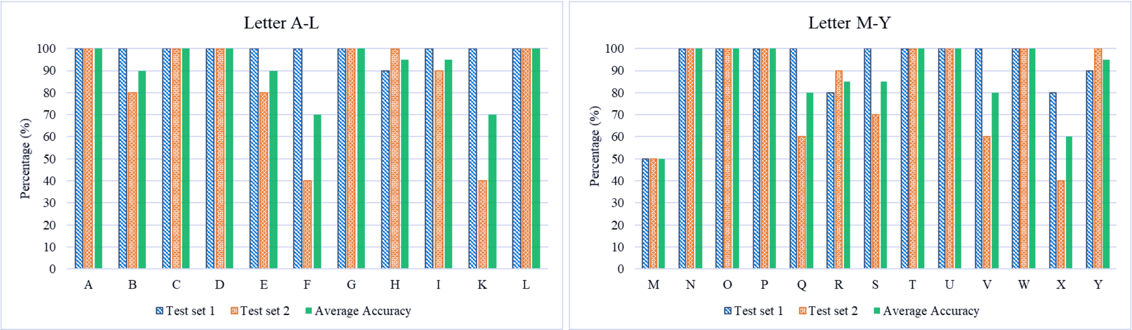


Figure 3.9: Accuracies on Hand Sign Language Test Set.

problems with images that look similar. For example, the letters K and V have similar shapes in hand sign language, as well as letters M and N. As a result, the CNN tends to select a letter over the other as seen in the letters M and N results. Nevertheless, as a hand sign language detection and recognition system, the proposed method is successful.

3.3 Summary

In this chapter, novel data augmentations on both training and test-time are proposed to improve the object detection of a network trained on a limited dataset. The data augmentation methods in the training improve the accuracy of a network by automatically generating new images using the objects extracted in the training set and adding the new images to the training set. Therefore, the network can learn more information from the training set. In addition, the data augmentation methods in the test-time improve the performance of a network by presenting different viewpoints of the input image such as magnifying the image in different levels. By matching the input image attributes with the learning of the network, the network can detect objects better. These methods are valuable to networks that have their performance limited by a small dataset. In another application in the chapter, the CNN is proposed to combine with Haar Cascades for hand sign language detection and recognition. The fast object detection of Haar Cascades and the accurate image classification of CNN produce a decent hand sign language detection recognition system.

CHAPTER IV

NEUROEVOLUTION FOR CNN TECHNIQUES AND APPLICATIONS

Neuroevolution is a powerful approach to automatically design an optimal network architecture. It uses evolutionary algorithms to find architectures that yield the best accuracies and can compete with the best hand-engineered networks such as ResNet (Sun et al., 2020). In Section 4.1, different proposed techniques to apply neuroevolution to CNN are carefully explained. Moreover, the proposed methods to utilize the learning of a trained network to neuroevolution-produced networks are discussed in Section 4.2. Aside from image classification, neuroevolution can also be utilized in different applications such as object detection and adversarial defense. In Section 4.3, the proposed neuroevolution methods to discover optimal CNN architectures for different applications are explained in detail.

4.1 Neuroevolution Techniques

The neuroevolution framework designed for convolutional neural network (CNN) can be summarized by the following steps.

Step 1. Initialize a population of networks

Step 2. Train the individual networks

Step 3. Evaluate each network using a fitness function

Step 4. Select and mutate the fit individuals to form the next population for the next generation

Step 5. Repeat the process in Step 2 until the end of the set generation

There are two configurations of neuroevolution tested, which are *generic neuroevolution* and *steady-state neuroevolution*. Depending on the configuration, the steps are applied differently.

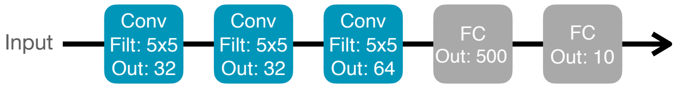


Figure 4.1: Basic Network for Neuroevolution Initialization.

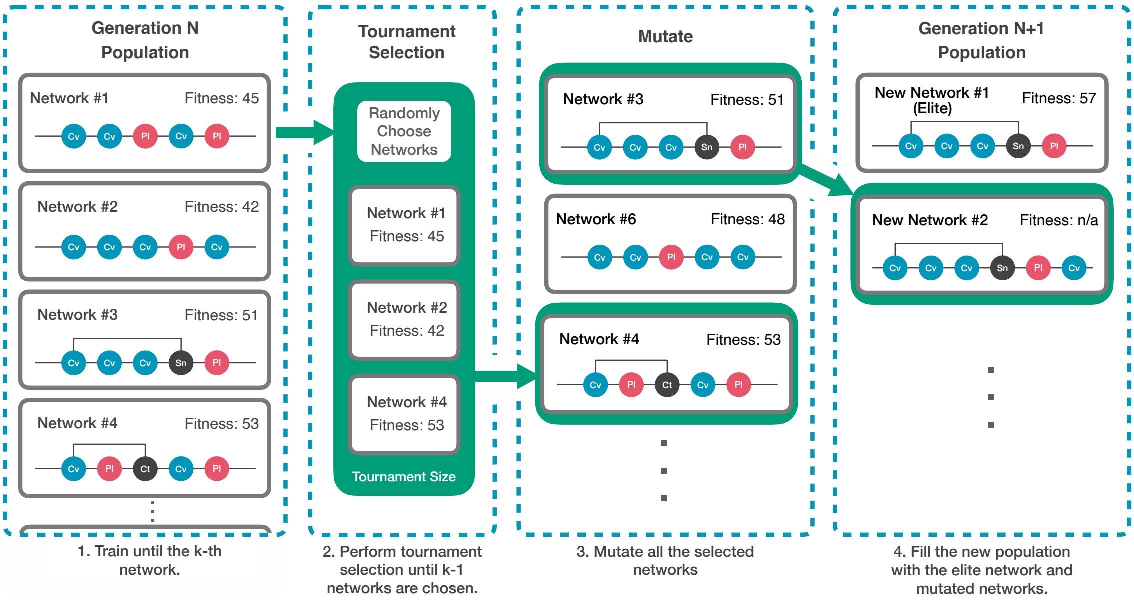


Figure 4.2: Generic neuroevolution process. The process starts by training all the networks in the population and obtaining their fitness (i.e., accuracy). Subsequently, networks are randomly selected with the tournament size (e.g., three networks). The fittest network in the tournament will fill a slot in the mutation area, which has a total of (population size minus one) slots. The tournament continues until all the slots are filled. Then, the networks in the mutation area are mutated one by one and become part of the new population for the next generation. The last new population slot for the next generation is filled by the fittest individual from the previous generation without mutation.

4.1.1 Generic Neuroevolution

In the generic neuroevolution, the networks, which form the population in Step 1, are all initialized to have a basic network architecture as shown in Fig. 4.1. The intuition behind this is to let the networks grow into complicated structures that suit the dataset instead of starting from unoptimized complicated structures (Stanley and Miikkulainen, 2002). Moreover, this setting saves computational costs by slowly growing the network architectures. In Step 2, all the networks are trained using the standard CNN training procedure. Subsequently in Step 3, each network in the population is evaluated with a fitness function, which is the network accuracy on the test set. The selection in Step 4 is specific to generic neuroevolution. After assigning the fitness to each network, all the networks in the population are replaced by networks using *tournament selection* except

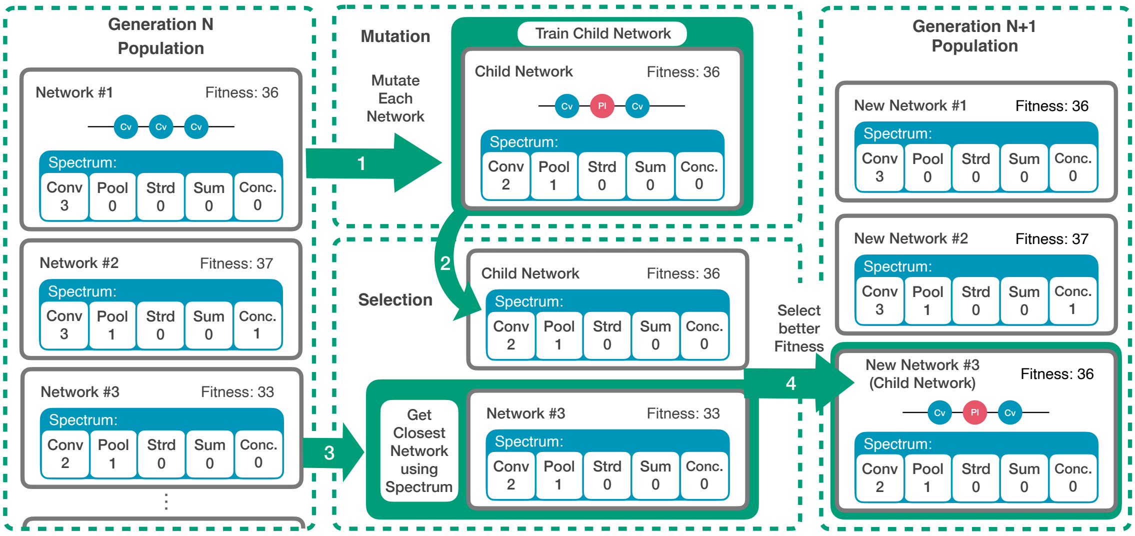


Figure 4.3: Steady-state neuroevolution process. The process starts by training all the individuals in the population and obtaining their accuracy. Next, each network in the population will produce a child network through mutation, and every child is trained to obtain its fitness. Each child is compared to the architecturally closest network in the population using the spectrum. If the child has better fitness, it replaces the architecturally closest network in the population. After all the children networks are evaluated, the process repeats from the first network producing a child network, which implies a new generation.

for the last one. In the tournament selection, a subset of the population is obtained and the best network individual in the subset fills the network population slot as explained in Section 2.6.1. This procedure repeats to fill the new population until the penultimate network slot. Afterward, these networks are mutated as part of the evolution process. For the last network slot in the new population, the best network also known as the *elite* in the previous population is carried over to the new population without any mutation, which is also called *elitist selection*. This ensures that the performance of the new population does not decrease on the next generation. The process from Step 2 is repeated until the last generation. The generic neuroevolution implementation is shown in Fig. 4.2.

4.1.2 Steady-state Neuroevolution

Steady-state neuroevolution is inspired by the steady-state genetic algorithm (Agapie and Wright, 2014; Vargas and Murata, 2016). Generic neuroevolution is a good approach when the number of network individuals is large enough to be able to find potential best architectures in a large search space. However, since CNNs have inherently deep architectures to effectively learn the dataset features, training each network in the population repeatedly every generation is computationally expensive and time-consuming.

As a solution to this issue, steady-state neuroevolution is proposed. In Step 1, instead of initializing the networks with the basic network architecture as in generic neuroevolution, they are initialized to have diverse architectures using the spectrum-based niching method (Vargas and Murata, 2016). Each network architecture is represented by an array called *spectrum*. The spectrum consists of the network properties namely: number of convolutional blocks, number of pooling blocks, number of strided convolutional blocks, number of summation blocks, and number of concatenation blocks. The distance between two networks is measured by calculating the Euclidean distance between their respective spectrum. The distance between networks N_1, N_2, is computed as

$$d_{\text{spec}}(N_1, N_2) = \|\text{spec}(N_1) - \text{spec}(N_2)\|, \tag{4.1}$$

where $\text{spec}(N_i)$ denotes the spectrum of network N_i and $\| \cdot \|$ is the Euclidean norm. Next, in step 2, the networks in the population are trained with the standard CNN training. Afterward, the process changes from the typical neuroevolution implementation. Each network in the population produces two to three children networks by aggressive mutation (i.e., multiple mutations on the network). The children are also trained and their fitness (i.e., test set accuracy) is compared to the closest network in the population using the spectrum. If a child network has better fitness than its closest network, the child replaces it in the population. The process from networks producing children to children replacing individuals in the population is repeated through all the generations. A basic implementation of the steady-state neuroevolution is shown in Fig. 4.3.

Contrary to generic neuroevolution, steady-state neuroevolution does not rely on the population size to encourage diversity among the networks and find the best architectures. Instead, the combination of diverse population initialization, aggressive mutation, and individual network replacement when it is outperformed, allows steady-state neuroevolution to be effective even on a small population (e.g., steady-state and generic neuroevolution using four and twelve networks respectively). If diverse population initialization and aggressive mutation are applied in generic neuroevolution with a small population, the networks will simply converge into similar architectures due to population replacement every generation and fail to evolve into good network architectures. Moreover, the networks in the steady-state neuroevolution are trained only once unlike in the generic neuroevolution where the whole population is trained. Finally, in steady-state neuroevolution, the individuals in the population are ensured to have better accuracy along with the generations.

The mutations in both neuroevolution configurations are done by adding, modifying, or deleting a convolutional block, pooling block, or strided convolutional block. In addition, the summation block, which sums the output channels of two prior blocks, and the concatenation block, which concatenates the output channels of two prior blocks are added as skip connection blocks. In the summation block, if the output channels of two prior blocks do not match, the smaller output channel is padded with zeroes to match the larger output channel before adding them together.

4.2 Learning from trained Networks Methods

Using the learning of a trained network is beneficial for improving the training and performance of another network. One of the well-known techniques that uses this concept is *transfer learning*. The key idea in transfer learning is to reuse the weights or features extracted from a dataset to another dataset in order to reduce training time and compensate for the features provided by a small dataset. Transfer learning is effective in CNNs because CNNs share the same low-level feature extractor in their first few layers. Transfer learning makes standard networks (e.g., ResNet), even with very deep architectures useful to train with different datasets. However, transfer learning is not directly accessible to neuroevolution-produced networks due to the complexity of their architectures. Therefore, two transfer learning methods are modified to apply to neuroevolution-produced networks. In addition, another technique that uses the learning of a trained network, which is called *knowledge distillation*, is also applied to neuroevolution-produced networks. In knowledge distillation, instead of using the weights of a trained network, the response of a trained network (e.g., logits) to an input is used to guide the learning of the target network. With this method, the learning of a trained network can be transferred to a network even with a different architecture.

4.2.1 Transfer Learning

There are two types of transfer learning methods incorporated to neuroevolution training. The first type is fine-tuning, where the weights of a network trained on a different dataset (e.g., ImageNet dataset) are used as initial weights of the target network.

Next, the target network is evolved using neuroevolution to further optimize its architecture to the dataset in addition to the weights optimization of the training process. The second type of transfer learning is feature extraction. In feature extraction, the weights transferred to the target network are frozen. Subsequently, the network is evolved using neuroevolution while keeping the weights from the trained network the same through the generations. Neuroevolution essentially adds blocks that are necessary to improve the performance of the target network.

4.2.2 Knowledge Distillation

In knowledge distillation, as discussed in Section 2.5.2, the learning of a trained network or teacher network is shared to another network or student network through their logits (Hinton et al., 2015). Both networks are fed with the same image and the logits (i.e., the output of the last layer) of each network is fed to the modified softmax. The modified softmax function has a parameter called *temperature* denoted as T, which regulates the importance of each logits. The modified softmax is formally defined as

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}, \tag{4.2}$$

where z_i denotes the logits of a network at i-th class, p_i denotes the probability of an input belonging to i-th class, and T as the temperature. When $T \to \infty$, the probability between the classes becomes the same. However, when $T \to 0$, the probability between the classes becomes a one-hot label, that is the class with the highest probability has a value of 1 while other classes become zero. Since in knowledge distillation, the class probability assignment of the teacher network is to be taught to the student network, the T should properly be adjusted. Subsequently, the distillation loss, which encourages the student network to match its logits with the teacher network, is defined as

$$L_D(p(z_t, T), p(z_s, T)) = \sum_i -p_i(z_{ti}, T)\log(p_i(z_{si}, T)), \tag{4.3}$$

where z_t and z_s are the logits of the teacher network and student network respectively. The distillation loss is added to the cross-entropy loss of the student network, which makes the total loss defined as

$$L = L_{CE} + L_D. \tag{4.4}$$

The combination of the two losses encourages the student network to optimize its parameters not only to output correct image labels but also output logits similar to that of the teacher network. This method is tested with neuroevolution-produced networks to evaluate whether a trained network can help improve the performance of neuroevolution-produced networks.

4.3 Neureovolution Applications

To concretely demonstrate the effectiveness of neuroevolution in discovering optimal network architectures, it is applied to two specific problems in deep neural networks. First, neuroevolution is used to find good architectures for detecting dangerous objects in X-ray images. Second, neuroevolution is tasked to find network architectures that are robust to the transferability of adversarial examples.

4.3.1 X-ray Object Detection

In this application, a CNN is fed with X-ray images that contain dangerous objects. The aim of CNN is to perform object detection, which is creating a bounding box around dangerous objects while correctly labeling them in the process. In contrast to image classification, object detection is more complicated because it combines localization of objects and image classification. In addition, applications that use specialized images such as X-ray datasets as shown in Fig. 2.37, tend to have a limited number of images. This is due to the difficulty in data collection and scarcity of resources (e.g., cancer tumors). In the case of dangerous objects X-rays, companies are reluctant to divulge such datasets publicly over the risk of jeopardizing security. Thus, tackling this issue is a good measure for the capability of neuroevolution.

To combine object localization and image classification in CNNs, there are specialized network architectures and training evaluations to effectively extract the necessary information from the dataset. One of the most successful implementations of object detection in CNN is the YOLO algorithm (Redmon and Farhadi, 2018). The YOLOv3 has a network architecture called Darknet-53 as seen in Fig. 4.4. It is an iteration from its previous version that utilizes skip connections to create a deep network. The YOLOv3 has 53 convolutional blocks as opposed to 19 from the previous version. The improvement solves different problems including the ability to detect small objects.

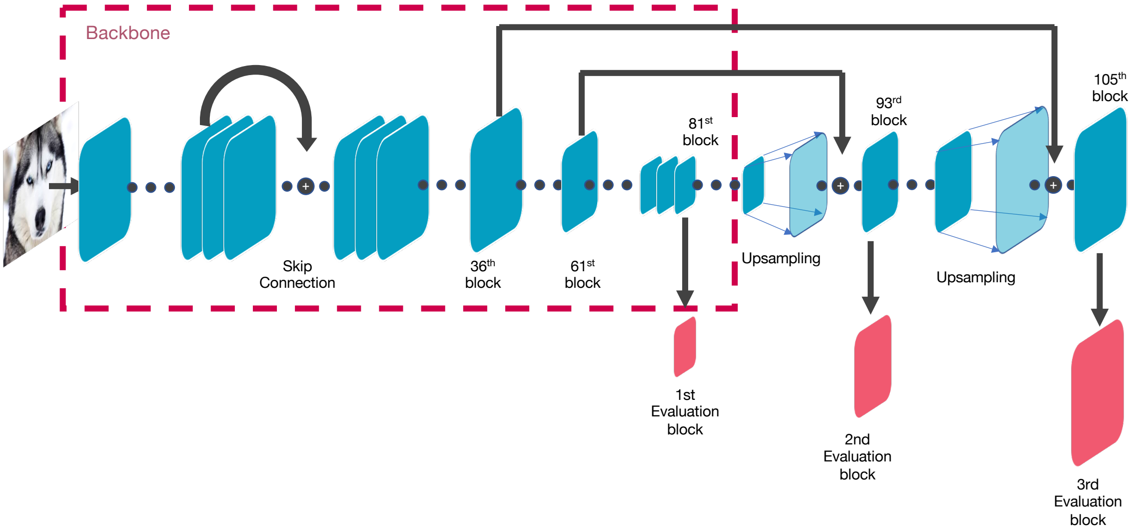


Figure 4.4: YOLOv3 Network Architecture.

Applying a deep YOLOv3 network to a specialized application such as dangerous objects X-ray detection, which has limited images, may not be optimal. It is because a limited dataset does not need a deep network architecture. The YOLOv3 architecture can be divided into two sections as seen in Fig 4.4. The first section, which constitutes the bulk of the architecture, is similar to the architecture of ResNet with skip connections every few convolutional blocks. This section is denoted as *backbone*. The second section, which is comprised of evaluation blocks, upsampling blocks, and long skip connections from the backbone is denoted as *evaluation*. The evaluation is responsible for calculating the loss of input images. Since the backbone has numerous convolutional blocks, neuroevolution is proposed to create a substitute architecture that maintains or improves the performance of the original backbone while reducing the number of blocks and optimizing the architecture to the specific dataset (i.e., dangerous objects X-ray).

The neuroevolution configuration used in this application is the generic neuroevolution. Under this configuration, there are two methods proposed to create a new backbone for the YOLOv3. The first method uses a different dataset and application (i.e., CIFAR-10 dataset and image classification) to create a network using neuroevolution, and then, the produced network replaces the YOLOv3 backbone. The second method, on the other hand, replaces the backbone of YOLOv3 with a basic network and uses multiple copies of it to form a neuroevolution population. The population of YOLOv3 networks with a basic backbone undergoes neuroevolution where the backbone mutates every generation

Algorithm 2 Algorithm for NECNN-C10

```
Input: $b$: basic network, $s$: size of population, $l$: max network block count, $t$: training length
Output: $F$: output network
$P = \{N_1, N_2, ..., N_s\}$    ▷Initialize the all the networks with the network architecture $b$
for $i = 1, 2, ..., (l{\cdot}t)$ do
    for each $N$ in $P$ do
        $N$ = Train($N$)
    end for
    if $i \bmod t = 0$ then
        for $j = 1, 2, ..., s$ do
            $C_j$ = Evaluate($N_j$)    ▷Evaluate the fitness of each network, which is the accuracy
        end for
        for $j = 1, 2, ..., s$ do
            $M_j$ = TournamentSelection($C, P$)
        end for
        for $j = 1, 2, ..., s$ do
            $N_j$ = Mutation($M_j$)
        end for
    end if
end for
for $j = 1, 2, ..., s$ do
    $C_j$ = Evaluate($N_j$)
end for
return SelectBest($C, P$)    ▷Select the best network using the fitness
```

to optimize the architecture to the dataset. The first and second methods are summarized in Algorithms 2 and 3 respectively.

In Algorithm 2, the neuroevolution population is initialized with s number of individual networks. All the individual networks have the same basic architecture b. The networks are trained with the CIFAR-10 dataset for t epochs and the networks are evaluated for fitness, which is the image classification accuracy. Afterward, $s-1$ networks are selected using tournament selection. The selected individuals are mutated and replaces the current individuals in the next generation of population. For the last individual slot in the population, the elite individual from the current population is copied to the next generation of the population. Then, the cycle from the training of networks in the population repeats l times. After the evolution has finished, the best individual network F in the population is returned and becomes the YOLOv3 backbone.

Conversely, in Algorithm 3, neuroevolution directly evolves a population of YOLOv3 networks with a basic backbone. During initialization, the population is filled with s number of the same YOLOv3 network with a basic backbone. Each individual in the population is trained with the dangerous objects X-ray dataset for t epochs and subse-

Algorithm 3 Algorithm for NECNN-XR

Input: b: YOLOv3 with basic backbone, s: size of population, l: max network block count, t: training length, e: refine epoch count

Output: F: output network

```
$P = \{N_1, N_2, ..., N_s\}$    ▷Initialize the all the networks with the network architecture $b$
for $i = 1, 2, ..., (l \cdot t)$ do
    for each $N$ in $P$ do
        $N$ = Train($N$)
    end for
    if $i$ mod $t = 0$ then
        for $j = 1, 2, ..., s$ do
            $C_j$ = Evaluate($N_j$)    ▷Evaluate the fitness of each network, which is the accuracy
        end for
        $M$=SelectBest($C, P$)    ▷Select the best network using the fitness
        for $j = 1, 2, ..., s$ do
            if $j = 1$ then
                $N_j = M$
            end if
            $N_j$ = Mutate($M$)
        end for
    end if
end for
for $j = 1, 2, ..., s$ do
    $C_j$ = Evaluate($N_j$)
end for
$K$=SelectBest($C, P$)
for $i = 1, 2, ..., e$ do
    $K$ = Train($K$)    ▷Refining the best network
end for
return $K$
```

quently evaluated for fitness. Since the application is object detection, the fitness is the mean average precision of the network. In contrast to Algorithm 2, only the best network from the current population is mutated every time to fill the $s - 1$ population slots for the next generation. The reason behind this is the considerably high computational cost of training a YOLOv3 network compared to a regular CNN for image classification. The final population slot for the next generation is again filled by the best network from the current network without mutations. After the evolution, the best network is obtained from the population and trained further for e epochs. The refined network is returned as F.

4.3.2 Transferability of Adversarial Examples Defense

One of the most intriguing problems in deep neural networks is adversarial attacks as discussed in Section 2.3.4. Adversarial attacks can fool the network into misclassifying images by slightly perturbing the image pixels, which are not visible to the human eye.

Furthermore, an attacked image, known as *adversarial example*, designed for a specific network can astonishingly fool another network without additional perturbations. To combat the transferability of adversarial examples, most researches focus on improving the network training process (e.g., adversarial training) or filtering the input images (e.g., JPEG compression) (Guo et al., 2018; Shafahi et al., 2019; Wong et al., 2019). However, there are recent and limited researches that try to find networks that are robust to adversarial examples from the architecture perspective. Thus, as a contribution to the literature, a method to use neuroevolution to find robust network architectures is proposed.

Although most standard networks (e.g., ResNet, DenseNet, etc.) share transferability of adversarial examples between them, it has been empirically observed that the extent of transferability depends on the similarity of the network architecture. The more architectural similarities two networks share, the higher the transferability of adversarial examples is shared between them. Therefore, neuroevolution is proposed to find network architectures that can resist transferability. However, using neuroevolution alone to find such networks requires complex mutation procedures and a large number of generations (Kotyan and Vargas, 2020). To make the architecture search efficient, a technique called gradient misalignment (GM) is combined with neuroevolution. The networks that share high transferability between them also exhibit aligned input gradients (Liu et al., 2016; Demontis et al., 2019). The gradient misalignment technique aligns the direction of input gradients of a network opposite to those of the network where the adversarial examples are generated. Using this technique alone can make a network robust but combining it with neuroevolution produces a significantly more robust network as demonstrated in the experiments. Moreover, GM allows neuroevolution to use simpler network mutations and a smaller number of generations than the previous study. Formally, neuroevolution with GM is proposed to find networks that are robust to the transferability of adversarial examples for this application.

The neuroevolution configuration utilized in this application is the steady-state neuroevolution. As such, it follows the neuroevolution process described in Section 4.1.2. The main differences in the neuroevolution implementation of this application are the training procedure and the fitness function. As mentioned before, GM is combined with neuroevolution, and it is done by adding the GM loss to the cross-entropy loss of the network being trained. To calculate the GM loss, the network that generates the adversarial examples, denoted as reference network, is employed. During the training of a network, the images fed to the network are also fed to the reference network. Afterward, the cosine similarity of the input gradients of the network being trained and the reference network are obtained as the GM loss $\mathcal{L}_{\text{GM}}$, which measures how much the input gradients are misaligned. Formally, given a dataset $\mathcal{D}$, the average cosine similarity of networks f_c and f_r are calculated as follows:

$$\mathcal{L}_{\text{GM}} = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \frac{\langle \nabla_x \ell_{f_c}(x,y), \nabla_x \ell_{f_r}(x,y) \rangle}{\|\nabla_x \ell_{f_c}(x,y)\|, \|\nabla_x \ell_{f_r}(x,y)\|}, \tag{4.5}$$

where $|\mathcal{D}|$ denotes the size of dataset, $\langle \cdot, \cdot \rangle$ denotes the inner product of vectors, and $\| \cdot \|$ is the Euclidean norm. The total network loss of the network being trained becomes

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{GM}}, \tag{4.6}$$

where λ is a hyperparameter. The λ hyperparameter is adjusted carefully according to the characteristics of a dataset. If λ is too large, the network does not learn the correct image labels because it only focuses on the misalignment of input gradients. In contrast, if the λ is too small, the network fails to misalign its input gradients with respect to the reference network. With the GM loss added to the total loss of a network, the network is encouraged to have misaligned input gradients without explicitly interfering with the network learning process and weights. In this way, the network weights can converge into values that learn the correct labels of images while having misaligned input gradients with respect to the reference network. The networks trained with the GM loss are the individuals in the neuroevolution population and their children.

Since the task of neuroevolution is to find networks that are robust to transferability of adversarial attacks while maintaining good accuracy on unperturbed images, the accuracy on unperturbed images (i.e., clean accuracy) and accuracy on adversarial examples (i.e., adversarial accuracy) are both used to evaluate the fitness of a network. The fitness of a network is defined as the minimum between clean accuracy and adversarial accuracy.

$$\mathcal{F} = \min(\mathcal{A}_{\text{CL}}, \mathcal{A}_{\text{AD}}), \tag{4.7}$$

Algorithm 4 Neuroevolution with GM

```
Input: $s$: size of population, $n$: number of children to produce, $d$: minimum network distance,
    $g$: number of generations
Output: $F$: evolved networks after neuroevolution
$P = \{N_1, N_2, ..., N_s\}$    ▷ Initialize the population by $s$ candidate networks $N_1, \ldots, N_s$ with
    minimum network distance $d$.
for $t = 1, 2, \ldots, g$ do
    for each $N$ in $P$ do
        $\{C_1, C_2, \ldots, C_n\} = \text{mutation}(N)$
        for $j = 1, 2, \ldots, n$ do
            $N_j = \text{train}(C_j)$    ▷ Train with loss Eq. (4.6).
            $N^* = \arg\min_{N' \in P} d_{\text{spec}}(N', C_j)$    ▷ Find the closest network
            if $\mathcal{F}(N^*) < \mathcal{F}(C_j)$ then
                Replace $N^*$ with $C_j$.
            end if
        end for
    end for
end for
return $P$
```

where $\mathcal{A}_{\text{CL}}$ is the clean accuracy and $\mathcal{A}_{\text{AD}}$ is the adversarial accuracy. By using the minimum between the clean and adversarial accuracies, the lower bound of both accuracies will always increase through the course of evolution. If the fitness is not designed properly, the neuroevolution tends to prioritize one accuracy over the other. However, the proposed fitness function gives priority to both values so that the network does not reduce the transferability of adversarial examples at the cost of clean accuracy.

The neuroevolution with GM framework is summarized in Algorithm 4. Given a population s, number of children n, minimum network distance d, and number of generations g, the process starts by initializing a population of s networks with a minimum network distance of d to each other. Each network in the population produces n children through aggressive mutation. Subsequently, each child is trained with GM and compared to the network in the population P with the least network distance. If there are multiple networks with the least network distance, the first closest network is used. However, the probability that this will occur is low due to the combination of aggressive mutation and spectrum-based niching. During the comparison, if the child has better fitness than the closest network, the child replaces the closest network in the population. The cycle continues until the last generation. At the end of the process, the population of evolved networks is returned as output F. The evolved networks in F are trained further to refine the weights.

4.4 Summary

In summary, this chapter presents the different neuroevolution approach to CNN techniques, which are generic neuroevolution and steady-state neuroevolution. Moreover, the methods that use a trained network to help improve the training of another network such as transfer learning are discussed. The techniques to apply these methods to neuroevolution-produced networks are proposed. Finally, the proposed methods to use neuroevolution to discover optimal network architectures specifically for dangerous objects X-ray detection and transferability of adversarial examples defense applications are also explained in this chapter.

CHAPTER V

NEUROEVOLUTION FOR CNN EXPERIMENTS

In the neuroevolution for CNN experiments, the methods detailed in Chapter 4 are demonstrated. The experiments are divided into two parts. The first part, *neuroevolution techniques*, examines the different implementations of neuroevolution, which is discussed in Section 5.1. The second part, *Learning from trained networks*, explores the effects of adding transfer learning, and knowledge distillation to neuroevolution, which is examined in Section 5.2.

5.1 Neuroevolution Techniques

Neuroevolution is an algorithm used to evolve network architectures to optimize them for a particular dataset or application. There are two configurations of neuroevolution (NE) that are experimented with in this section. The first configuration is generic neuroevolution, and the second is steady-state neuroevolution, which are both discussed in Sections 4.1.1 and 4.1.2 respectively. The neuroevolution-produced CNN, denoted as NECNN, are generated with the following experiment hyperparameters. In the first configuration, the population is initialized to have 10 networks, each with the basic architecture as shown in Fig. 4.1. The selection method is tournament selection with a tournament size of three. The maximum block (e.g., convolutional block) count for each network is 30 blocks, excluding the fully connected layers. The batch size is fixed at 128, and the optimizer is stochastic gradient descent (SGD). In the second configuration, the population count is four networks that are randomly designed to have at least a network distance of 4 to each other. Each network in the population produces two children every generation. The total number of generations is 30, and the maximum block count for each network is 30. Similar to the first configuration, the batch is fixed to 128, and the optimizer is SGD. For the other hyperparameters such as learning rate, they are fixed to the default PyTorch DNN library (Paszke et al., 2019) settings without fine-tunings to establish an objective comparison between the networks. For each configuration, the best network produced is trained further for 500 epochs from scratch to ensure that the accuracies converge into stable values. The hyperparameter settings for each configuration

Table 5.1: Neuroevolution hyperparameter settings.

	Population	Network Init.	Mutations per Gen.	Generations
Generic NE	10	basic	9 networks	25
Steady-state NE	4	random	8 networks	30
	Max Blocks	Batch Size	Optimizer	Fine-tuning Epoch
Generic NE	30	128	SGD	500
Steady-state NE	30	128	SGD	500

Table 5.2: The NECNN networks evolved using the generic neuroevolution and steady-state neuroevolution are compared. Furthermore, the ResNet-18 and ResNet-34 are also used as baselines. Learning from a trained network is also employed by using transfer learning and knowledge distillation on the networks. The NECNN networks have performed better than the ResNet networks on all comparisons.

	ResNet and NECNN Networks			
	ResNet-18	ResNet-34	Generic NE	Steady-state NE
Accuracy	89.21%	89.25%	**91.59%**	91.00%
	ResNet Networks with Transfer Learning			
	ResNet-18+TL		ResNet-34+TL	
Accuracy	88.78%		**89.87%**	
	ResNet and NECNN Networks with Knowledge Distillation			
	ResNet-18+KD	ResNet-34+KD	Generic NE+KD	Steady-state NE+KD
Accuracy	90.99%	91.90%	**92.87%**	92.38%

are summarized in Table 5.1.

The network architectures produced by the generic NE and steady-state NE are compared to the ResNet-18 and ResNet-34. The ResNet networks are trained with the same settings as the NECNN networks. In terms of block count, the NECNN is about the same as the ResNet-18 when all the blocks (e.g., convolutional, skip connection, etc.) are counted. Moreover, since the use of transfer learning is a common practice in the training of standard networks such as ResNet, it is also employed as a comparison. For the transfer learning settings, the trained network weights are obtained from training the ResNet on the ImageNet dataset. The trained weights are used as pre-training or initialization to the ResNet networks. In line with learning from a trained network, knowledge distillation, as discussed in Section 4.2.2 is also applied on NECNN and ResNet networks. In knowledge distillation, the teacher network utilized is refined in CIFAR-10 to have an accuracy of 94.38%.

As reported in Table 5.2, the NECNN networks from both NE configurations outperform the ResNet networks. In particular, despite having fewer blocks than ResNet-34, the optimized NECNN network architectures have higher accuracies than ResNet-34. Training the ResNet-34 with transfer learning slightly improves its performance but it is still lower than NECNN networks. The knowledge distillation improves both the ResNet and NECNN network accuracies and the increase is roughly the same for all the networks when compared to their vanilla versions. Thus, the NECNN networks have the best performance in all comparisons. The results demonstrate that the two configurations of NE are capable of producing network architectures that can perform at the same level as standard networks such as ResNet but with an optimized architecture.

The network architectures produced by generic NE and steady-state NE are shown in Fig. 5.1 and 5.2. The NECNN from generic NE has a total of 26 blocks, which is comprised of 17 convolutional blocks, 2 pooling blocks, 5 skip connection blocks, and 2 fully connected layers. On the other hand, the NECNN from steady-state NE has a total of 32 blocks, which consist of 9 convolution blocks, 7 pooling blocks, 14 skip connection blocks, and 2 fully connected layers. The NECNN from generic NE is evolved for 25 hours, whereas the steady-state NECNN is evolved for 71 hours. These evolution periods are substantially lower than CGP-CNN, in which the shortest evolution period took 13.7 days (Suganuma et al., 2017). In addition, the highest accuracy for the CGP-CNN network, which is 94.02%, is only higher than NECNN network by 1.15%. Moreover, compared to generic NECNN, the steady-state NECNN has higher architectural complexity. This is attributed to the aggressive mutation and spectrum-based niching applied to the population networks. The aggressive mutation creates sophisticated block combinations through repeated mutations, whereas spectrum-based niching allows very different architectures to survive along with the generations. In contrast, generic NE uses only one mutation every generation. Furthermore, it also uses tournament selection in a relatively small population (i.e., 10 individuals) that lets a particular block combination prevail among the individuals and survive along with the generations. For these reasons, the network architecture produced by generic NE tends to be relatively simpler than steady-state

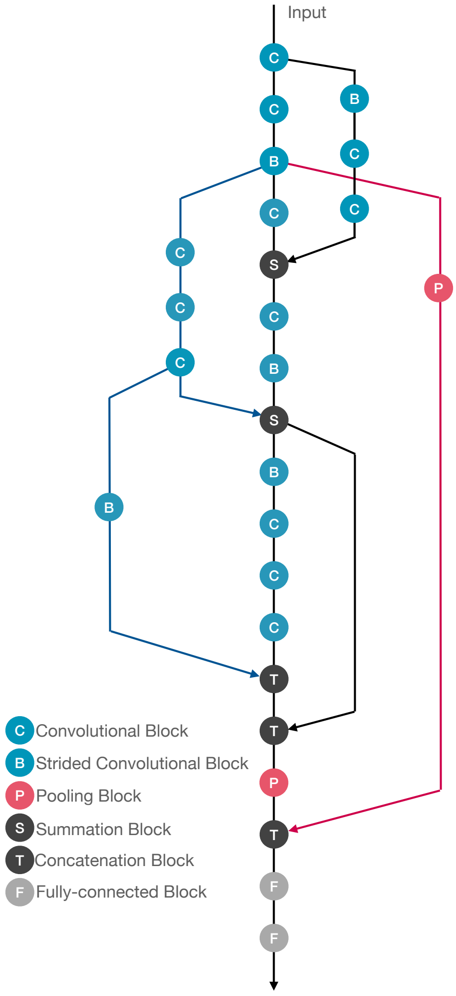


Figure 5.1: Generic NE produced network.

network architecture but more complicated than standard networks (e.g., VGG).

5.2 Learning from trained Networks

Utilizing the learning of a trained network is considered one of the good practices in training deep neural networks. Standard networks (e.g., ResNet, VGG, etc.) are commonly trained on large-scale datasets such as ImageNet. The network learning or weights acquired from these trainings can be utilized to train the network on new applications such as CIFAR-10, X-ray datasets, etc. This method is called transfer learning as discussed in Section 2.5.1. The first few layers of a network, regardless of the dataset, operate in the same way as a low-level feature extractor. As a result, the weights on these layers can be used directly as weights initialization or pre-training to train the network on a new dataset. Furthermore, if the new dataset is similar to the dataset the network weights are

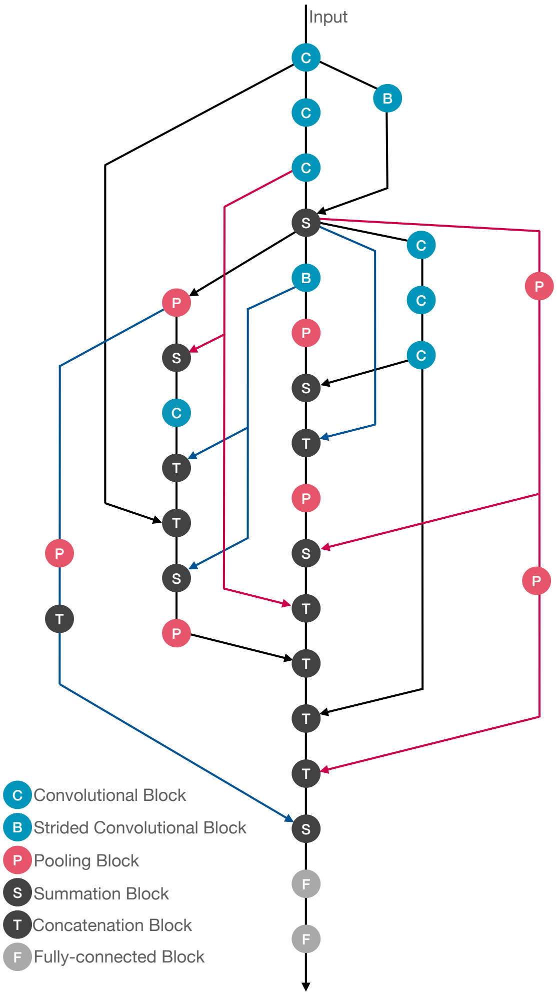


Figure 5.2: Steady-state NE produced network.

trained with, these weights of the network can be entirely frozen except for the last layer to train on the new dataset, which is called feature extraction. Implementing transfer learning can accelerate network convergence and reduce training time. In this experiment, the ways to incorporate transfer learning to NE are explored. Generally, transfer learning cannot be used in NE because the network architecture changes every generation in NE. The weights of a standard network (e.g., ResNet) simply cannot fit those of the NECNN network. Thus, different methods are experimented with.

In addition, another technique, which uses the learning of a trained network to teach another network, called knowledge distillation is employed. In knowledge distillation as discussed in Section 2.5.2, instead of using the weights of the trained network, the network is used as a teacher to help another network or student network converge

Table 5.3: The baseline networks are ResNet-18 recreated using self-implemented blocks (NB), NECNN with 40 blocks, and ResNet-18, which is evolved using steady-state NE to have 40 blocks.

	Baseline Networks		
	ResNet-18(NB)	NECNN-40	ResNet-18(NB)+NE
Accuracy	86.73%	90.28%	91.48%

to better weights. The logits or the values in the last layer of a teacher network reflect the function it represents. By encouraging the student network to emulate the logits of the teacher network, the student network can effectively learn better weights and subsequently, perform better. In this section, knowledge distillation is also experimented to help networks learn, especially the NECNN networks, which cannot normally use transfer learning to learn from trained networks.

5.2.1 Network Baselines

For the baselines, there are three networks used, which are trained with the same settings as in Section 5.1. The first network is a ResNet-18 network but reimplemented using the blocks utilized in NE, which is denoted as ResNet-18(NB). There are slight differences between the original ResNet-18 and ResNet-18(NB) that contributed to the dip in the accuracy of ResNet-18(NB). However, since the blocks in NE are used, ResNet-18(NB) architecture can be evolved using NE. The ResNet-18(NB) has approximately 30 blocks. The second network is NECNN evolved with 40 blocks. Lastly, the third network is ResNet-18(NB), which is evolved using NE for 20 generations to gain 10 more blocks. The intuition of creating the last network is to refine the ResNet architecture using NE.

The accuracies of the baseline networks are shown in Table 5.3. The results show that using a successful hand-engineered architecture such as ResNet-18 with NE can evolve into a better network. Although NECNN-40 and ResNet-18(NB)+NE have the same block count, the latter performs better. The results also demonstrated that a good network architecture can be used as a starting architecture for NE. Instead of inheriting the weights to learn better as in transfer learning, the network architecture can be inherited to evolve better. The ResNet-18(NB)+NE network architecture can be seen in Fig. 5.3. After the evolution, ResNet-18(NB)+NE only maintains the ResNet structure for the first few

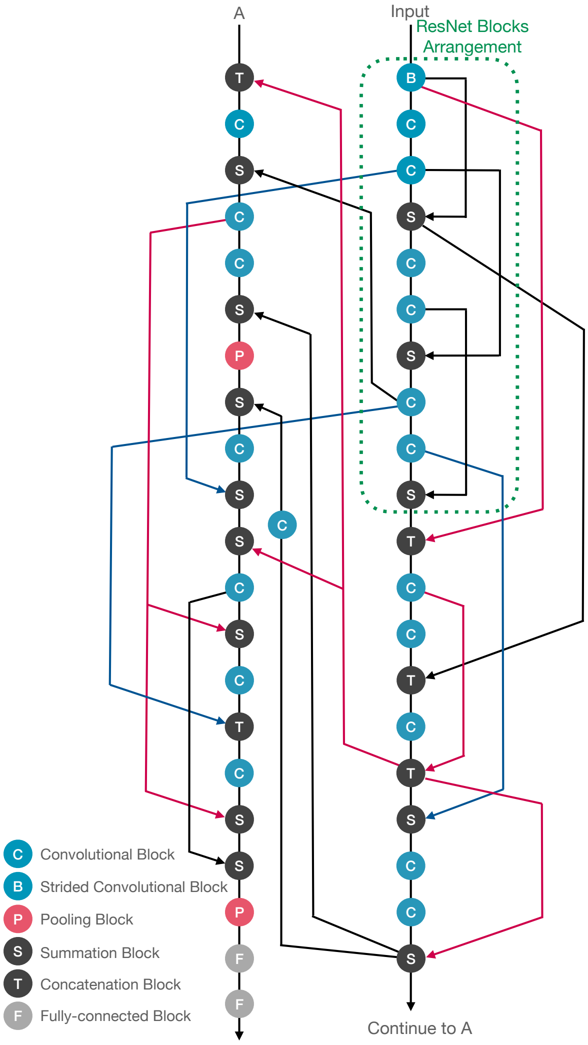


Figure 5.3: This network is ResNet-18(NB), which is evolved using NE to add 10 more blocks. The blocks inside the green dotted rectangle resembles the ResNet architecture.

blocks as shown inside the green dotted rectangle. In the entirety of the architecture, it can be observed that NE populated the network with abundant skip connection blocks linking different parts of the architecture.

5.2.2 NECNN Networks with Transfer Learning

To understand the effects of transfer learning on the evolution of networks using NE, two transfer learning configurations are experimented. Each configuration has training settings that are the same as in Section 5.1. The first configuration is the *fine-tuning*, in which the networks are initialized with the weights of the network trained on another dataset. In this experiment, the weights come from the ImageNet dataset training. Under

Table 5.4: The results of combining transfer learning with NE are shown below. Adding fine-tuning transfer learning to NE does not improve the accuracy, whereas adding feature extraction transfer learning to NE slightly improves the accuracy of the network compared to the baseline.

	Fine-tuning TL		Feature extraction TL	
	ResNet-18(NB)	ResNet-18(NB)+NE	ResNet-18(NB)	ResNet-18(NB)+NE
Accuracy	87.84%	87.78%	77.99%	88.38%

the fine-tuning, two networks are utilized. The first network is simply using the trained weights as initialization and then train normally. The second network is also initialized by the trained weights but undergoes evolution similar to the baseline network. The second transfer learning configuration is the *feature extraction*. It also uses two networks as in the first configuration but the main difference is the freezing of parameters. For the first network, after initializing it with the trained weights, the parameters are frozen except for the last layer. Similarly, the second network also has frozen weights but the blocks that are added during evolution can be trained. Furthermore, if an original block that has a frozen weight is replaced during evolution, the replacement block can be trained again. The results are shown in Table 5.4.

As expected, using the transfer learning as fine-tuning to ResNet-18(NB) yields better performance. Using the trained weights as initialization provides a good starting point for the network than learning from random weights. However, adding NE to the same network (ResNet-18(NB)+NE) does not improve the performance, which is also attributed to the trained weights. Since the ResNet-18(NB) can now converge better due to the trained weights, when using steady-state NE, the child networks cannot easily replace their parents. It results in many generations passing in NE without change in the population networks. In contrast to fine-tuning, the feature extraction transfer learning resulted in a decrease in performance for ResNet-18(NB). With only the final layer of the network being available to train, it shows that the weights in the final layer are not enough to shift the network learning from ImageNet to CIFAR-10. However, the difficulty in training the network with feature extraction is taken advantage of by NE. Since the networks in the NE population cannot converge easily, these can be simply replaced by their children networks. The better evolution process resulted in a network (ResNet-18(NB)+NE) that has the highest accuracy among the four networks. The experiments indicate that using transfer learning with NE can slightly improve performance but they are not very com-

Table 5.5: The networks from the baseline in Table 5.3 are retrained with knowledge distillation. The teacher network is refined with CIFAR-10, which has 94.38% accuracy. All of the networks have remarkably increased their performance.

	Knowledge Distillation		
	ResNet-18(NB)	NECNN-40	ResNet-18(NB)+NE
Accuracy	88.07%	91.79%	93.29%

patible with each other. If the ResNet-18(NB)+NE from feature extraction is compared to the ResNet-18(NB)+NE from the baseline, the difference is 3.1%, which shows that NE can perform better without transfer learning. Thus, a different method should be employed to learn from a trained network.

5.2.3 NECNN Networks with Knowledge Distillation

The last method to learn from a trained network is knowledge distillation. In contrast to transfer learning, which shares network learning through network weights, knowledge distillation shares network learning by teaching a student network to evaluate inputs similar to the teacher. As explained in Section 4.2.2, the logits of the teacher network and the student network are compared, and the difference becomes an additional loss to the student network to encourage the student to output logits similar to the teacher. Since only logits are being compared in knowledge distillation, it can be applied to NECNN networks easily. In the experiments, the networks in the baseline are retrained with knowledge distillation as student networks. The teacher network is the same network used in Section 5.1, which is refined in CIFAR-10 to have an accuracy of 94.38%. The results are reported in Table 5.5.

The results demonstrate that sharing the learning of the teacher network through knowledge distillation can notably increase the performance of networks. The ResNet-18(NB)+NE has remarkably achieved 93.29% accuracy, which has increased by 1.81% from the baseline result. Furthermore, the increase is approximately the same for the other two networks. Therefore, knowledge distillation is a powerful method to guide the networks to learn better weights than simply relying on the training dataset. In addition, knowledge distillation is a suitable technique to make the NECNN networks learn from a trained dataset since any network architecture can be used unlike in transfer learning.

5.3 Summary

In this chapter, the neuroevolution techniques, which are generic neuroevolution and steady-state neuroevolution, are experimented with. The results show that both techniques have produced network architectures that have remarkable performances in the CIFAR-10 dataset. Moreover, training the neuroevolution-produced networks with knowledge distillation further improves the network performances. In contrast, transfer learning does not provide substantial benefits to neuroevolution-produced network training.

CHAPTER VI

NEUROEVOLUTION FOR CNN APPLICATIONS

The neuroevolution for CNN applications are experiments in which neuroevolution is employed in real-world CNN problems. The first application is discussed in Section 6.1. In this application, neuroevolution is used to evolve small backbones (network architectures) that replace the deep backbone of YOLOv3. The small backbone is optimized to the dangerous objects X-ray dataset, which has limited images, unlike the large backbone of YOLOv3, which has redundant parameters. The second application is expounded in Section 6.2. This application uses neuroevolution to evolve robust networks that reduce the transferability of adversarial examples. Adversarial examples are images generated by adversarially attacking a network, and these images can also fool other networks without any modification. However, the extent of the transferability of adversarial examples (i.e., fooling other networks) varies with the architecture design. Therefore, neuroevolution is utilized to discover exceptional architectures that can lower the transferability of adversarial examples significantly.

6.1 NECNN for X-ray Object Detection

In this application, the YOLOv3 network is compared to the YOLOv3 network that has its backbone replaced with NECNN (i.e., neuroevolution-produced CNN) network. As shown in Fig. 4.4, the YOLOv3 backbone, which is enclosed with a dashed rectangle, has a deep architecture with 81 blocks and skip connections similar to ResNet. To examine the significance of the backbone to the whole YOLOv3 architecture, different backbones of YOLOv3 are created from the original YOLOv3 backbone by trimming it. The first network is YOLOv3 with transfer learning weights (i.e., weights pre-trained on ImageNet). The second YOLOv3 network is simply the original network with random initialization. The third network is called YOLO Med, which is YOLOv3 but has its backbone trimmed to match the block count of the NECNN backbones. The last network is YOLO Tiny, which is a network resembling YOLOv2. It does not utilize skip connections and contains only one evaluation block. For the NECNN backbones, there are three networks used in the experiment, which are all generated using the generic neuroevolu-

Table 6.1: List of YOLO networks and NECNN networks with their description.

Network	Description
YOLOv3+TL	YOLOv3 Arch. with Transfer Learning
YOLOv3	YOLOv3 Arch. with random initialization
YOLO Med	YOLOv3 Arch. with backbone significantly decreased
YOLO Tiny	Architecture that resembles YOLOv2
NECNN-C10	Backbone produced by Alg. 2 with YOLOv3 evaluation
NECNN-XR1	Backbone produced by Alg. 3 with YOLOv3 evaluation
NECNN-XR2	Backbone produced by Alg. 3 with YOLOv3 evaluation

Table 6.2: YOLO Networks and NECNN Networks Block Counts.

Network	Block Count
YOLOv3+TL	106 blocks
YOLOv3	106 blocks
YOLO Med	55 blocks
YOLO Tiny	23 blocks
NECNN-C10	49 blocks (24 backbone + 25 evaluation)
NECNN-XR1	47 blocks (22 backbone + 25 evaluation)
NECNN-XR2	50 blocks (25 backbone + 25 evaluation)

tion. The first one is NECNN-C10, which is evolved using Algorithm 2. The second and third networks are NECNN-XR1 and NECNN-XR2, which are evolved using the Algorithm 3 with slight modifications in the neuroevolution parameters. The summary of the networks experimented with is shown in Table 6.1.

The YOLOv3 has a total of 106 blocks, which consist of convolutional blocks, upsampling blocks, skip connections, etc. YOLO Med and YOLO Tiny have 55 blocks and 23 blocks respectively. Conversely, the NECNN-C10 backbone has 24 blocks that are comprised of convolutional blocks, pooling blocks, summation blocks, etc. When the NECNN-C10 is combined with the YOLOv3 *evaluation*, which has 25 blocks, the total architecture block count becomes 49 blocks. NECNN-XR1 and NECNN-XR2 have 47 and 50 blocks respectively. The summary of the network block counts is shown in Table 6.2.

Furthermore, the parameter size of each network, measured in megabytes (MB) is also compared. For the YOLO networks, YOLOv3 has the largest parameter size with 242 MB, followed by YOLO Med with 136 MB and YOLO Tiny with 34 MB. For the NECNN networks, NECNN-C10 has 46 MB on the backbone and 42 MB on the evaluation

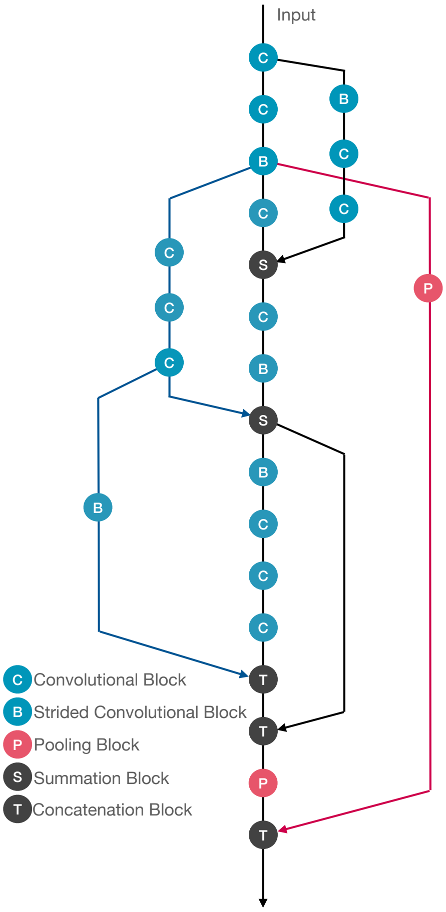


Figure 6.1: NECNN-C10 Network Architecture.

that is 88 MB in total. NECNN-XR1 and NECNN-XR2 have a total of 197 MB and 174 MB respectively. Notably, NECNN-C10 has a low parameter size despite having a similar block count with NECNN-XR networks. This is due to the evolution process in Algorithm 2, which allows more skip connections that have blocks in it as shown in Fig. 6.1. On the other hand, NECNN-XR networks produced using Algorithm 3 have a more constrained evolution because the networks are already connected to the YOLOv3 evaluation. The effects of the constraint on NECNN-XR networks resulted in fewer skip connections as shown in Fig. 6.2, which has preserved the parameter size on every block. The summary of the network parameter sizes is shown in Table 6.3.

The training parameters used for all the networks are the same to establish an objective comparison. The batch size is fixed to 32 and the input size is 128. The data augmentation utilized is horizontal flip only. To obtain the results, every network is trained for 1000 epochs multiple times and averaged. The metric used to measure the

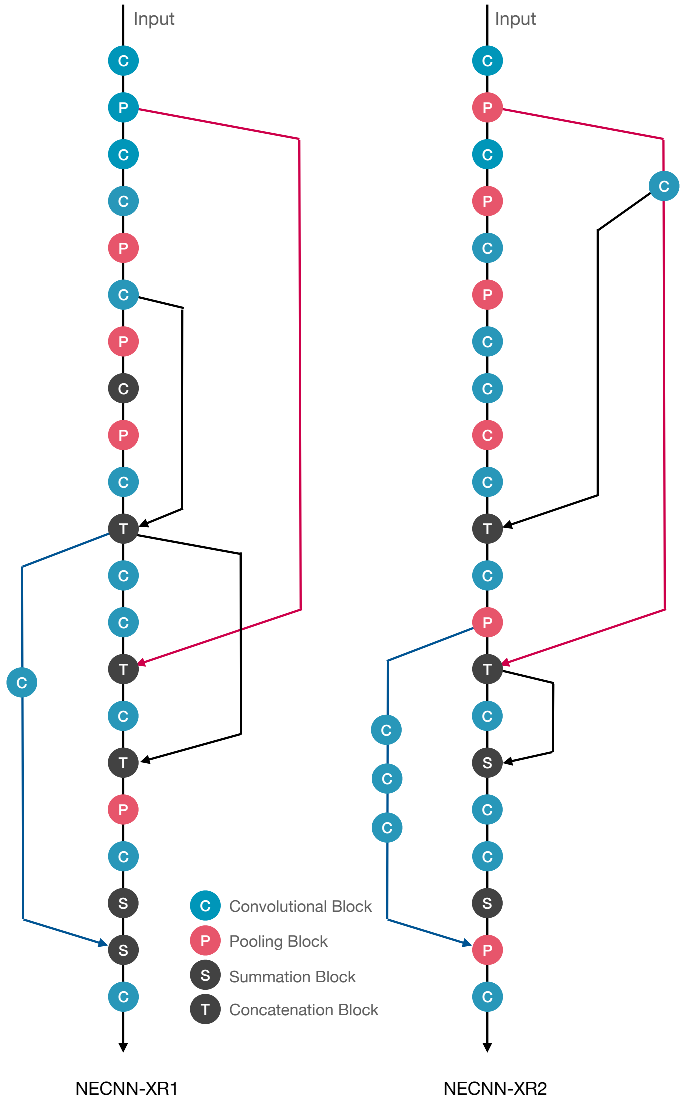


Figure 6.2: NECNN-XR1 and NECNN-XR2 Network Architectures.

performance of the networks is the mean average precision (mAP), which is the standard metric in object detection. The dataset employed is the dangerous objects X-ray, which has 662 training images and 442 testing images. Each image has an average size of 600 × 600 pixels. The example raw X- images are shown in Fig. 3.4.

The results of the training are shown in Table 6.4. YOLOv3 has an mAP of 52.9%. However, initializing YOLOv3 with transfer learning increases its mAP to 65.1%. YOLO Med and YOLO Tiny have low mAPs with 47.9% and 47.7% respectively. Moreover, NECNN-C10, NECNN-XR1, and NECNN-XR2 have decent mAPs with 60.8%, 63.8%, and 62.5% respectively.

Table 6.3: YOLO Networks and NECNN Networks Parameter Sizes.

Network	Parameters Size (MB)
YOLOv3+TL	242
YOLOv3	242
YOLO Med	136
YOLO Tiny	34
NECNN-C10	88 (46 backbone + 42 evaluation)
NECNN-XR1	197 (155 backbone + 42 evaluation)
NECNN-XR2	174 (132 backbone + 42 evaluation)

Table 6.4: The mAP of the YOLO networks and NECNN networks. YOLOv3+TL has the highest mAP among the YOLO networks, whereas NECNN-XR1 has the highest mAP among the NECNN networks.

Network	Accuracy (mAP)
YOLOv3+TL	**65.1%**
YOLOv3	52.9%
YOLO Med	47.9%
YOLO Tiny	47.7%
NECNN-C10	60.8%
NECNN-XR1	**63.8%**
NECNN-XR2	62.5%

The performance of the YOLO networks is within the expectations because when similarly structured networks (e.g., convolution-convolution and convolution-convolution-convolution networks) that mainly differ in depth are compared, the deeper network usually corresponds to better accuracy. However, depending on the size of the dataset, the deeper network may become a disadvantage. When the dataset size is limited, the deeper network cannot be properly trained as it tends to overfit the dataset and consequently fails to generalize. In this experiment, the object detection application is complicated enough to prevent the overfitting of YOLOv3. However, the backbone of YOLOv3 is still redundant as observed in the performance of small and optimized NECNN networks. The produced backbones (i.e., NECNN networks) are designed to have comparable performances with the unaltered YOLOv3. Unexpectedly, the NECNN networks have outperformed the YOLOv3 by at least 7%. It shows that the limited dataset cannot fully train the deep architecture of the YOLOv3. However, YOLOv3 with transfer learning weights has performed the best, and it is attributed to the weights that are refined using another dataset (i.e., ImageNet). The network layers or blocks in YOLOv3+TL are

Table 6.5: The mAP of the YOLOv3+TL and NECNN-C10 networks when trained on a bigger input size of 416 × 416 pixels.

Network	Accuracy (mAP)
YOLOv3+TL	86.6%
NECNN-C10	84.0%

trained feature extractors, which are refined by the limited X-ray dataset. In the case of YOLOv3, the limited X-ray dataset used to create its feature extractors cannot compete with the feature extractors of YOLOv3+TL.

The NECNN backbones have achieved better mAP than the YOLO networks with the exception of YOLOv3+TL. Despite having a lower number of blocks compared to YOLOv3, the NECNN networks utilize skip connections and multiple blocks in the skip connections to compensate for the lack of depth. Furthermore, the small backbone provided by the NECNN networks has made the training process easier than deep networks, especially with the limited dangerous objects X-ray dataset.

Finally, the results presented are in the range of ∼60% because of the small input size that is 128. If the input size is increased to 416 (default image size for YOLO), the accuracy increases into the range of ∼80% because the accuracy is directly proportional to the input size. However, for the sake of faster computation in this proof of concept, a smaller input size is utilized. Table 6.5 shows the mAP of the NECNN-C10 and YOLOv3+TL networks when trained using the input size of 416 × 416 pixels.

6.2 NECNN as Transferability of Adversarial Examples Defense

In this application, neuroevolution (NE) is utilized to evolve network architectures that are robust to the transferability of adversarial examples. Particularly, neuroevolution is combined with gradient misalignment (GM) to produce architectures that have high accuracy on unperturbed images (i.e., clean accuracy) and high robustness to adversarial examples (i.e., adversarial accuracy) generated by a reference network. Four networks are being compared, which are the following: (i) two NE+GM networks, which are the top two networks produced by the NE with GM method as summarized in Algorithm 4; a reference network, which is employed to generate the adversarial examples; and (iii) a reference network with GM, which is a network that has the same architecture as the reference network and is trained with GM. The networks in (ii) and (iii) are collectively denoted as the baseline networks.

To obtain the NE+GM networks, steady-state neuroevolution is employed. The NE population is initialized with four or five network individuals called candidate networks. The architecture of the candidate networks is randomly designed to have at least a network distance of four to each other. Each candidate network is trained for 20 epochs. During the training of the networks, the batch size is set to 128, and the remaining hyperparameters (e.g., learning rate) are set to the default PyTorch DNN library settings (Paszke et al., 2019) without fine-tuning. This is done to maintain an objective comparison between the NE+GM networks and the baseline networks. The NE evolves the networks for 50 generations, where each candidate network produces two children through aggressive mutation (i.e., multiple mutations on a network). Subsequently, the children are trained with GM for 20 epochs. Note that the number of epochs increases by five every 10 generations to account for the network complexity of the candidate networks after several generations. This implementation also helps lower the computational cost and time since the simple networks in the first few generations can converge with small epochs. At the end of the NE process, the top two networks from the final population are refined by training them for another 1000 epochs to ensure that the clean accuracy and adversarial accuracy converge to a stable value. Afterward, the refined networks are used as the NE+GM networks.

Two main experiments are explored in this application. The first experiment utilizes a full-dataset, and the second experiment utilizes a reduced-dataset. In the first experiment, the following datasets are used: CIFAR-10, MNIST, and KMNIST. In addition, ResNet-18, VGG, DenseNet, and SqueezeNet (Iandola et al., 2016) are used as network models. The baseline networks and the NE+GM networks are evaluated using the datasets and the adversarial examples generated by the reference network and the introduced network models. The evaluation metrics used to measure the performance of the baseline and NE+GM networks are the clean accuracy, which is the percentage of correctly classified images on a test set, and adversarial accuracy, which is the percentage of correctly classified images on an adversarially attacked test set. The first experiment demonstrates that the NE+GM networks have notably higher adversarial accuracies than

Table 6.6: The clean accuracy and adversarial accuracy of the baseline networks and NE+GM networks trained with a full dataset is shown here. In this table, the full CIFAR-10 dataset is employed. The NE+GM networks have remarkably higher adversarial accuracy (i.e., lower transferability of adversarial examples) than the reference network without GM (Ref. Network) and with GM (Ref. Network+GM).

	Ref. Net.	Ref. Net.+GM	NE+GM Net. 1	NE+GM Net. 2
Clean Acc.	78.40%	80.82%	**81.09%**	80.62%
Adv. Acc. (L_∞-PGD)	23.10%	42.90%	**54.12%**	53.27%

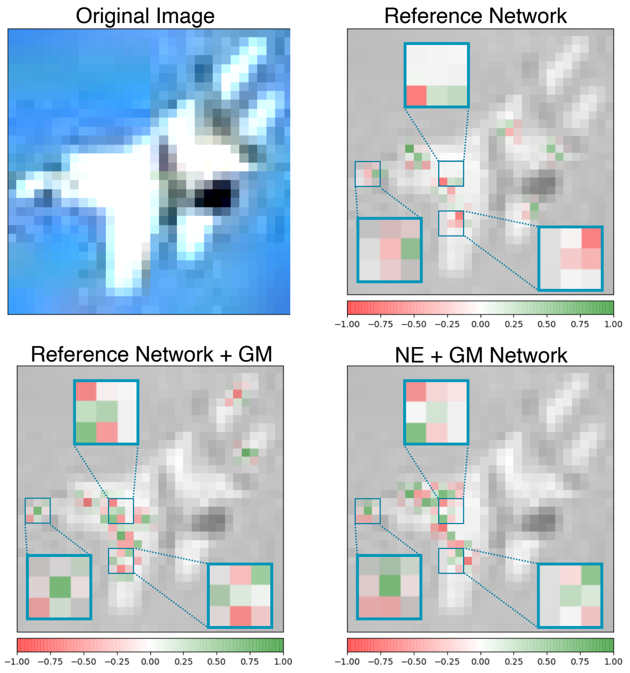
the baseline networks while maintaining good clean accuracy.

In the second experiment, the reduced versions of the aforementioned datasets are employed to test whether the neuroevolution with GM method can also work well with limited datasets. Several specialized applications utilize limited datasets (Operiano et al., 2020; Shaikhina and Khovanova, 2017; Zhang and Ling, 2018; Oh et al., 2020; Wang et al., 2020) such as the dangerous objects X-ray and it is important that the proposed method is dataset size agnostic. Since the datasets used are limited, the baseline networks are replaced by simple hand-engineered networks to compensate for the dataset size. The second experiment demonstrates that even with a limited dataset, the produced NE+GM networks can still attain high clean accuracy and good adversarial accuracy.

6.2.1 Full-Dataset Experiment

6.2.1.1 Clean Accuracy and Adversarial Accuracy

In this subsection, the clean accuracies and adversarial accuracies of the baseline and NE+GM networks trained on full-dataset are compared. For the reference network, ResNet-34 is employed. The reference network, together with PGD with L_∞ norm (L_∞-PGD) are used to generate the adversarial example equivalent of all the test set images in CIFAR-10. The results of the baseline networks and NE+GM networks are shown in Table 6.6. When the clean accuracies are compared, the NE+GM networks have higher results than the reference network. Moreover, it is noteworthy that the NE+GM networks have achieved a remarkable increase in the adversarial accuracy with a difference of at least 31% against the reference network and 11% against the reference network with GM. The results on the reference network with GM indicate that the NE can successfully find more robust network architectures, which help GM to misalign input gradients better,



(dist. from Ref. Net. : 0.687) (dist. from Ref. Net. : 1.297)

Figure 6.3: Integrated gradients (Sundararajan et al., 2017) of the reference network, reference network with GM, and NE+GM network show the focus areas of every network using the relative values in the range of $[-1, 1]$. Through the comparison of the colors on the same pixel locations, it can be observed that the focus areas of the reference network with GM (lower left) and NE+GM network (lower right) have complementary colors to that of the reference network, which suggests dissimilarity in the input gradients. Moreover, the total pixel distance of the NE+GM network (1.297) against the reference network is higher than that of the reference network with GM (0.687). It implies that the NE+GM network input gradients are more dissimilar to those of the reference network.

than the reference network architecture.

The capability of GM to misalign input gradients of networks can be visually confirmed using integrated gradients (Sundararajan et al., 2017). In Fig. 6.3, there are three networks used, which are the reference network, reference network with GM, and the NE+GM network. Each network shows its input gradients values through the color overlay on the pixels of the example image. The red color indicates a negative direction, whereas green indicates a positive direction. If the upper right box is observed, it can be seen that the color of the pixels in the reference network changes into the complementary color in the reference with GM and NE+GM networks. The change in pixel colors shows that the direction of the input gradient becomes the opposite. The same occurrence can be observed in other boxes, which implies that GM can successfully encourage networks to misalign their input gradients with respect to the reference network. In addition, the

Table 6.7: The adversarial robustness of the baseline networks and NE+GM network against the adversarial examples from different networks are presented here. NE+GM network shows consistent robustness while the two baselines do not.

	Ref. Network	Ref. Network+GM	NE+GM Network
ResNet-18	36.84%	47.40%	**53.99%**
VGG	42.98%	52.35%	**55.23%**
DenseNet	36.67%	48.44%	**53.93%**
SqueezeNet	44.23%	52.48%	**55.09%**

total pixel distance of the NE+GM network with the reference network is higher than those of the reference network with GM with the reference network (1.297 and 0.687 respectively). Thus, the proposed NE with GM framework can find network architectures that misalign input gradients better than the reference network architecture.

6.2.1.2 Transferability of Adversarial Examples generated from Standard Networks

The robustness of the NE+GM networks is also tested with the adversarial examples generated by other networks. In addition to the reference network (ResNet-34), there are four sets of adversarial examples generated from the CIFAR-10 test set using the network models namely: ResNet-18, VGG, DenseNet, and SqueezeNet. In Table 6.7, the reference network exhibits the lowest adversarial accuracies, which demonstrates its vulnerability to the transferability of adversarial examples from the network models. Training the reference network architecture with GM improves the adversarial accuracies by 5-10% depending on the network type. However, the NE+GM network has remarkably consistent adversarial accuracies (lowest transferability), regardless of the architecture type of the source of adversarial examples. These accuracies have significantly outperformed the reference network and reference network with GM, which is attributed to the superior network produced by the NE with GM method.

Additionally, it has been observed that the results in Table 6.7 support the idea that the extent of transferability varies depending on the type and similarity of the networks. For example, the reference network has a higher transferability of adversarial examples from the ResNet-18 than the VGG network. This is because the reference network, which uses the ResNet-34 architecture, shares similarities with the ResNet-18, which both use

Table 6.8: The baseline networks and the NE+GM network produced in Section 6.2.1.1 are retrained using MNIST and KMNIST datasets. Among the networks, the NE+GM network has the best clean accuracy and adversarial accuracy.

	MNIST		
	Ref. Net.	Ref. Net.+GM	NE+GM Net.
Clean Acc.	97.71%	97.73%	**99.10%**
Adv. Acc.(L_∞-PGD)	87.90%	95.50%	**98.10%**
	KMNIST		
	Ref. Net.	Ref. Net.+GM	NE+GM Net.
Clean Acc	95.69%	96.06%	**97.51%**
Adv. Acc.(L_∞-PGD)	75.80%	89.10%	**94.40%**

skip connections. VGG does not employ skip connections. Therefore, the difference in architecture has made adversarial examples from the VGG less effective on the reference network. This observation also holds true on DenseNet, which uses skip connections, and SqueezeNet, which does not use skip connections similar to ResNet. The reference network is more vulnerable to adversarial examples from DenseNet than SqueezeNet. In contrast, though the NE+GM network employs skip connections, it is designed in such a way that it can resist adversarial attacks from networks with skip connections. Moreover, it is also robust to adversarial attacks from networks without skip connection, resulting in a consistently robust network.

6.2.1.3 Results on other Datasets

The NE+GM networks produced in Section 6.2.1.1 are tested by retraining (i.e., training from scratch) them with different datasets, which are MNIST and KMNIST datasets. Similar to the previous experiment, the adversarial examples for each dataset are obtained by adversarially attacking all the test set images using the reference network and PGD with L_∞ norm. The NE+GM network, as shown in Table 6.8, has achieved remarkable clean and adversarial accuracies on both datasets, which agrees with the results in Table 6.6. Although the NE+GM network architecture is optimized for the CIFAR-10 dataset, it has outperformed the reference network with GM in all comparisons on different datasets. This performance is attributed to the abundant features provided by CIFAR-10 to evolve a particularly robust NE+GM network. Thus, the NE+GM network can adjust its parameters to the features of different datasets such as MNIST and still

Table 6.9: The clean accuracy and adversarial accuracy of baseline networks, an additional baseline network that is a simple hand-engineered network with GM (denoted as HE. Net. +GM), and NE+GM network trained with a reduced dataset (i.e., reduced CIFAR-10) is shown here. The results show that the NE+GM networks have the best clean accuracy while having comparable adversarial accuracy with reference network with GM.

	Ref. Net.	Ref. Net.+GM	HE. Net.+GM	NE+GM Net. 1	NE+GM Net. 2
Clean Acc.	50.00%	53.40%	55.40%	**57.80%**	56.20%
Adv. Acc.	8.80%	63.06%	43.32%	63.17%	**63.56%**

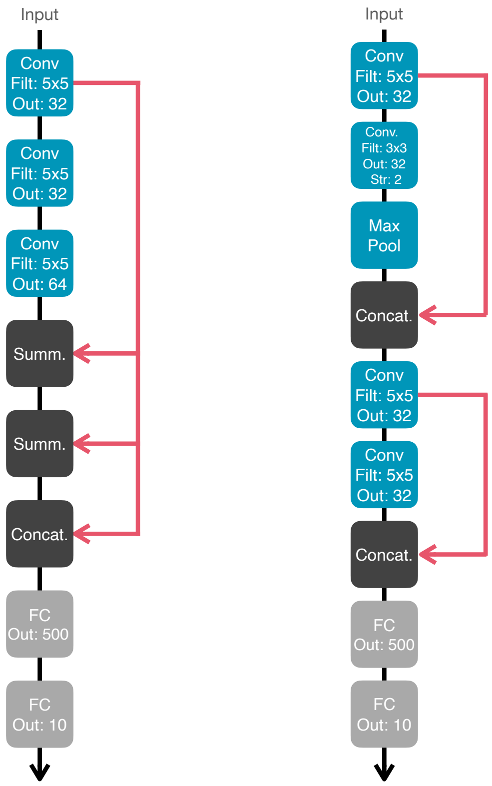
maintain a good performance.

6.2.2 Reduced-Dataset Experiment

6.2.2.1 Clean Accuracy and Adversarial Accuracy

In this subsection, the clean accuracy and adversarial accuracy of the baseline and NE+GM networks trained on a reduced dataset are compared. To create a reduced CIFAR-10 dataset, the full CIFAR-10 training set is decreased from 50,000 images down to 10,000 images (1,000 images per label), and the test set from 10,000 images down to 1,000 images (100 images per label). The adversarial examples from the reduced test set are generated again using the reference network and the PGD with L_∞ norm. With a limited dataset, the reference network is changed into a simple hand-engineered network because a deep network such as ResNet-34 is hard to train using a small dataset. In addition to the baseline networks, which are the reference network and reference network with GM, another simple hand-engineered network with GM is introduced. The simple hand-engineered networks consist of convolutional blocks, pooling blocks, and skip connection blocks that are no more than 10 blocks in total. The architecture detail of the simple hand-engineered networks is shown in Fig. 6.4.

As reported in Table 6.9, the networks that are trained with GM achieve better clean accuracy than the reference network. In particular, the NE+GM networks have achieved better clean accuracy than all the networks. This result confirms the capacity of neuroevolution to find better and optimal network architectures. The adversarial accuracies of the reference network with GM and the NE+GM networks are comparable but they are surprisingly higher than their respective clean accuracies. Although this result is counterintuitive, it has not been observed in the full-dataset experiment.



(a) Simple Hand-engineered Network (Reference Network)

(b) Simple Hand-engineered Network (Additional Baseline Network)

Figure 6.4: Simple Hand-engineered Networks Architecture.

Hence, this behavior is regarded as distinct to limited datasets. Finally, the NE+GM networks have better adversarial accuracy than the hand-engineered network with GM, which shows that the extent of transferability of adversarial examples can be reduced when the network architecture is properly designed.

6.2.2.2 Results on Different Adversarial Attack Methods

To examine the impact of the type of adversarial attack on the robustness of the NE+GM networks produced in Section 6.2.2.1, different adversarial attacks are tested. There are three adversarial attacks tested, which are PGD with L_∞ norm, PGD with L_2 norm (L_2-PGD), and FGSM. Each adversarial attack generates its adversarial examples on all the reduced CIFAR-10 test set images. The evaluation metric used in this experiment is the fooling rate because with the small dataset, the average clean accuracy is relatively low, and the high misclassification of the unperturbed images can obscure the actual performance of the networks on adversarial examples. The fooling rate is based

Table 6.10: The fooling rate under three adversarial attack methods (lower is better). NE+GM networks perform best for most of the attack methods.

	Ref. Net.+GM	HE. Net.+GM	NE+GM Net. 1	NE+GM Net. 2
L_∞-PGD	17.42%	30.26%	**16.10%**	18.77%
L_2-PGD	36.04%	36.41%	**28.83%**	31.19%
FGSM	69.07%	**54.36%**	60.26%	65.21%

on the ratio of images that changes labels when adversarially attacked. It implies that the images before being adversarially attacked can be correctly classified by the network. For each adversarial attack method, the fooling rate of a network from the baseline and NE+GM networks is computed by extracting all the images that are correctly labeled by both the network being tested and the reference network. Next, the extracted images are adversarially attacked using the adversarial attack method to generate adversarial examples. Subsequently, the adversarial images are evaluated by the network being tested, and the percentage of mislabeled adversarial examples is obtained as the fooling rate result.

As shown in Table 6.10, the fooling rate results in the L_∞-PGD agrees with Table 6.9 as expected. L_2-PGD usually generates weaker adversarial attacks than the L_∞-PGD. Consequently, the networks trained with GM have lower results on L_2-PGD than L_∞-PGD because the input gradients of the reference network are more susceptible to L_∞-PGD. The stronger the adversarial attack, the more effective misalignment of input gradients becomes in reducing transferability. Still, among the networks, the NE+GM networks have the lowest fooling rate on L_2-PGD adversarial attack. The FGSM adversarial attack is the weakest among the three techniques. To easily compare the performance of the networks in FGSM, its perturbation level is adjusted to output stronger adversarial attacks but with very obvious image perturbations to the human perception. The hand-engineered network with GM has outperformed all the networks with the lowest fooling rate unexpectedly. One of the reasons NE+GM networks have not performed best is due to the adversarial attack method used in the evolution, which is L_∞-PGD. However, the NE+GM networks still have performed better than the reference network with GM.

Table 6.11: The fooling rate of the baseline networks and NE+GM network are compared to different adversarial defense methods. The free adversarial training and fast adversarial training are denoted as A.T. (Free) and A.T. (Fast) respectively in this table. The results show that NE+GM networks can perform better than other adversarial defense methods (lower is better).

	Ref. Net.	Ref. Net.+GM	HE. Net.+GM	NE+GM Net.
L_∞-PGD	85.00%	22.80%	24.20%	**22.40%**
	JPEG Comp.	Bilateral Filt.	A.T. (Free)	A.T. (Fast)
L_∞-PGD	84.4%	48.80%	51.00%	43.80%

6.2.2.3 Comparison with Standard Adversarial Defense Methods

The NE+GM networks are compared to several adversarial defense methods to measure the effectiveness of the proposed NE with GM method. In the adversarial defense group, there are two image filtering methods, which are JPEG compression (Guo et al., 2018) and bilateral filtering (Xie et al., 2019), and two adversarial training methods, which are free adversarial training (Shafahi et al., 2019) and fast adversarial training (Wong et al., 2019) that are employed. All the adversarial defense methods are applied on the reference network to evaluate the performance of each adversarial defense method. Since the dataset is still limited, the fooling rate is used as the evaluation metric. In contrast to the image extraction method presented in Section 6.2.2.2, the images extracted for this experiment are obtained from the reduced CIFAR-10 test set that are correctly classified by the reference network only. Subsequently, the extracted images are adversarially attacked using the reference network and the L_∞-PGD. The adversarial examples generated are used for all the adversarial defense methods, including the baseline and NE+GM networks as well. Note that the fooling rate described in this section is modified to account for the image filtering methods that cannot classify images on their own.

In Table 6.11, the results show that the image filtering techniques, specifically the JPEG compression, do not demonstrate a strong adversarial attack due to the passive nature of the defense technique. However, the NE+GM network shows strong defense because the NE with GM method actively searches for the architectural solutions, which makes the network it produces robust even with the strong adversarial attacks such as L_∞-PGD. In the case of fast and free adversarial training methods, the limited dataset has hindered the ability of these techniques to produce good results. Otherwise, a large

Table 6.12: The baseline networks and the NE+GM network produced in Section 6.2.2.1 are retrained using other datasets (i.e., MNIST, FMNST, and KMNIST dataset). Although the results between the reference network with GM and NE+GM network are comparable in the FMNIST and KMNIST datasets, the NE+GM network consistently performs well on clean accuracy and adversarial accuracy across all the datasets.

	MNIST		
	Ref. Net.	Ref. Net.+GM	NE+GM Net.
Clean Acc.	**93.20%**	92.30%	89.10%
Adv. Acc.(L_∞-PGD)	63.40%	77.00%	**84.50%**
	FMNIST		
	Ref. Net.	Ref. Net.+GM	NE+GM Net.
Clean Acc	**88.90%**	85.90%	87.00%
Adv. Acc.(L_∞-PGD)	51.20%	**82.90%**	81.90%
	KMNIST		
	Ref. Net.	Ref. Net.+GM	NE+GM Net.
Clean Acc.	85.40%	**86.80%**	86.60%
Adv. Acc.(L_∞-PGD)	56.90%	83.50%	**83.70%**

dataset such as the full CIFAR-10 applied on adversarial training techniques produces good performance. However, when using a limited dataset, balancing the clean accuracy and the adversarial accuracy on the adversarial training techniques produces fooling rates that are significantly higher than those of the reference network with GM, hand-engineered network with GM, and NE+GM network.

6.2.2.4 Results on other Datasets

The robustness of the NE+GM networks produced in Section 6.2.2.1 is tested with different reduced datasets, similar to the experiment in Section 6.2.1.3. The datasets used in this experiment are the MNIST, FMNIST, and KMNIST datasets. Each dataset, as in Section 6.2.2.1, is reduced to the same training set and test set counts as in the reduced CIFAR-10 dataset and the adversarial examples are generated in the same manner. As shown in Table 6.12, the reference network has the best clean accuracies in the MNIST and FMNIST datasets in exchange for the worst accuracies on its adversarial accuracies counterparts. In contrast, the reference network with GM and the NE+GM network have managed to balance the clean and adversarial accuracies with the exception of the MNIST dataset, in which the reference network with GM has a significantly lower adversarial accuracy than the NE+GM network. The decent performance of the NE+GM network on datasets that it is not optimized for is attributed to the limited features provided by the reduced CIFAR-10 dataset. Unlike the NE+GM network produced for the full CIFAR-10 dataset, the NE+GM network produced for the reduced CIFAR-10 dataset is relatively unrefined.

6.3 Summary

In this chapter, the experiments conducted on utilizing neuroevolution to discover optimal architectures for different applications are discussed. In the first application, neuroevolution finds small network architectures that can replace the deep backbone of YOLOv3 without a decrease in performance. Moreover, the size of the small network architecture is less than half the YOLOv3 backbone. In the second application, neuroevolution combined with gradient misalignment discovers networks that are robust to the transferability of adversarial examples. The NE+GM networks even outperform the reference network trained with gradient misalignment. This chapter demonstrates that neuroevolution has a valuable contribution in automatically designing networks for specific applications.

CHAPTER VII

CONCLUSION AND FUTURE WORK

The success of a convolutional neural network is predicated on its very deep architecture and a large dataset that trains it. Although this combination has led to achieving great performance in various applications and even surpassing human-level performance in image classification, it requires a considerable amount of computational power. However, this amount of resources is not accessible to most researchers. Moreover, the very deep network architecture is not always the optimal depth to train different datasets, especially specific applications which have limited datasets. Furthermore, designing a small network that can compete with deep networks requires expertise and a great effort in trial-and-error since the inner workings of the network are not yet fully understood. Therefore, it is pertinent to develop architectures that are optimized to datasets and perform as well as very deep and complex networks.

Neuroevolution is a method that employs a genetic algorithm to search and optimize the connection weights and architecture of an artificial neural network. Using the guidelines in NEAT, which is one of the best implementations of neuroevolution, the architecture of the convolutional neural network optimized with neuroevolution is proposed. The experiments on neuroevolution techniques demonstrated that neuroevolution discovers network architectures that perform on par with standard networks such as ResNet. Moreover, transfer learning is a common technique to utilize the learning of a trained network to another network. However, since neuroevolution produces different architectures, it cannot utilize the trained weights. Instead, knowledge distillation can be used to guide the training of a neuroevolution-produced network with a trained network as confirmed by the experiments. Through knowledge distillation, the neuroevolution-produced network can have performance boosts. Finally, neuroevolution is also tested to optimize the convolutional neural network architecture for specific applications (i.e., dangerous object X-ray detection). In one application, neuroevolution established that it can discover networks that are significantly smaller than the standard deep networks and still have comparable performance. In another application, neuroevolution found architectures that are robust to the transferability of adversarial attacks, particularly when combined with gradient misalignment.

The experiments and applications conducted for neuroevolution proved that it can significantly help the convolutional neural network to a great extent. It helps researchers to harness the ability of a convolutional neural network that accommodates their requirements without needing considerable computation resources and expertise. Many future directions can be explored in neuroevolution for convolutional neural networks. For example, the evolutions of the architecture and the optimization of weights using backpropagation may be studied further to understand their relationship. When it is understood, the basic building blocks of convolutional neural networks can be established and even the trained weights can be transferred and utilized. Furthermore, it can offer some understanding on how the networks learn during training. All of these are considered as a continuation of this study.

References

Abner, N. 2014. There once was a verb: the predicative core of possessive and nominalization structures in american sign language. Sign language & linguistics 17 (06 2014)

Agapie, A. and Wright, A. H. 2014. Theoretical analysis of steady state genetic algorithms. Applications of mathematics 59.5 (2014): 509–525.

Altman, N. S. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. The american statistician 46.3 (1992): 175–185.

Angeline, P. J., Saunders, G. M., and Pollack, J. B. 1994. An evolutionary algorithm that constructs recurrent neural networks. IEEE transactions on neural networks 5.1 (1994): 54–65.

Bartlett, P. and Downs, T. 1990. Training a neural network with a genetic algorithm. University of Queensland.

Belew, R., McInerney, J., and Schraudolph, N. 1991. Evolving networks: using genetic algorithms with connectionist learning.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. 2013. Evasion attacks against machine learning at test time. In Joint european conference on machine learning and knowledge discovery in databases, pp. 387–402.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. 1992. A training algorithm for optimal margin classifiers. In Proceedings of the annual workshop on computational learning theory, pp. 144–152.

Buciluă, C., Caruana, R., and Niculescu-Mizil, A. 2006. Model compression. In Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining, pp. 535–541. : ACM.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. 2019. Once-for-all: train one network and specialize it for efficient deployment. In International conference on learning representations.

Canziani, A., Paszke, A., and Culurciello, E. 2016. An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678 (2016)

Chan, C., Ginosar, S., Zhou, T., and Efros, A. A. 2019. Everybody dance now. In Proceedings of the IEEE/CVF international conference on computer vision, pp. 5933–5942. : IEEE.

Chen, D., Giles, C. L., Sun, G.-Z., Chen, H., Lee, Y.-C., and Goudreau, M. W. 1993. Constructive learning of recurrent neural networks. In IEEE international conference on neural networks, pp. 1196–1201. : IEEE.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. 2018. Deep learning for classical japanese literature. arXiv preprint arXiv:1812.01718 (2018)

Collins, R. J. and Jefferson, D. R. 1992. The evolution of sexual selection and female choice. In Toward a practice of autonomous systems: proceedings of the first european conference on artificial life, pp. 327–336.

Das, N., Shanbhogue, M., Chen, S.-T., Hohman, F., Chen, L., Kounavis, M. E., and Chau, D. H. 2017. Keeping the bad guys out: protecting and vaccinating deep learning with jpeg compression. arXiv preprint arXiv:1705.02900 (2017)

Demontis, A., Melis, M., Pintor, M., Jagielski, M., Biggio, B., Oprea, A., Nita-Rotaru, C., and Roli, F. 2019. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In Proceedings of the USENIX conference on security symposium, pp. 321–338.

Devaguptapu, C., Agarwal, D., Mittal, G., and Balasubramanian, V. N. 2020. On adversarial robustness: a neural architecture search perspective. arXiv preprint arXiv:2007.08428 (2020)

DeVries, T. and Taylor, G. W. 2017. Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552 (2017)

Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., and Li, J. 2018. Boosting adversarial attacks with momentum. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 9185–9193. : IEEE.

Dumoulin, V. and Visin, F. 2016. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285 (2016)

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. 2017. Dermatologist-level classification of skin cancer with deep neural networks. volume 542, pp. 115–118. : Nature Publishing Group.

Fontanari, J. F. and Meir, R. 1990. The effect of learning on the evolution of asexual populations. Complex systems 4.4 (1990)

Fukushima, K. 1980. Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological cybernetics 36.4 (1980): 193—202.

Gatys, L. A., Ecker, A. S., and Bethge, M. 2016. Image style transfer using convolutional neural networks. In The IEEE conference on computer vision and pattern recognition. : IEEE.

Goldberg, D. E. and Holland, J. H. 1988. Genetic algorithms and machine learning. (1988)

Gomez, F. J., Miikkulainen, R., et al. 1999. Solving non-markovian control tasks with neuroevolution. In IJCAI, volume 99, pp. 1356–1361.

Goodfellow, I., Shlens, J., and Szegedy, C. 2015. Explaining and harnessing adversarial examples. In International conference on learning representations.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. 2020. Generative adversarial networks. Communications of the acm 63.11 (2020): 139–144.

Gou, J., Yu, B., Maybank, S. J., and Tao, D. 2021. Knowledge distillation: a survey. International journal of computer vision 129.6 (2021): 1789–1819.

Gruau, F., Whitley, D., and Pyeatt, L. 1996. A comparison between cellular encoding and direct encoding for genetic neural networks. In Proceedings of the first annual conference on genetic programming, pp. 81–89.

Guo, C., Rana, M., Cisse, M., and van der Maaten, L. 2018. Countering adversarial images using input transformations. In International conference on learning representations.

Guo, M., Yang, Y., Xu, R., Liu, Z., and Lin, D. 2020. When nas meets robustness: in search of robust architectures against adversarial attacks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 631–640. : IEEE.

Harp, S. A., Samad, T., and Guha, A. 1989. Towards the genetic synthesis of neural network. In Proceedings of the third international conference on Genetic algorithms, pp. 360–369.

He, K., Zhang, X., Ren, S., and Sun, J. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778. : IEEE.

Hertz, J., Krogh, A., and Palmer, R. G. 2018. Introduction to the theory of neural computation. CRC Press.

Hinton, G., Vinyals, O., and Dean, J. 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)

Hinton, G. E., Osindero, S., and Teh, Y.-W. 2006. A fast learning algorithm for deep belief nets. Neural computation 18.7 (2006): 1527–1554.

Holland, J. H. et al. 1992. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. 2017. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700–4708. : IEEE.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. 2016. Squeezenet: alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)

Iba, H., Hasegawa, Y., and Paul, T. K. 2009. Applied genetic programming and machine learning. CRC Press.

Inoue, H. 2018. Data augmentation by pairing samples for images classification. arXiv preprint arXiv:1801.02929 (2018)

Ioffe, S. and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pp. 448–456.

Jalwana, M. A., Akhtar, N., Bennamoun, M., and Mian, A. 2020. Orthogonal deep models as defense against black-box attacks. IEEE Access 8 (2020): 119744–119757.

Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. 2009. What is the best multi-stage architecture for object recognition? In IEEE 12th international conference on computer vision, pp. 2146–2153. : IEEE.

Kariyappa, S. and Qureshi, M. K. 2019. Improving adversarial robustness of ensembles with diversity training. arXiv preprint arXiv:1901.09981 (2019)

Karras, T., Aila, T., Laine, S., and Lehtinen, J. 2017. Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)

Kenney, J. F. and Keeping, E. 1962. Linear regression and correlation. Mathematics of statistics 1 (1962): 252–285.

Kim, H., Garrido, P., Tewari, A., Xu, W., Thies, J., Niessner, M., Pérez, P., Richardt, C., Zollhöfer, M., and Theobalt, C. 2018. Deep video portraits. ACM transactions on graphics 37.4 (2018): 1–14.

Kitano, H. 1990. Designing neural networks using genetic algorithms with graph generation system. Complex systems 4 (1990): 461–476.

Kotyan, S. and Vargas, D. V. 2020. Evolving robust neural architectures to defend from adversarial attacks. CEUR workshop proceedings 2640 (2020)

Koza, J. R. and Rice, J. P. 1991. Genetic generation of both the weights and architecture for a neural network. In International joint conference on neural networks, volume 2, pp. 397–404. : IEEE.

Krizhevsky, A., Hinton, G., et al. 2009. Learning multiple layers of features from tiny images. (2009)

Krizhevsky, A., Sutskever, I., and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pp. 1097–1105.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86.11 (1998): 2278–2324.

Lee, S.-W. 1996. Off-line recognition of totally unconstrained handwritten numerals using multilayer cluster neural network. IEEE transactions on pattern analysis and machine intelligence 18.6 (1996): 648–652.

Li, Z., Xiong, X., Ren, Z., Zhang, N., Wang, X., and Yang, T. 2018. An aggressive genetic programming approach for searching neural network structure under computational constraints. arXiv preprint arXiv:1806.00851 (2018)

Lienhart, R. and Maydt, J. 2002. An extended set of haar-like features for rapid object detection. In Proceedings of international conference on image processing, volume 1. : IEEE.

Lim, S.-H., Young, S., and Patton, R. 2016. An analysis of image storage systems for scalable training of deep neural networks.

Lin, M., Chen, Q., and Yan, S. 2013. Network in network. arXiv preprint arXiv:1312.4400 (2013)

Liu, H., Simonyan, K., and Yang, Y. 2018a. Darts: differentiable architecture search. In International conference on learning representations.

Liu, J. and Jin, Y. 2021. Multi-objective search of robust neural architectures against multiple types of adversarial attacks. Neurocomputing 453 (2021): 73–84.

Liu, X., Wang, X., and Matwin, S. 2018b. Improving the interpretability of deep neural networks with knowledge distillation. In IEEE international conference on data mining workshops, pp. 905–912. : IEEE.

Liu, Y., Chen, X., Liu, C., and Song, D. 2016. Delving into transferable adversarial examples and black-box attacks. arXiv preprint arXiv:1611.02770 (2016)

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. 2018. Towards deep learning models resistant to adversarial attacks. In International conference on learning representations.

Matsunaga, K., Hamada, A., Minagawa, A., and Koga, H. 2017. Image classification of melanoma, nevus and seborrheic keratosis by deep neural network ensemble. arXiv preprint arXiv:1703.03108 (2017)

Mikołajczyk, A. and Grochowski, M. 2018. Data augmentation for improving deep learning in image classification problem. In International interdisciplinary phd workshop, pp. 117–122. : IEEE.

Miller, G. F., Todd, P. M., and Hegde, S. U. 1989. Designing neural networks using genetic algorithms. In ICGA, volume 89, pp. 379–384.

Minh, T. N., Sinn, M., Lam, H. T., and Wistuba, M. 2018. Automated image data preprocessing with deep reinforcement learning. arXiv preprint arXiv:1806.05886 (2018)

Minsky, M. and Papert, S. 1969. Perceptron: an introduction to computational geometry. The MIT Press, Cambridge, expanded edition 19.88 (1969): 2.

Mitchell, M. 1998. An introduction to genetic algorithms. MIT press.

Mitchell, T. 1997. Machine learning. (1997)

Montana, D. J., Davis, L., et al. 1989. Training feedforward neural networks using genetic algorithms. In IJCAI, volume 89, pp. 762–767.

Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. 2017. Universal adversarial perturbations. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1765–1773. : IEEE.

Nair, V. and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In Proceedings of the international conference on machine learning, pp. 807–814.

Nelder, J. A. and Wedderburn, R. W. 1972. Generalized linear models. Journal of the royal statistical society: Series A 135.3 (1972): 370–384.

Oh, Y., Park, S., and Ye, J. C. 2020. Deep learning covid-19 features on cxr using limited training data sets. IEEE transactions on medical imaging 39.8 (2020): 2688–2700.

Operiano, K. R. G., Iba, H., and Pora, W. 2020. Neuroevolution architecture backbone for x-ray object detection. In IEEE symposium series on computational intelligence, pp. 2296–2303. : IEEE.

Papernot, N., McDaniel, P., and Goodfellow, I. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv preprint arXiv:1605.07277 (2016)

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. 2017. Practical black-box attacks against machine learning. In Proceedings of the ACM on asia conference on computer and communications security, pp. 506–519. : ACM.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. 2019. Pytorch: an imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (ed.), Advances in neural information processing systems, pp. 8024–8035. : Curran Associates, Inc.

Perez, F., Vasconcelos, C., Avila, S., and Valle, E. 2018. Data augmentation for skin lesion analysis. In OR 2.0 context-aware operating theaters, computer assisted robotic endoscopy, clinical image-based procedures, and skin image analysis, pp. 303–311. : Springer.

Quinlan, J. R. 1986. Induction of decision trees. Machine learning 1.1 (1986): 81–106.

Radosavovic, I., Dollár, P., Girshick, R., Gkioxari, G., and He, K. 2018. Data distillation: Towards omni-supervised learning. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4119–4128. : IEEE.

Redmon, J. and Farhadi, A. 2017. Yolo9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 7263–7271. : IEEE.

Redmon, J. and Farhadi, A. 2018. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)

Rosenblatt, F. 1957. The perceptron, a perceiving and recognizing automaton project para. Cornell Aeronautical Laboratory.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review 65.6 (1958): 386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. Learning representations by back-propagating errors. Nature 323.6088 (1986): 533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. 2015. Imagenet large scale visual recognition challenge. International journal of computer vision 115.3 (2015): 211–252.

Samuel, A. L. 1959. Some studies in machine learning using the game of checkers. IBM journal of research and development 3.3 (1959): 210–229.

Shafahi, A., Najibi, M., Ghiasi, A., Xu, Z., Dickerson, J., Studer, C., Davis, L. S., Taylor, G., and Goldstein, T. 2019. Adversarial training for free! In Proceedings of the international conference on neural information processing systems, pp. 3358–3369.

Shaikhina, T. and Khovanova, N. A. 2017. Handling limited datasets with neural networks in medical applications: a small-data approach. Artificial intelligence in medicine 75 (2017): 51–63.

Sharif Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. 2014. Cnn features off-the-shelf: an astounding baseline for recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp. 806–813. : IEEE.

Shorten, C. and Khoshgoftaar, T. M. 2019. A survey on image data augmentation for deep learning. volume 6, p. 60. : Springer.

Simonyan, K. and Zisserman, A. 2014a. Two-stream convolutional networks for action recognition in videos. In Advances in neural information processing systems, pp. 568–576.

Simonyan, K. and Zisserman, A. 2014b. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

Stanley, K. O. and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. Evolutionary computation 10.2 (2002): 99–127.

Suganuma, M., Shirakawa, S., and Nagao, T. 2017. A genetic programming approach to designing convolutional neural network architectures. In Proceedings of the genetic and evolutionary computation conference, pp. 497–504. : ACM.

Summers, C. and Dinneen, M. J. 2019. Improved mixed-example data augmentation. In IEEE winter conference on applications of computer vision, pp. 1262–1270. : IEEE.

Sun, Y., Xue, B., Zhang, M., Yen, G. G., and Lv, J. 2020. Automatically designing cnn architectures using the genetic algorithm for image classification. IEEE transactions on cybernetics 50.9 (2020): 3840–3854.

Sundararajan, M., Taly, A., and Yan, Q. 2017. Axiomatic attribution for deep networks. In International conference on machine learning, pp. 3319–3328.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. 2013. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. 2015. Going deeper with convolutions. In The IEEE conference on computer vision and pattern recognition. : IEEE.

Takahashi, R., Matsubara, T., and Uehara, K. 2020. Data augmentation using random image cropping and patching for deep cnns. volume 30, pp. 2917–2931. : IEEE.

Topchy, A. and Lebedko, O. 1997. Neural network training by means of cooperative evolutionary search. Nuclear instruments and methods in physics research section a: accelerators, spectrometers, detectors and associated equipment 389.1-2 (1997): 240–241.

Torrey, L. and Shavlik, J. 2010. Transfer learning. In Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, pp. 242–264. : IGI global.

Tramèr, F., Boneh, D., Kurakin, A., Goodfellow, I., Papernot, N., and McDaniel, P. 2018. Ensemble adversarial training: attacks and defenses. In International conference on learning representations.

Tramèr, F., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. 2017. The space of transferable adversarial examples. arXiv preprint arXiv:1704.03453 (2017)

Tsukada, R., Zou, L., and Iba, H. 2020. Evolving deep neural networks for x-ray based detection of dangerous objects. In Deep neural evolution, pp. 325–355. : Springer.

Vargas, D. V. and Murata, J. 2016. Spectrum-diverse neuroevolution with unified neural models. IEEE transactions on neural networks and learning systems 28.8 (2016): 1759–1773.

Viola, P. and Jones, M. 2001. Rapid object detection using a boosted cascade of simple features. In Proceedings of the IEEE computer society conference on computer vision and pattern recognition, volume 1. : IEEE.

Wang, D., Mo, J., Zhou, G., Xu, L., and Liu, Y. 2020. An efficient mixture of deep and machine learning models for covid-19 diagnosis in chest x-ray images. Plos one 15.11 (2020): e0242535.

Whitley, D., Starkweather, T., and Bogart, C. 1990. Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel computing 14.3 (1990): 347–361.

Wong, E., Rice, L., and Kolter, J. Z. 2019. Fast is better than free: revisiting adversarial training. In International conference on learning representations.

Wu, D., Wang, Y., Xia, S.-T., Bailey, J., and Ma, X. 2019. Skip connections matter: on the transferability of adversarial examples generated with resnets. In International conference on learning representations.

Wu, R., Yan, S., Shan, Y., Dang, Q., and Sun, G. 2015. Deep image: Scaling up image recognition. arXiv preprint arXiv:1501.02876 7.8 (2015)

Xiao, H., Rasul, K., and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)

Xie, C., Wu, Y., Maaten, L. v. d., Yuille, A. L., and He, K. 2019. Feature denoising for improving adversarial robustness. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 501–509. : IEEE.

Xie, G., Wang, J., Yu, G., Zheng, F., and Jin, Y. 2021. Tiny adversarial mulit-objective oneshot neural architecture search. arXiv preprint arXiv:2103.00363 (2021)

Xie, L. and Yuille, A. 2017. Genetic cnn. In Proceedings of the IEEE international conference on computer vision, pp. 1379–1388. : IEEE.

Xu, W., Evans, D., and Qi, Y. 2017. Feature squeezing: detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155* (2017)

Yao, X. 1999. Evolving artificial neural networks. *Proceedings of the IEEE* 87.9 (1999): 1423–1447.

Yao, X. and Liu, Y. 1997. A new evolutionary system for evolving artificial neural networks. *IEEE transactions on neural networks* 8.3 (1997): 694–713.

Yao, X. and Liu, Y. 1998. Making use of population information in evolutionary artificial neural networks. *IEEE transactions on systems, man, and cybernetics, part b* 28.3 (1998): 417–425.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. 2014. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328.

Yurtsever, E., Lambert, J., Carballo, A., and Takeda, K. 2020. A survey of autonomous driving: common practices and emerging technologies. *IEEE access* 8 (2020): 58443–58469.

Zeiler, M. D. and Fergus, R. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. 2017. mixup: beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017)

Zhang, Y. and Ling, C. 2018. A strategy to apply machine learning to small datasets in materials science. *Npj computational materials* 4.1 (2018): 1–8.

Zhong, Z., Zheng, L., Kang, G., Li, S., and Yang, Y. 2020. Random erasing data augmentation. In *AAAI*, pp. 13001–13008.

Zou, L., Tanaka, Y., and Iba, H. 2018. Dangerous objects detection of x-ray images using convolution neural network. In *International conference on security with intelligent computing and big-data services*, pp. 714–728.

APPENDIX

APPENDIX A

PUBLICATION

International Journal Publication

1. Kevin Richard G. Operiano, Wanchalerm Pora, Hitoshi Iba, and Hiroshi Kera. Evolving Architectures with Gradient Misalignment toward Low Adversarial Transferability. In 2021 IEEE Access.

International Conference Publications

1. Kevin Richard G. Operiano, Hitoshi Iba, and Wanchalerm Pora. Neuroevolution Architecture Backbone for X-ray Object Detection. In 2020 IEEE Symposium Series on Computational Intelligence (SSCI).
2. Kevin Richard G. Operiano, Pann Mya Hmue, Wanchalerm Pora, and Suree Pumrin. American Alphabet Hand Sign Language Detection and Recognition Using Haar Cascades and Convolutional Neural Networks. In 2018 11th Regional Conference on Electrical and Electronic Engineering.

Vita

NAME	Kevin Richard G. Operiano
DATE OF BIRTH	21 February 1992
PLACE OF BIRTH	Antipolo City, Rizal, Philippines
INSTITUTIONS ATTENDED	De La Salle University - Manila, Philippines
HOME ADDRESS	B3 L6 Pines Hill St. Ridgemont Exec. Vill. Cainta Rizal, Philippines