BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

NEURAL NETWORKS: EXPERIMENTATION WITH ACTIVATION FUNCTIONS

NEURONOVÉ SÍTĚ: EXPERIMENTACE S AKTIVAČNÍMI FUNKCEMI

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **Bc. LILIT MOVSESIAN**
AUTOR PRÁCE

SUPERVISOR **Ing. KAREL FRITZ**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

164938

Institut:	Department of Computer Systems (DCSY)
Student:	**Movsesian Lilit**
Programme:	Information Technology
Title:	**Neural Networks: Experimentation with Activation Functions**
Category:	Artificial Intelligence
Academic year:	2024/25

Assignment:

1. Explore the role of activation functions in neural networks and their impact on neuron outputs and network dynamics.
2. Study different activation functions (e.g., sigmoid, ReLU, tanh, softmax) and examine their properties, advantages, and disadvantages.
3. Design experiments to test various activation functions using metrics like accuracy, convergence speed, and computational efficiency.
4. Implement at least two neural networks with different activation functions and test them on standard datasets (e.g., CIFAR-10).
5. Conduct experiments and observe how activation functions influence network performance.
6. Analyze results and discuss key performance factors.

Literature:

- According to the instructions of the project supervisor.

Requirements for the semestral defence:

- Completion of items 1 - 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Fritz Karel, Ing.**
Head of Department:	Sekanina Lukáš, prof. Ing., Ph.D.
Beginning of work:	1.11.2024
Submission deadline:	14.5.2025
Approval date:	31.10.2024

Abstract

Activation functions are key components of neural networks, as they determine neuron outputs based on inputs and weights. Nevertheless, the choice of activation function is often made without a comprehensive analysis. This thesis investigates the impact of activation functions on the performance of convolutional neural networks (CNNs) in image classification tasks on two datasets, CIFAR-10 and Fashion MNIST. A series of experiments was designed and conducted, focusing on classification accuracy, convergence speed, generalization ability, gradient magnitudes, and resistance to FGSM attacks. The main contribution of this thesis is the comprehensive analysis of various activation functions, including commonly used ones, sinusoidal activations, linear combinations, and the SELU function with a focus on internal normalization. Additionally, the study introduces effective techniques like noise injection into the activation functions and stochastic activation function selection as defense mechanisms against FGSM attacks.

Abstrakt

Aktivační funkce jsou klíčovými komponentami neuronových sítí, protože určují výstupy neuronů na základě vstupů a váh. Výběr aktivačních funkcí se nicméně často provádí bez komplexní analýzy. Tato bakalářská práce zkoumá vliv aktivačních funkcí na výkon konvolučních neuronových sítí (CNNs) při úlohách klasifikace obrazů na dvou datových sadách, CIFAR-10 a Fashion MNIST. Byla navržena a provedena řada experimentů, zaměřených na přesnost klasifikace, rychlost konvergence, schopnost generalizace, velikosti gradientů a odolnost proti útokům FGSM. Hlavním přínosem této práce je komplexní analýza různých aktivačních funkcí, včetně běžně používaných funkcí, sinusových aktivací, lineárních kombinací a funkce SELU se zaměřením na vnitřní normalizaci. Dále studie představuje účinné techniky, jako je přidání šumu do aktivačních funkcí a stochastický výběr aktivační funkce, jako obranné mechanismy proti útokům FGSM.

Keywords

Activation functions, trainable linear combinations of activation functions, stochastic selection of activation functions, noise injection to ReLU, periodic activation functions, SELU, trainable activation functions, resistance to FGSM.

Klíčová slova

Aktivační funkce, trénovatelné lineární kombinace aktivačních funkcí, stochastický výběr aktivačních funkcí, přidání šumu do ReLU, periodické aktivační funkce, SELU, trénovatelné aktivační funkce, odolnost vůči FGSM.

Reference

MOVSESIAN, Lilit. *Neural Networks: Experimentation with Activation Functions.* Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Karel Fritz

Rozšířený abstrakt

Aktivační funkce jsou klíčovou součástí neuronových sítí, protože určují výstupy neuronů na základě vstupů a vah a zavádějí do modelu nelinearitu, čímž zásadně ovlivňují proces učení. Výběr aktivačních funkcí se nicméně často provádí bez komplexní analýzy. Tato bakalářská práce se zaměřuje na podrobnou analýzu vlivu různých aktivačních funkcí na výkonnost konvolučních neuronových sítí (CNN) při klasifikaci obrazových dat. Cílem bylo zjistit, jak rozdílné aktivační funkce ovlivňují přesnost klasifikace, rychlost konvergence, generalizační schopnost, velikost gradientů a odolnost vůči útokům FGSM.

V rámci řešení bylo navrženo několik experimentů na datových sadách CIFAR-10 a Fashion MNIST. Testovány byly standardní aktivační funkce jako sigmoid, tanh, ReLU a její varianty (SELU, SiLU, ELU, GELU, parametrické ReLU), ale také méně běžné přístupy – sinusová aktivační funkce $\sin(x)$, lineární kombinace několika aktivačních funkcí s trénovatelnými parametry a dále přístupy založené na stochastickém výběru aktivační funkce a přidání šumu (Gaussovský, Poissonův, rovnoměrný) do výstupu aktivační funkce.

Výsledky experimentů ukázaly, že pro jednodušší úlohy jako Fashion MNIST nemá volba aktivační funkce zásadní vliv na výkon modelu, zatímco u složitějších úloh jako CIFAR-10 je výběr aktivační funkce klíčový. ReLU a jeho varianty dosahovaly vysoké přesnosti, ale byly náchylnější k přeučení. Sigmoidní a tanh funkce se vyznačovaly lepší generalizací, ale pomalejší konvergencí a problémy s mizejícím gradientem, zejména u CIFAR-10.

Sinusová funkce $\sin(x)$ prokázala schopnost efektivní konvergence, avšak model nevyužíval periodičnosti funkce naplno, zejmena v případě CIFAR-10. Navíc při použití sinusové funkce se projevily problémy s explodujícími gradienty, což vyžadovalo dodatečnou regulaci.

Lineární kombinace více aktivačních funkcí s trénovatelnými váhami sice nabídla flexibilitu a teoretickou možnost napodobit kteroukoliv ze základních funkcí, nicméně v praxi tyto kombinace nedosáhly lepších výsledků než ReLU.

SELU s interní normalizací vykazovala lepší stabilitu než ReLU v pozdějších epochách, obzvlášť při přidání šumu do vstupních dat.

Zavedení šumu do ReLU výrazně zvýšilo odolnost sítě vůči adversariálním útokům typu FGSM. Podobného efektu bylo dosaženo i náhodným výběrem aktivační funkce.

Tato práce přináší komplexní přehled o tom, jak různé aktivizační funkce ovlivňují učení a ukazuje, že techniky, jako je přidání šumu do aktivačních funkcí a stochastický výběr aktivační funkce, mohou být efektivní obrannou strategií proti útokům FGSM. Výsledky práce poskytují podněty pro další výzkum v oblasti robustnosti neuronových sítí a návrhu nových aktivačních funkcí.

Neural Networks: Experimentation with Activation Functions

Declaration

I hereby declare that this thesis is my original work, conducted independently under the supervision of Ing. Karel Fritz. I have listed all used information sources and literature that contributed to my research.

.....
Lilit Movsesian
April 29, 2025

Acknowledgements

I would like to thank my supervisor Ing. Karel Fritz for guiding my bachelor's thesis, for his expert advice, support and approachable attitude.

Contents

List of Figures

List of Tables

List of abbreviations

Adam	Adaptive Moment Estimation. 14, 29
BCE	Binary Cross-Entropy. 13
CCE	Categorical Cross-Entropy. 13
CDF	Cumulative Distribution Function. 20
CNN	Convolutional Neural Network. 2, 11, 12, 26–28, 34, 39, 45
CUDA	Compute Unified Device Architecture. 22
ELU	Exponential Linear Unit. 2, 19, 22, 25, 31, 33, 51
FGSM	Fast Gradient Sign Method. 24, 25, 36, 47, 48, 51, 52, 54
FNN	Feedforward Neural Network. 10
GELU	Gaussian Error Linear Unit. 2, 19–21, 25, 31, 33, 51
GPU	Graphics Processing Unit. 11, 12, 22
LC	Linear Combination. 25, 38, 39, 43
LSTM	Long Short-Term Memory. 12
MAE	Mean Absolute Error. 13
MBE	Mean Bias Error. 13
MLP	Multilayer Perceptron. 10, 13
MSE	Mean Squared Error. 13
MSGD	Mini-Batch Stochastic Gradient Descent. 14
NAG	Nesterov Accelerated Gradient. 14, 29
NLP	Natural Language Processing. 10
ReLU	Rectified Linear Unit. 2, 8, 11, 17–19, 21, 25, 31, 33, 39–41, 43–51, 54
RMSprop	Root Mean Square Propagation. 14, 29
SELU	Scaled Exponential Linear Unit. 2, 21, 22, 25, 31, 33, 45, 46, 51, 54

SGD Stochastic Gradient Descent. 11, 14, 29

SiLU Sigmoid-Weighted Linear Unit. 2, 20, 21, 25, 31, 33, 51

TDNN Time Delay Neural Network. 11

VGG Visual Geometry Group. 12

ZIP Zone Improvement Plan. 11

Chapter 1

Introduction

Activation functions are essential in neural networks as they determine the output of a neuron based on its input and weight, and nonlinear functions are particularly important as they enable the neural network to solve complex nonlinear problems. Moreover, activation functions significantly affect the network's ability to converge and generalize from the training data.

The choice of activation function in single-layer neural networks, as well as for the output units in deep neural networks, is significantly influenced by the type of distribution being modeled and the desired outcome of the classification task.

In the context of hidden units of deep neural networks, a wider range of activation functions becomes relevant. The only essential requirement is that these functions must be differentiable to enable gradient-based optimization, which leaves a variety of possibilities for activation function selection depending on the architecture and the problem being solved.

Commonly used activation functions include the logistic sigmoid, hyperbolic tangent (tanh), Rectified Linear Unit (ReLU), and softmax. Each function has distinct advantages and disadvantages that impact the learning process, such as susceptibility to the problem of vanishing gradients in the case of sigmoid and tanh functions, the computational efficiency of ReLU, or the multiclass classification capabilities of softmax [3].

The choice of activation function is crucial to the performance of neural networks, yet it is common practice to apply the same activation function across all hidden layers with insufficient consideration of alternative approaches. While ReLU is often used by default due to its computational simplicity and efficiency, this approach may overlook activation functions that could be better suited for specific problems.

The aim of this bachelor's thesis is to provide a deeper understanding of how these variations affect network behavior and performance by experimenting with a range of known and custom activation functions.

Chapter 2 covers relevant machine learning fundamentals. The proposed experimental design, including the choice of datasets and evaluation criteria, are introduced in chapter 3, while chapter 4 details the experimental setup and presents and interprets the results obtained through experiments. Chapter 5 summarizes the obtained findings.

Chapter 2

Methodology

This chapter discusses the theoretical fundamentals of machine learning, from early neural networks to their modern implementations, and explains their main components.

2.1 Introduction to Neural Networks

Neural networks are models designed to solve complex problems, originally inspired by the human nervous system. Neurons in the brain communicate through electrical and chemical signals, forming an interconnected system that processes and transmits information. Artificial neural networks simplify these biological processes into mathematical models. The first artificial neural network, introduced by McCulloch and Pitts in 1943, laid the foundation for computational learning by representing a neuron as a sum of inputs with weights passed through a nonlinear activation function [30]. Mathematically, this is expressed as:

$$a = \sum_{i=1}^{M} w_i x_i, \quad y = f(a),$$

where x_i are inputs, w_i are weights representing strengths, a is the pre-activation, and $f(\cdot)$ is the activation function [3]. This fundamental model provided the basis for further advancements in machine learning to the present day.

2.2 Perceptron

The perceptron is considered the first of three key phases in the history of neural network development [3]. It was one of the earliest models of artificial neural networks, introduced and simulated by Frank Rosenblatt in 1957 [34], and later extended in his 1962 book [35]. The perceptron is a single-layer neural network designed for binary classification tasks.

The perceptron is referred to as a single-layer neural network, even though it has multiple processing layers in its structure. This is because only one layer is learnable from the data, where each node is connected to the input features through weights. The output is computed by passing the weighted sum of the inputs plus a bias term through the nonlinear activation function.

The perceptron is characterized by its use of a nonlinear step activation function [3]:

$$f(a) = \begin{cases} 0 & \text{if } a \leq 0, \\ 1 & \text{if } a > 0. \end{cases}$$

The output is computed as:

$$y(x) = f(\mathbf{w}^T \boldsymbol{\varphi}(x)),$$

where $\boldsymbol{\varphi}(x)$ is the transformed input vector, $\mathbf{w}^T$ represents the transposed weight vector, and $f(\cdot)$ is the step function.

The perceptron algorithm aims to minimize misclassification by adjusting the weights during training. The perceptron learning algorithm works by calculating the output for each input pattern, updating the weights by adding or subtracting the feature vector if the output is misclassified, and repeating this process until convergence.

The error, or loss, function, based on the total number of misclassified patterns, is piecewise constant with discontinuities. This characteristic makes methods that rely on changing weights using the gradient of the loss function ineffective, as the gradient is zero almost everywhere. An alternative loss function called the perceptron criterion is used to provide a continuous, linear loss function, allowing for weight updates during training.

The perceptron convergence theorem guarantees that if the data is linearly separable, the algorithm will eventually find a solution. However, if the data is not linearly separable, the algorithm will fail to converge. Additionally, the perceptron does not generalize easily to more than two classes and does not provide probabilistic outputs [2].

Modern uses of perceptrons mainly involve multilayer perceptions (MLP), which is sometimes used to refer to multilayer feedforward networks with fully connected layers. MLP can perform binary and multiclass classification problems, as well as more complex prediction problems such as part-of-speech tagging and sequence segmentation in natural language processing (NLP) [14].

2.3 Backpropagation

The introduction of modern backpropagation represents the second milestone in the evolution of neural networks. The algorithm was initially introduced in several studies during the 1970s, first applied to neural networks in 1982 [50], and formally described as a component of machine learning in 1986 [36]. Backpropagation revolutionized neural networks by enabling the training of multilayer neural networks with multiple layers of parameters.

The algorithm relies on differential calculus and gradient-based optimization to propagate errors backward through the network, enabling the optimization of parameters in all layers of the neural network. This process is critical for updating the weights during training. The main modifications made to earlier versions of neural networks involved replacing the step activation function with differentiable activation functions and introducing differentiable loss functions to enable gradient-based optimization of parameters.

The parameters of each layer are first initialized using a random number generator or a specific initialization algorithm. A forward pass is then performed to compute the output and evaluate the loss function, giving rise to the term „feedforward neural networks“ (FNN). During the error backpropagation process, errors are propagated from the output layer to the input layers. This process calculates the gradients for each parameter, which are

subsequently used by optimization algorithms, such as stochastic gradient descent (SGD), to update the parameters iteratively [3].

The introduction of backpropagation was an important milestone as this method made it possible to train multilayer neural networks.

2.4 Deep Neural Networks

The third key phase of the evolution of neural networks is the current phase of deep neural networks with many hidden layers [3]. One of the reasons for the deepening of neural networks was the development of training of neural networks with the use of graphics processing units (GPUs). The idea, first published in 2004 [31] and studied in the following years [43] [5], enabled faster and more efficient computations for training deep learning models. It laid the foundation for developing deeper and more sophisticated architectures.

As the architectures grew in complexity, so did the number of parameters. In the 1980s, neural networks typically contained only a few hundred or thousands of parameters, but this number has grown dramatically over time. Modern state-of-the-art models can contain up to a trillion (10^{12}) parameters [3].

However, the depth of these networks can cause problems such as the problem of vanishing or exploding gradients, which can hamper the training. In addition, the increase in depth often slows the training process, making optimization more complex. To address these issues, techniques like residual connections, introduced in 2015 through the ResNet architecture [17], have been employed. These connections allow information to bypass certain layers.

2.5 Convolutional Neural Networks

A convolutional neural network (CNN) is a type of feedforward network that is used mainly for computer vision tasks.

The first CNN architecture, along with max pooling, which is an important downsampling method used in CNNs, was introduced in 1979 by Fukushma [13]. However, backpropagation, which is now a standard method in training neural networks, was not used in the learning algorithm of this pioneering model. Notably, this scientist was also the first to employ the ReLU activation function for visual feature extraction [12], which has since become a widely favored activation function in CNNs.

The time delay neural network (TDNN), introduced by Waibel et al. in 1989 [49], extended the concepts of CNNs to phoneme recognition tasks. Phonemes are the smallest possible speech sounds of the language. The TDNN utilized backpropagation, which was not implemented in the learning algorithm of the pioneering CNN.

A significant milestone in CNN development was LeNet, introduced by LeCun et al. in 1989, which was successfully used for handwritten ZIP code recognition [26]. This evolved into LeNet-5 in 1998, a deeper architecture capable of recognizing digits in 32×32 pixel images, becoming one of the first practical applications of CNN in banking systems to process handwritten checks [27].

Wei Zhang made significant contributions to the theoretical foundations of CNNs. In 1990, Zhang et al. introduced a parallel distributed processing model with local space-invariant interconnections, which means that connections between neurons follow a pattern that remains unchanged regardless of the location of those neurons, which enables the model

to classify patterns even if the objects are shifted, deformed or distorted [54]. Furthermore, Zhang's research on using CNNs for the detection of cancer in digital mammograms marked an important application of neural networks in medical imaging [53].

The beginning of the 21st century was marked by the development of training of neural networks with the (GPUs), which significantly increased the complexity and performance of CNNs.

This advancement led to the emergence of deep architectures such as the GPU-trained CNN introduced in 2011, which achieved superhuman performance in visual pattern recognition contests [7]. AlexNet, introduced in 2012, also won the computer vision competition due to its excellent performance and again demonstrated the power of deeper networks trained with GPUs [25]. Other important models include architectures such as very deep Visual Geometry Group (VGG) [39], and GoogleNet, which introduced the inception module, which creates parallel branches of convolutional kernels of different sizes to enhance the feature extraction [46].

To address the challenges of training very deep neural networks, more advanced architectures were proposed, such as Highway Networks [42] and their well-known variant ResNet [17], which allow data to skip layers. Highway Networks introduced gating mechanisms that help regulate the flow of information across many layers, inspired by the recurrent neural network architecture long short-term memory (LSTM) [19]. These skip connections facilitate the effective training of networks with hundreds of layers. This idea was extended by the concept of dense connectivity, which was introduced by Huang et al. in 2017 with the DenseNet architecture [21], where each layer is connected to every other layer, promoting feature reuse and improving learning efficiency.

2.5.1 Architecture of CNNs

The architecture of CNNs generally consists of several key components: an input layer, a sequence of convolution and pooling layers, fully connected layers, activation functions, and an output layer. The input layer receives raw data, while the following initial layers, composed of convolution and pooling layers, focus on feature extraction. The fully connected layers perform classification or regression tasks on the extracted features and produce the output layer.

Convolution layers use filters or kernels, which are small matrices applied to the input, to detect features such as edges, textures, or more complex shapes. Each kernel consists of three dimensions: length, width, and depth. The length and width represent the size of the kernel, with the most commonly used sizes being 3×3 and 5×5. The depth corresponds to the number of channels or the number of feature maps produced at the output of this layer. As the depth increases, feature extraction can be enhanced, but this also leads to a rise in computational complexity.

The convolution is performed by moving the kernel over the input image, element-wise multiplying the kernel and the portion of the image it covers, and then summing the results to produce a resulting pixel in the output feature map. The size of the output feature map is influenced by the size, stride, and padding of the kernel. Stride refers to how much the kernel shifts across the input image after each operation. A padding represents the number of extra pixels added around the border of the input image to preserve the dimensions after the convolution.

The activation functions are typically applied to the resulting feature maps to add nonlinearity to the model.

Pooling, or down-sampling, layers, which typically follow convolutional layers, serve to reduce the dimensions of feature maps and decrease the computational complexity of the model. The pooling layers again have the kernel size that represents the length and width of the region to be aggregated during the operation, and the stride, which defines the shift of the kernel after the operation. The most widely used pooling functions are max-pooling, where the maximum value within the region is selected, and average pooling, which selects the arithmetic mean of all values in the region.

Fully connected layers reduce the number of neurons after the feature maps have been processed by convolutional and pooling layers. Each neuron in fully connected layers is linked to all the neurons in the previous layers, similar to the structure of MLP networks [55]. They aggregate the learned features into a one-dimensional vector, making it suitable for the final classification or regression task. Typically, an activation function is applied after the fully connected layers to add nonlinearity. The selection of the activation function for the output layer depends on the type of distribution being modeled. The softmax activation function is typically used for multiclass classification, while the logistic sigmoid activation function is often applied for binary classification tasks.

2.6 Loss Functions

Loss function, also referred to as a cost or error function, is a quantitative measure of how well predictions align with the actual data.

For regression problems, where the model predicts continuous values, the loss functions directly measure the magnitude of residuals, which are the difference between the observed and predicted values. For such tasks, commonly used loss functions include Mean Bias Error (MBE), Mean Absolute Error (MAE), and Mean Squared Error (MSE) and others [6].

In classification tasks, where the goal is to assign inputs to discrete classes, loss functions measure the difference between the output probabilities and the actual labels. For binary classification tasks, the Binary Cross-Entropy (BCE) loss is widely used, while for multi-class classification, the Categorical Cross-Entropy (CCE) loss is one of the most popular approaches. The CCE loss is mathematically expressed as follows:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log(p_{i,j}),$$

where N is the number of samples, C is the number of classes, $y_{i,j}$ is a ground truth, that equals 1 if sample i belongs to class j, $p_{i,j}$ is the output probability of sample i belonging to class j [48].

2.7 Optimization Algorithms

Optimization algorithm is an integral part of the neural network training process, as it allows updating the model parameters in order to minimize the loss function [44].

One important parameter of optimization algorithm is the learning rate, which specifies the size of a step at which the model parameters are updated during training. Choosing an appropriate learning rate is crucial, as optimization with a very large value can result in divergence, while small value can cause slow convergence or convergence to a local minimum of the loss function [51].

A common approach is gradient-based optimization, where the parameters are adjusted based on the gradient of the loss function. The earliest and the most common optimization method is the gradient descent, where parameters are updated iteratively by moving in the opposite direction of the gradients.

Stochastic Gradient Descent (SGD) reduces the update time and the computational complexity of gradient descent by updating the model parameters based on one randomly selected sample per iteration. Thus, it calculates the estimation of the real gradient instead of directly calculating the exact gradient.

However, SGD's random sample selection introduces additional noise, leading to oscillations of gradient direction during training. To solve this problem, a compromise between the gradient descent and SGD, the mini-batch gradient descent method (MSGD), was proposed. The difference is that the model parameters are updated based on a small batch of samples, rather than a single sample or the entire dataset.

SGD with momentum is another approach, which can accelerate the convergence by introducing a variable that preserves the influence of previous update directions on the next iteration to a certain degree.

Nesterov Accelerated Gradient Descent (NAG) is a momentum-based optimizer that enhances the previous method by „looking ahead“ to the future position of parameters when calculating gradients. This approach incorporates more gradient information, leading to faster convergence [44].

Adaptive gradient-based methods, such as AdaGrad, Root Mean Square Propagation (RMSprop), Adam, AdamW, AdaDelta, and AdaMax, dynamically adjust the learning rates for each parameter based on historical gradient information, often leading to better performance on complex datasets. Among them, AdaGrad was one of the earliest adaptive learning rate methods proposed to enhance SGD by adjusting the learning rate based on the sum of squared gradients from previous iterations.

Building upon this idea, Adaptive Moment Estimation (Adam) combines the benefits of momentum and adaptive learning rates by maintaining an exponentially decaying average of the gradients and an exponentially decaying average of the squared gradients [44]. Adam has become one of the most commonly used optimization algorithms due to its high efficiency.

2.8 Activation Functions in Neural Networks

Activation functions are essential elements of neural networks because they determine the output of the neuron by applying a mathematical function to its input and weights. Activation functions can be divided into two types: linear and nonlinear. Linear activation functions, however, are rarely used in modern neural networks, as they perform a linear transformation and do not allow a neural network to approximate complex, nonlinear relationships in the data. A neural network with only linear activation functions in hidden layers would essentially act as a single-layer network, as successive linear transformations are equivalent to a single transformation [2]. This limitation is why nonlinear activation functions such as softmax, ReLU, sigmoid, and tanh are predominantly used in practice.

The activation functions in the output layers are chosen depending on the distribution that needs to be modeled. For hidden layers, the only important requirement is that the function needs to be differentiable to allow backpropagation, which allows researchers to explore how different activation functions influence key neural network parameters such as convergence speed, classification accuracy, resistance to adversarial attacks, and so on.

While there is a range of commonly used activation functions that have been extensively studied and applied in various architectures, researchers continue to explore novel activation functions to improve network performance.

2.8.1 Logistic Sigmoid Activation Functions

Logistic sigmoid is a nonlinear differentiable activation function that was widely used in the early stages of neural network development. While its use has diminished in modern architectures due to its limitations, such as vanishing gradients, it remains relevant in specific applications. Logistic sigmoid is particularly suitable for binary classification tasks or probability predictions, as it maps input values to a range of 0 to 1, representing probabilities.

The function is mathematically defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

The sigmoid function is differentiable, and its derivative is given by:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))[45].$$

The graph of the logistic sigmoid function is displayed in Figure 2.1, while the graph of its derivative is shown in Figure 2.2.

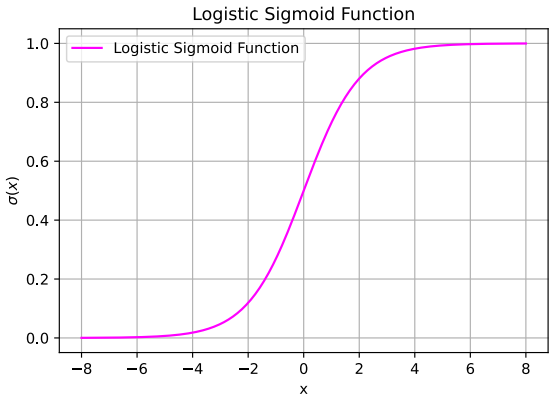

Figure 2.1: Logistic Sigmoid Function

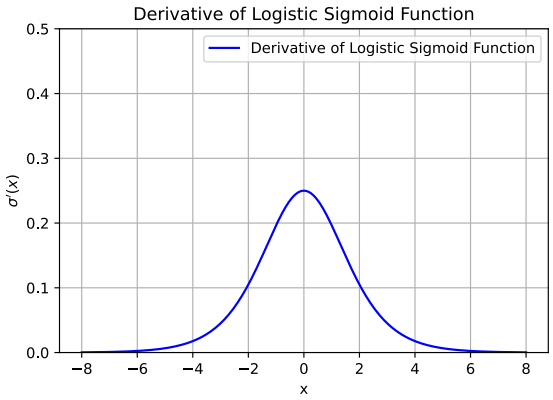

Figure 2.2: Derivative of Sigmoid Function

The derivative of the logistic sigmoid function reveals that the gradients are significant only within the range of approximately -5 to 5. Outside of this range, the gradient values of the function will be very small and approach zero. As the depth of the neural network increases, it may encounter the problem of vanishing gradients. This occurs when the gradients of the early layers diminish significantly during backpropagation due to the chain rule, causing the training of these layers to slow down or, in extreme cases, to stop entirely.

Another challenge of the sigmoid function is that it is not symmetric around zero. Since all gradients and outputs in a layer have the same sign, this creates imbalanced gradients for positive and negative inputs. This can potentially cause slower convergence, particularly in deeper networks.

2.8.2 Hyperbolic Tangent Activation Function

Hyperbolic tangent or tanh activation function is closely related to logistic sigmoid function. In fact, it is a scaled and shifted version of the logistic sigmoid function, which maps input values to the range from -1 to 1. Notably, every network with the logistic sigmoid activation functions in its hidden layers has an equivalent network utilizing the hyperbolic tangent function [2].

The tanh function is mathematically defined as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The tanh function is differentiable, and its derivative is expressed as:

$$\tanh'(x) = 1 - \tanh^2(x).$$

The hyperbolic tangent function is displayed in Figure 2.3, and its derivative is presented in Figure 2.4.

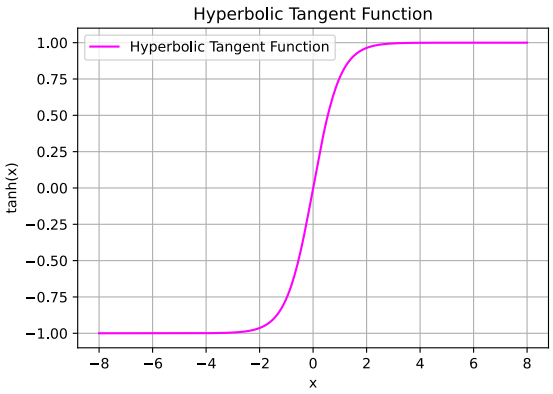

Figure 2.3: Hyperbolic Tangent Function

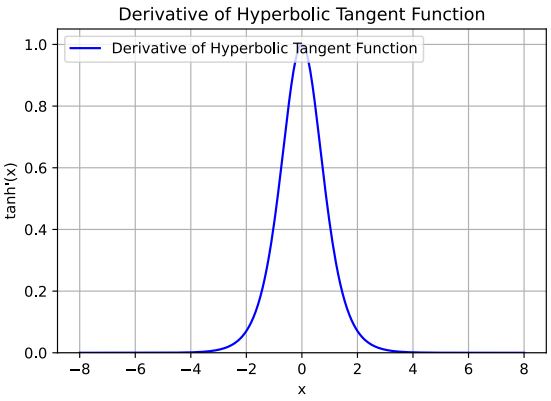

Figure 2.4: Derivative of Tanh

As illustrated in the graph of the derivative, the tanh function also suffers from the vanishing gradient problem. However, its zero-centered outputs provide an advantage over the logistic sigmoid function, improving gradient-based optimization in deep networks [45].

2.8.3 Softmax Activation Function

Softmax activation function generalizes the logistic sigmoid function to multiple dimensions, transforming an input into a probability distribution. This makes it particularly useful in the output layer of neural networks designed for multiclass classification, where probabilities indicate the likelihood of each class.

Mathematically, the softmax function σ is defined as:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \quad \text{for } i = 1, \ldots, K,$$

where $x = (x_1, x_2, \ldots, x_K) \in \mathbb{R}^K$ is the input vector, and K is the number of classes. Each value $\sigma(x)_i$ of the output vector is within the range from 0 to 1, and the sum of all components is 1, which represents probabilities [16].

2.8.4 ReLU Activation Function

Rectified Linear Unit, or ReLU, is one of the most effective and best-performing activation functions and is widely used in various architectures due to its simplicity and computational efficiency. Despite its name, ReLU is a differentiable piecewise linear function. It maps negative inputs to 0, while positive values remain unchanged, which enables ReLU to have a derivative everywhere except 0. The derivative at 0 is typically assigned a value of either 0 or 1.

Mathematically, the ReLU function is expressed as:

$$\mathrm{ReLU}(x) = \max(0, x).$$

The derivative of the ReLU function is expressed as:

$$\mathrm{ReLU}'(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0. \end{cases}$$

The ReLU function is displayed in Figure 2.5, and its derivative is presented in Figure 2.6.

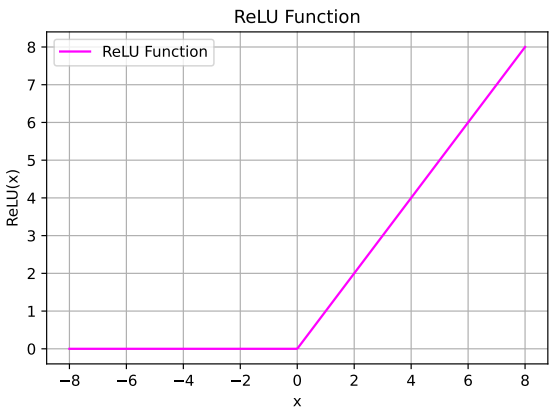

Figure 2.5: ReLU Function

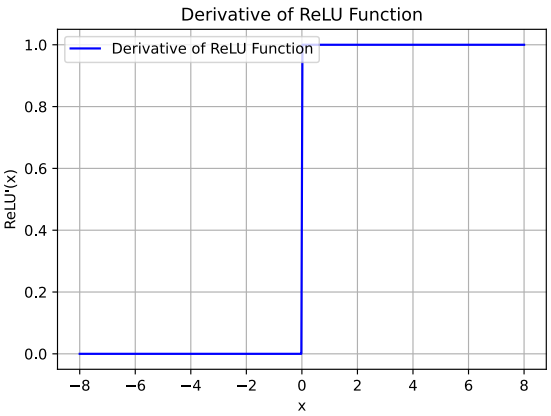

Figure 2.6: Derivative of ReLU

ReLU activation function has several advantages in comparison with the activation functions discussed earlier. One important strength is computational efficiency due to sparse activation, where approximately 50% of neurons are activated.

Additionally, due to its linear property, ReLU does not saturate in either direction and avoids the vanishing gradient problem that is common in sigmoid-like activation functions. ReLU is unbounded, and it has been reported that neural networks with unbounded activation functions have the property of universal approximation and are theoretically capable of approximating any function due to large gradients, which allow efficient feature extraction [41].

However, large gradients can lead to the „exploding gradients“ problem, which occurs when gradients during backpropagation become very large and can cause unstable parameter updates and even convergence failure. To address this issue, techniques such as gradient clipping, proper weight initialization, and normalization layers are often used to ensure that gradients remain within a certain range.

Another significant limitation of ReLU is the „Dying ReLU“ or „dying neurons“ problem. This issue arises due to sparse activation, which occurs when the inputs to a neuron are always negative, causing the neuron to output zero for all inputs. As a result, no gradient flows through the neuron during backpropagation, and the weights of the neuron stop updating. This leads to dead neurons that no longer contribute to the learning process. A solution to this problem is to use leaky ReLU, where a small positive slope is added for negative inputs, which ensures that neurons do not become completely inactive [45].

2.8.5 Leaky ReLU Activation Function

The leaky ReLU (leaky Rectified Linear Unit) is an altered variant of the ReLU activation function. By introducing a small positive slope for negative inputs, it enables backpropagation for negative inputs and thus effectively addresses the problem of „dying ReLU“. While it shares many of the advantages of ReLU, the training process can be more time-consuming due to additional computations.

The leaky ReLU function is mathematically defined as:

$$(\text{Leaky ReLU})(x) = \begin{cases} \alpha x & \text{if } x \leq 0, \\ x & \text{if } x > 0, \end{cases}$$

where α is the slope for the negative part of the input. The derivative of the leaky ReLU function is given as:

$$(\text{Leaky ReLU})'(x) = \begin{cases} \alpha & \text{if } x \leq 0, \\ 1 & \text{if } x > 0 [45]. \end{cases}$$

The leaky ReLU is displayed in Figure 2.7, and its derivative is shown in Figure 2.8.

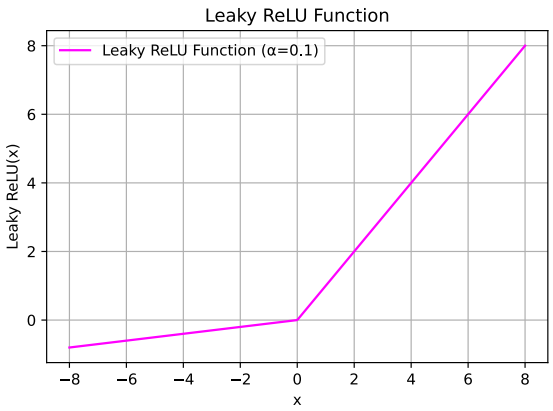

Figure 2.7: Leaky ReLU Function

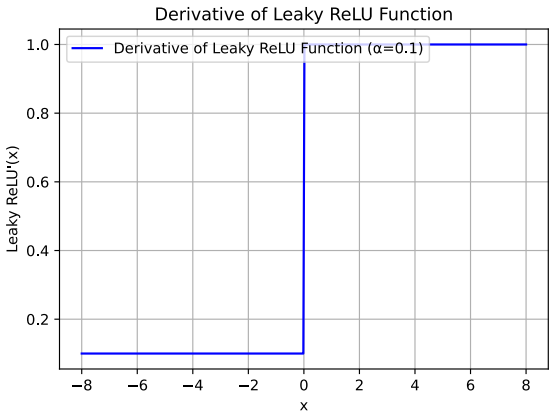

Figure 2.8: Derivative of leaky ReLU

2.8.6 Parametric ReLU Activation Function

The parametric ReLU (parametric Rectified Linear Unit) is a modification of both ReLU and leaky ReLU, where the slope α is a learnable parameter. Unlike fixed values in leaky ReLU, the slope α is optimized alongside the other network parameters through backpropagation, allowing the function to adapt dynamically during training.

Mathematically, the parametric ReLU function is defined as:

$$(\text{Parametric ReLU})(x) = \max(\alpha x, x),$$

where α is a trainable parameter.

2.8.7 ELU Activation Function

ELU, or an Exponential Linear Unit function, is an alternative to ReLU function introduced in 2016 by Clevert et al. [8]. Unlike ReLU, which outputs zero for negative inputs, ELU uses an exponential curve for negative values, allowing for a non-zero output that saturates at $-\alpha$. This feature helps mitigate the issue of „dying neurons“ by ensuring that negative inputs do not cause a complete loss of information and also shifts the mean activations of neurons closer to zero compared to ReLU, where outputs are biased towards large positive values.

The function is piecewise, with a linear output for positive inputs and an exponential output for negative inputs, defined as:

$$(\text{ELU})(x) = \begin{cases} \alpha\,(e^x - 1) & \text{if } x \leq 0, \\ x & \text{if } x > 0, \end{cases}$$

where $\alpha \geq 0$ is a hyperparameter that controls this saturation value for negative inputs.

The derivative of the ELU function is given as:

$$(\text{ELU})'(x) = \begin{cases} \alpha e^x & \text{if } x \leq 0, \\ 1 & \text{if } x > 0. \end{cases}$$

The ELU function is shown in Figure 2.9, and its derivative is displayed in Figure 2.10.

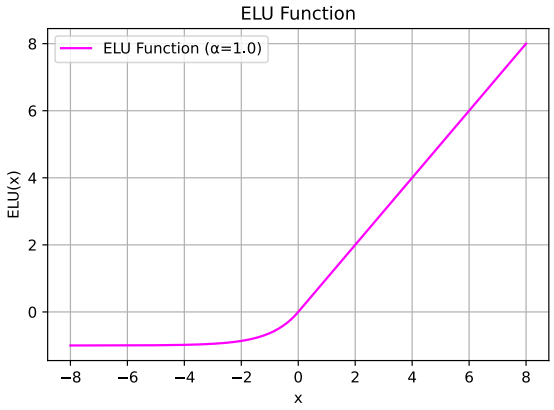

Figure 2.9: ELU Function

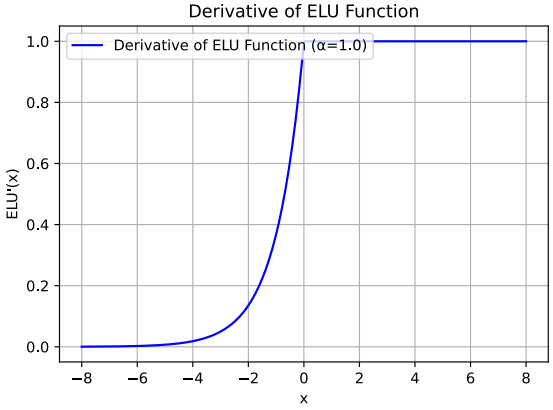

Figure 2.10: Derivative of ELU

2.8.8 GELU Activation Function

Gaussian-Error linear Unit, or GELU, is a high-performance activation function introduced by Hendrycks and Gimpel in 2016 [18]. Unlike ReLU, which multiplies the input by 0 or 1 based on its sign, GELU multiplies the inputs by values, which are probabilistically determined by the value of the input itself. Specifically, the GELU activation function

scales the input x by its standard Gaussian cumulative distribution function (CDF) $\Phi(x)$ as follows:

$$\text{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2}\left(1 + \text{loss}\left(\frac{x}{\sqrt{2}}\right)\right),$$

where $\Phi(x)$ is the CDF, and $\text{loss}(x)$ is the loss function.

Its derivative is defined as:

$$\text{GELU}'(x) = x \cdot \Phi'(x) + \Phi(x).$$

The GELU is shown in Figure 2.11, while its derivative is shown in Figure 2.12.

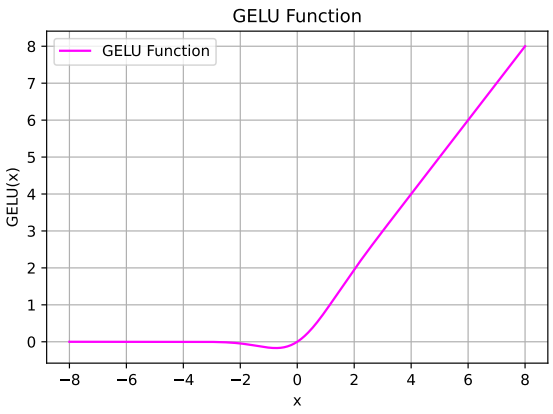

Figure 2.11: GELU Function

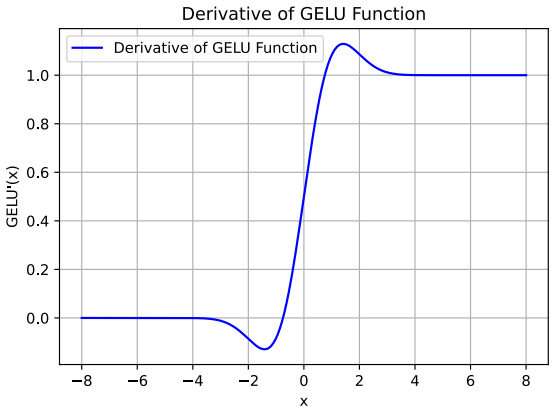

Figure 2.12: Derivative of GELU

GELU, similar to how dropout works, stochastically deactivates units during training. A GELU input with a lower value has a higher probability of being „dropped“ based on its probability according to a Gaussian distribution. This results in a smoother transition between activated and deactivated neurons, which can enhance the generalization of the model. The GELU function has been shown to outperform ReLU and ELU in various tasks, such as image classification and speech recognition [18].

2.8.9 SiLU Activation Function

Sigmoid Linear Unit Activation Function, or SiLU, was introduced by Elfwing et al. in 2017 [10]. SiLU function combines linearity with smooth behavior by scaling the input x by the sigmoid function, which helps avoid sharp transitions near 0. This is mathematically expressed as:

$$\text{SiLU}(x) = x \cdot \sigma(x),$$

where $\sigma(x)$ is the sigmoid function, defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

The derivative of SiLU is:

$$\text{SiLU}'(x) = \sigma(x) + x \cdot \sigma'(x),$$

where

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

This results in:

$$\text{SiLU}'(x) = \sigma(x) + x \cdot \sigma(x)(1 - \sigma(x)).$$

The SiLU function is shown in Figure 2.13, and its derivative is illustrated in Figure 2.14.

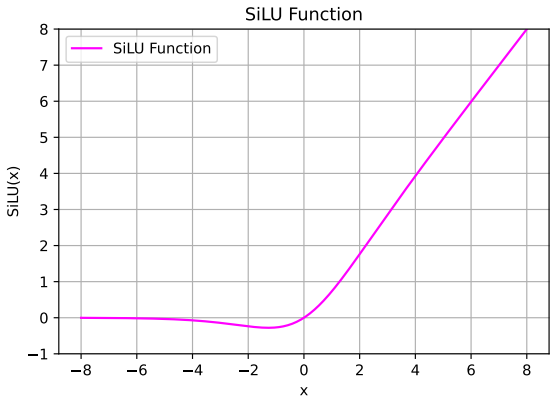

Figure 2.13: SiLU Function

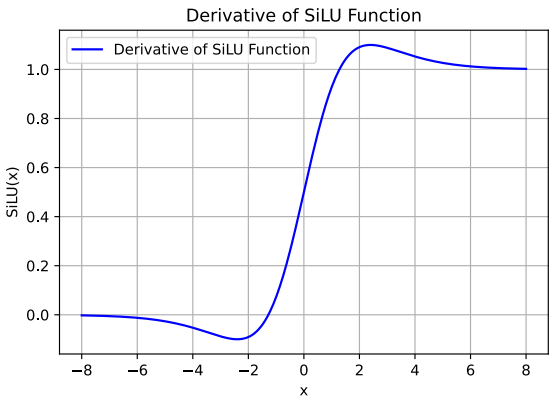

Figure 2.14: Derivative of SiLU

Compared to ReLU, SiLU and GELU activation functions produce non-zero outputs for small negative inputs, which mitigates the problem of „dying neurons“. Elfwing et al. demonstrated that SiLU outperformed traditional activation functions, such as ReLU and sigmoid, in tasks like stochastic SZ-Tetris and Tetris on small boards [10].

2.8.10 SELU Activation Function

Scaled Exponential Linear Unit, or SELU, is an activation function with self-normalizing properties, which was introduced in 2017 by Klambauer et al. [23]. SELU is designed to enable automatic normalization of activations across layers by driving the mean of activations towards 0 and the standard deviation towards 1. This feature accelerates convergence during training and help mitigate exploding and vanishing gradient problems. Networks using SELU often require no additional batch normalization layers.

The function is mathematically defined as:

$$\text{SELU}(x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0, \\ x & \text{if } x > 0, \end{cases}$$

where $\lambda > 1$ and $\alpha > 0$ are constants chosen to allow self-normalization. Typically, $\lambda = 1.0507$ and $\alpha = 1.6733$ are used, as these values were shown to provide optimal performance in deep networks.

The derivative of SELU is:

$$\text{SELU}'(x) = \lambda \begin{cases} \alpha e^x & \text{if } x \leq 0, \\ 1 & \text{if } x > 0. \end{cases}$$

The SELU function is shown in Figure 2.15, and its derivative is illustrated in Figure 2.16.

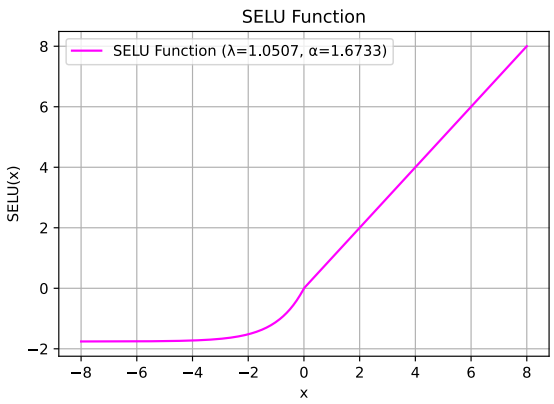

Figure 2.15: SELU Function

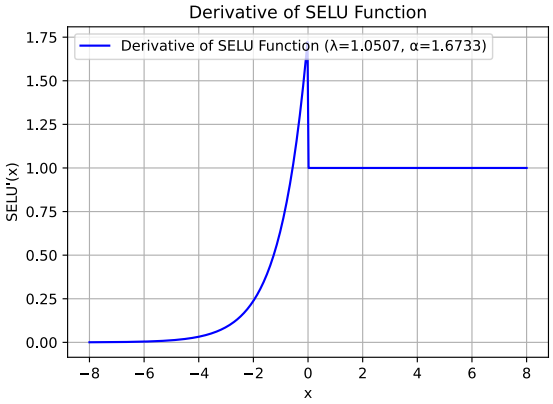

Figure 2.16: Derivative of SELU

Similarly to ELU, SELU also solves the „dying neuron“ problem by allowing both positive and negative inputs to propagate through the network.

2.8.11 Significance of Activation Functions in Neural Networks

The choice of activation functions significantly impacts the performance of a neural network. The activation functions can affect the convergence speed, generalization ability, resistance to overfitting, classification accuracy, resistance to adversarial attacks, and so on. This makes research on activation functions both relevant and valuable. In this bachelor's thesis, their influence on key neural network parameters is examined.

2.9 Relevant Tools

2.9.1 Python and PyTorch

Python is one of the most commonly used programming languages for machine learning and data analysis due to its simplicity and large number of relevant libraries. In this work, the PyTorch framework was employed to build and train all neural network models.

2.9.2 Visualization Tools: TensorBoard and Matplotlib

Two Python libraries were used to visualize the results of the experiments. TensorBoard was employed as a working tool during the experimental phase, while Matplotlib was used to create the plots presented in this thesis.

2.9.3 Compute Unified Device Architecture (CUDA)

To speed up the training of the neural networks, all experiments were performed on a Nvidia RTX A5000 GPU, using CUDA (Computer-Unified Device Architecture). CUDA, developed by Nvidia, is a software that gives access to the GPU and enables GPU-accelerated parallel computation [1]. PyTorch's built-in CUDA support ensures seamless integration with hardware.

Chapter 3

Proposed Solution

This chapter presents the chosen datasets, experimental methodology, and the design of experiments with activation functions.

3.1 Datasets

To conduct experiments on activation functions, two datasets were selected: CIFAR-10 and Fashion MNIST. The CIFAR-10 dataset was chosen as a more complex benchmark suitable for testing activation functions in deeper architectures, while the Fashion MNIST dataset was selected as a simpler benchmark for experiments with shallower neural networks. This selection allows for a comparative study of how different activation functions affect model performance depending on network depth and dataset complexity.

3.1.1 CIFAR-10 Dataset

The CIFAR-10 dataset, developed in 2009 by Krizhevsky, is commonly used in research on deep learning methods for image classification. It consists of 60,000 32×32 color images, organized into 10 classes, with 6,000 images per class. The dataset is divided into training and test subsets, with 50,000 images in the training subset and 10,000 in the test subset. Each class represents objects such as airplanes, cars, birds, cats, and so on, making the dataset moderately challenging due to the relatively small image size and the variation in object appearances across different classes [24].

3.1.2 Fashion MNIST Dataset

Fashion MNIST was developed in 2017 by Xiao et al. as a more challenging alternative to the original MNIST dataset [52]. Like MNIST, Fashion MNIST consists of 70,000 grayscale images of size 28×28, organized into 10 classes, with each class containing 7,000 images. The dataset is divided into a training set of 60,000 images and a test set of 10,000 images. However, instead of handwritten digits, Fashion MNIST contains images of different clothing items such as T-shirts, trousers, pullovers, sandals, and so on.

3.2 Evaluation Factors

In various experiments, the following factors were considered:

- **Convergence speed**: The rate at which the loss function decreases during model training.
- **Generalization ability and Overfitting resistance**: The model's capacity to perform effectively on new, unseen data instead of memorizing the training set.
- **Classification accuracy**: The percentage of correct predictions out of the total number of predictions.
- **Resistance to adversarial attacks**: The model's ability to maintain high performance under adversarial conditions.
- **Gradient Magnitudes**: The mean gradient per epoch, which indicates the strength of updates to the model's parameters.

3.3 Implementation of Adversarial Attack

The white-box FGSM (Fast Gradient Sign Method) attack was implemented to generate targeted perturbations in the correctly predicted input data. This allowed for the evaluation of how various activation functions behave under adversarial conditions.

Adversarial attacks, introduced by Szegedy et al. in 2014, were designed to mislead the model into making incorrect predictions by generating intentional perturbations in input data [47].

The FGSM attack is a technique introduced by Goodfellow et al. in 2015, where perturbations are generated using the gradient of the model's loss function [15]. The FGSM algorithm involves the following steps: compute the loss function, determine the gradients, calculate the perturbation based on the gradient, and adjust the pixel values of the input data [37].

The perturbation η is calculated as follows:

$$\eta = \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y)),$$

where ϵ is the magnitude of noise, typically kept small, θ represents the parameters of the model, x represents the input to the model, y refers to the ground truth labels, $J(\theta, x, y)$ is the loss function used for the optimization during training.

Using this perturbation η, the adversarial image adv_x can be calculated as:

$$\text{adv}_x = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))[15].$$

A white-box attack is a type of adversarial attack in which the attacker has full information about the model, including its parameters and gradients.

3.4 Design of Experiments with Activation Functions

The primary objective of this research was to analyze the impact of various activation functions on the performance of neural networks. A series of experiments was designed to

explore both traditional and novel activation functions. The key experimental settings are outlined in 3.1:

Experiment	**Objective**
Common and Advanced Activations	Compare widely used functions like tanh, sigmoid, ReLU, parametric ReLU, ELU, SELU, SiLU, GELU.
Sinusoidal Activation	Analyze the impact of sin(x) on training and assess the use of periodicity.
ReLU-Sigmoid-Tanh-Linear Activation	Analyze linear combination (LC) activation function with trainable weights ReLU-Sigmoid-Tanh-Linear.
ReLU-Sigmoid-Tanh-Linear-Sin	Analyze the impact of sin(x) as a part of LC activation function on training.
ReLU-Tanh Activation	Analyze LC activation function with trainable weights ReLU-Tanh.
Batch Normalization vs. SELU	Assess the impact of internal normalization effect on training.
Adding Noise to ReLU Activation	Test the impact of adding noise to the ReLU activation on resistance against FGSM attack.
Stochastic Activation Selection	Investigate the effect of randomly selecting the activation function for each neuron to enhance resistance to FGSM attack.

Table 3.1: Outline of Experiments

A detailed description of the experimental design is provided in the following chapter.

Chapter 4

Experiments and Results

4.1 Repeated Experiments and Averaging Results

To ensure reliability of the model evaluations, each experiment was carried out twice using random seeds 42 and 52. Although this is a relatively small number of repetitions, it was chosen to save computational power and time, while still reducing the risk of obtaining biased or inconsistent results due to randomness in training processes such as weight initialization and data shuffling. The final reported results were obtained by averaging the outcomes across these independent runs.

4.2 Experiments Setup

This section describes the selection of architectures, training strategies, and hyperparameter tuning for the upcoming experiments.

4.2.1 Selection of The Neural Network Architectures

The first step of the experimental design involved selecting a neural network architecture that is well-suited for the CIFAR-10 and Fashion MNIST datasets.

To determine the most suitable architectures, comparative experiments were conducted. The experiment for CIFAR-10 included two architectures from the study by Hossain et al. [20], where the CNN models for CIFAR-10 were obtained using two different search methods: one found through an optimized search algorithm FAWCA (Flexible-greedy Approach), and the other through brute-force search. To distinguish between them, these architectures are referred to as `Hossain_1` (optimized search) and `Hossain_2` (brute-force search).

Additionally, the experiment included five neural network architectures described in the work by Florea et al. [11], where different models were obtained using five hyperparameter optimization methods. These architectures vary in the number of convolutional and fully connected layers, as well as in the number of neurons per layer. For clarity, these models are referred to as `Florea_1` — `Florea_5`.

Finally, the experiment also examined a VGG-like model — an architecture inspired by the deep networks introduced by Simonyan et al. [39]. Their work demonstrated that increasing network depth significantly improves learning performance. The training and validation loss curves of the models are displayed in Figure 4.1.

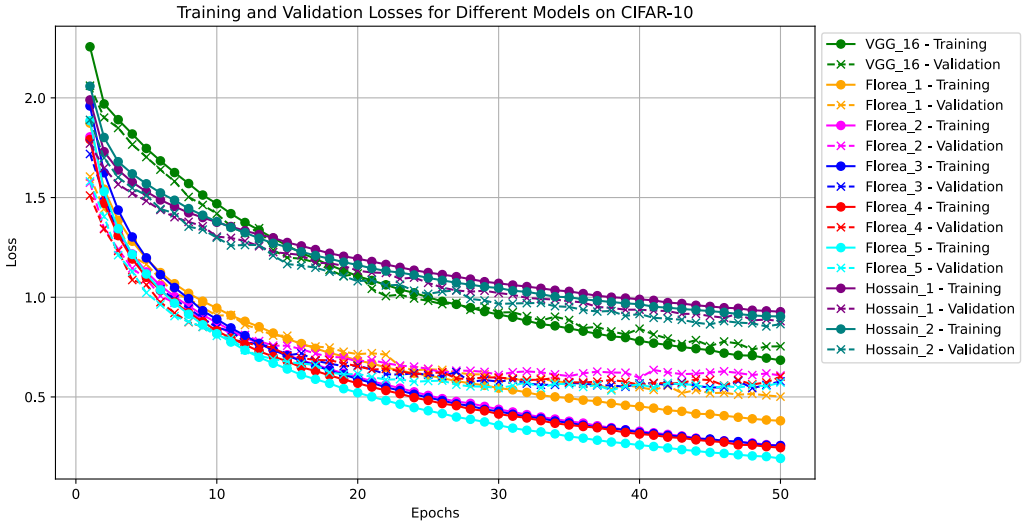

Figure 4.1: Loss Curves for Different Architectures on The CIFAR-10

Models `VGG_16`, `Hossain_1` and `Hossain_2` exhibited minimal overfitting to the training data but had a relatively slow convergence speed. Among the five models proposed by Florea et al., which demonstrated the best performance, `Florea_3` was selected for further experiments with activation functions due to its balanced performance, having the average training time per epoch and convergence speed among the five models. Its detailed architecture is presented in Figure 4.2.

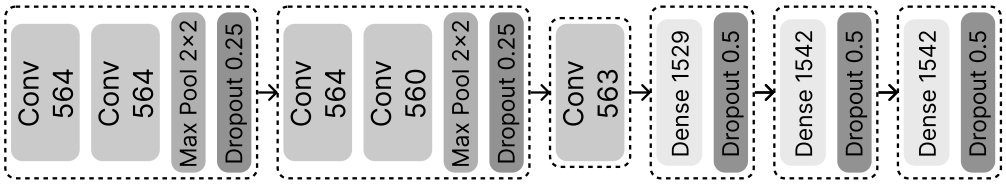

Figure 4.2: Selected CNN Architecture for CIFAR-10 Dataset

The experiment for the Fashion MNIST dataset included two architectures from the study by Hossain et al. [20], which were also discovered using the FAWCA (`Hossain_3`) and brute-force (`Hossain_4`) algorithms but specifically for Fashion MNIST.

Additionally, the experiment included two fully connected feedforward models, `FNN_300_100` and `FNN_500_150`, named according to the number of neurons in their layers. Furthermore, two custom small CNN architectures, `mini_CNN_2` and `mini_CNN_3`, were tested. These models are named based on the number of layers they contain. The training and validation loss trends of these models are shown in Figure 4.3.

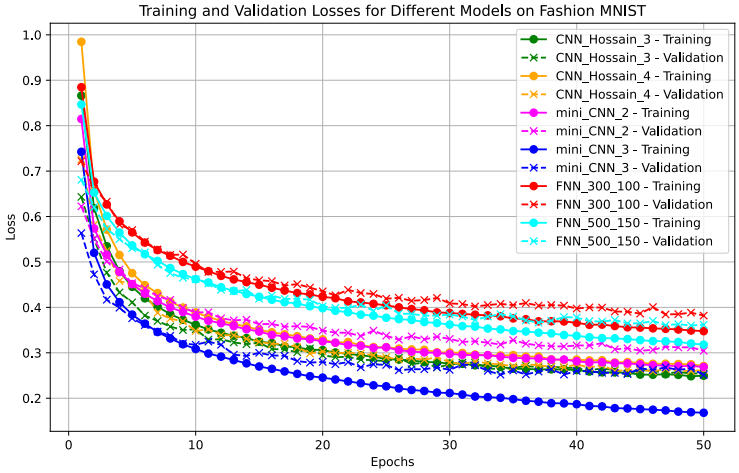

Figure 4.3: Loss Curves for Different Architectures on The Fashion MNIST

Models `Hossain_3` and `Hossain_4` demonstrated high resistance to overfitting but exhibited slow convergence. In contrast, `mini_CNN_3` demonstrated fast convergence and was chosen for further experiments. This choice was also made to specifically assess the impact of activation functions on shallower neural networks, as deeper architectures are already being tested in the CIFAR-10 experiments. The detailed architecture of `mini_CNN_3` is shown in Figure 4.4.

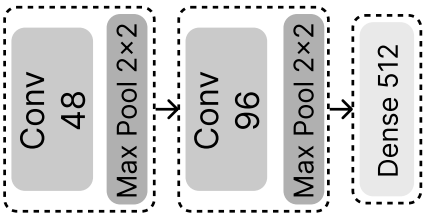

Figure 4.4: Selected CNN Architecture for Fashion MNIST Dataset

4.2.2 Selection of Training Strategies

Selecting an effective training configuration is essential to achieving optimal model performance. In all experiments, categorical cross-entropy was used as the loss function. The remaining training parameters for both datasets were determined through exploratory experiments to select the optimal setup for subsequent experiments. This process involved adjusting optimization algorithms, learning rates, data augmentation levels, and batch sizes.

Selection of Optimization Algorithms and Learning Rates

Different optimization algorithms and learning rates were tested to identify the most suitable combinations for further experiments.

The chosen method for determining the optimal learning rate was manual search, despite its inefficiency and computational expense. This approach involves testing various values and selecting the one that yields the best results. In practice, there are more sophisticated methods for selecting the learning rate, such as automated learning rate schedulers that

provide more efficient solutions. Another advanced method is the cyclical learning rate technique, which involves exponentially increasing the learning rate after each batch and tracking its impact on training loss [40].

The optimization algorithms tested included Adam, AdaGrad, RMSprop, AdamW, Adamax, SGD and NAG. Based on their performance in these experiments, Adam and RMSprop were selected for further analysis with two different learning rates for both datasets. The loss trends on CIFAR-10 and Fashion MNIST are displayed in Figures 4.5 and 4.6, respectively.

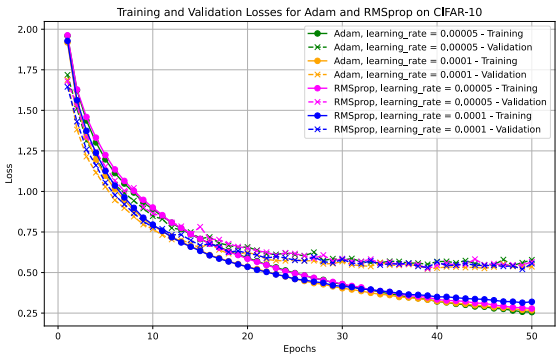

Figure 4.5: Optimizer Loss Curves – CIFAR-10

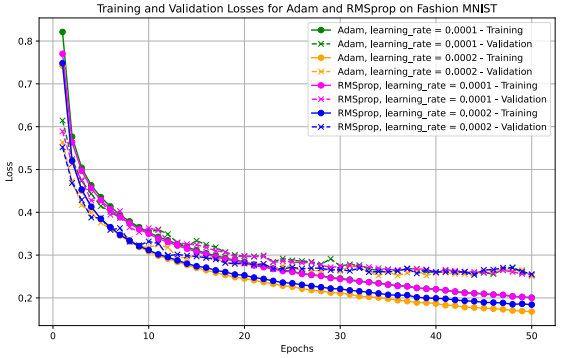

Figure 4.6: Optimizer Loss Curves – F. MNIST

For further experiments, Adam with a learning rate of 0.00005 was chosen for the CIFAR-10 dataset, as it demonstrated better training and validation trends in the final epochs, whereas the effectiveness of a learning rate of 0.0001 slightly diminished toward the later stages. Adam with a learning rate of 0.0002 was selected for the Fashion MNIST dataset.

Selection of Data Augmentation Strategy

In this experiments, different levels of data augmentation were tested to investigate their impact on model performance and resistance to overfitting and to select the appropriate option for further experiments.

Three levels of augmentation strategies were tested:

- **Minor Data Augmentation:** This level included random horizontal flipping and minor rotations (up to 10 degrees).
- **Medium Data Augmentation:** This level included random horizontal and vertical flipping, color jittering (brightness, contrast, saturation, hue), and moderate rotations (up to 25 degrees).
- **Intensive Data Augmentation:** This level included random horizontal and vertical flipping, more extensive color jittering (brightness, contrast, saturation, hue), and large rotations (up to 50 degrees).

The results showed that as the level of data augmentation increased, both the training and validation accuracies decreased compared to models with lower levels of augmentation.

However, the models trained with higher augmentation levels exhibited smaller discrepancies between training and validation accuracies, suggesting that the model was able to generalize better and reduce overfitting. With more epochs, the models with stronger data augmentation are expected to perform better. This effect is clearly illustrated in Figures 4.7 and 4.8.

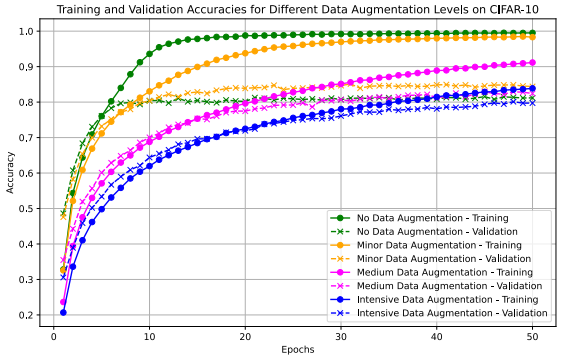

Figure 4.7: Impact of Data Augmentation on Accuracy – CIFAR-10

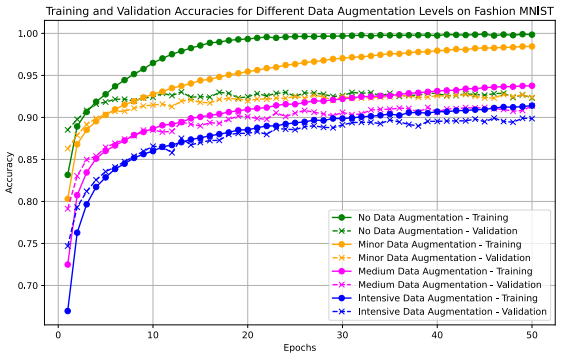

Figure 4.8: Impact of Data Augmentation on Accuracy – F. MNIST

For further experiments, the medium data augmentation strategy was selected for both datasets because it balances improving generalization and maintaining competitive accuracy.

Selection of Batch Sizes

The impact of different batch sizes (25, 50, and 100) on model training dynamics was evaluated. While batch size does not significantly affect model performace, the results indicate that larger batch sizes slightly mitigate overfitting but also slow down the convergence. This effect is illustrated in Figures 4.9 and 4.10.

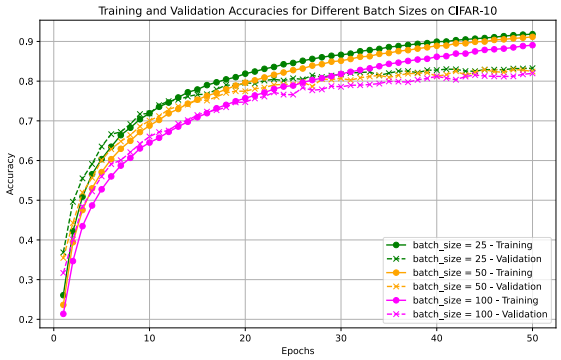

Figure 4.9: Impact of Batch Size on Accuracy – CIFAR-10

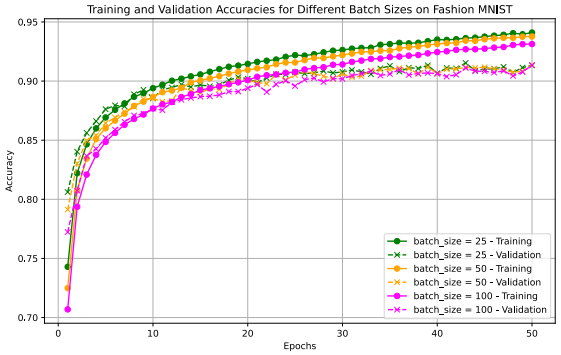

Figure 4.10: Impact of Batch Size on Accuracy – Fashion MNIST

For further experiments, the batch size of 50 was images was selected for both datasets.

4.3 Experiments with Activation Functions

4.3.1 Experiment with Common and Advanced Activation Functions

Motivation and Experimental Design

This experiment involved evaluating the performance of a neural network with the same activation function across all hidden layers, while the output layer utilized the softmax activation function. The activation functions tested included tanh, sigmoid, ReLU, parametric ReLU, ELU, SELU, SiLU, and GELU. The trainable parameter of parametric ReLU function was shared across all neurons within each layer.

Results and Analysis

The accuracy and loss curves for each model for both datasets were plotted to analyze training dynamics and convergence behavior. The accuracies of the models on the CIFAR-10 dataset are shown in Figure 4.11, while the accuracies on the Fashion MNIST dataset are presented in Figure 4.12.

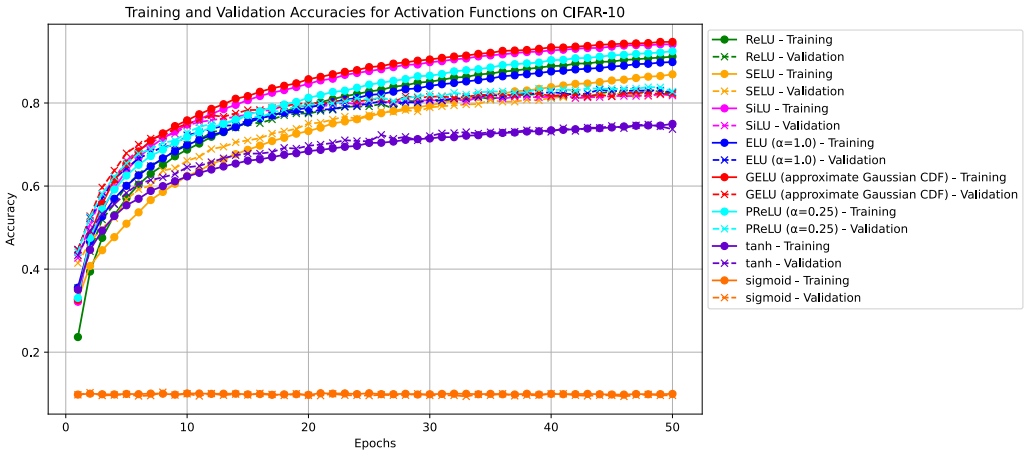

Figure 4.11: Accuracies for Activation Functions – CIFAR-10

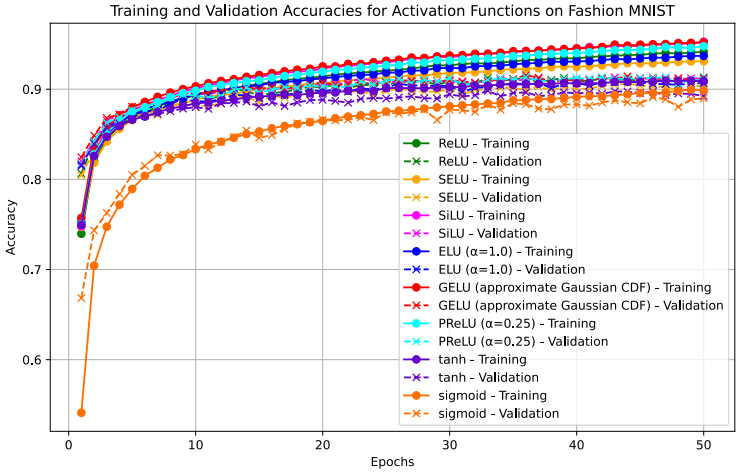

Figure 4.12: Accuracies for Activation Functions – Fashion MNIST

The impact of activation functions on model performance was more pronounced in the more complex dataset and the deeper model architecture. In contrast, the simpler dataset and the shallower model converged quickly, almost independently of the activation function used. One possible explanation is that due to the shallow architecture, gradients do not propagate as deeply, which reduces the influence of the activation function. Additionally, with fewer activation layers, their overall effect on learning dynamics becomes less pronounced. Furthermore, in a simple dataset, the features to be extracted are much simpler, which means that the activation function does not require strong feature extraction capabilities and plays a less critical role.

The model utilizing the sigmoid activation function in all hidden layers failed to train effectively on the CIFAR-10 dataset but was able to learn on the Fashion MNIST dataset. This can be explained by the saturation effect of the sigmoid function, which causes gradients to vanish in deep networks. The problem is stronger when applied to complex datasets like CIFAR-10, which require bigger gradients and more stable gradient propagation through many layers, increasing the risk of vanishing or exploding gradients. This explanation is supported by the mean gradients of the sigmoid function, as illustrated in Figures 4.13 and 4.14

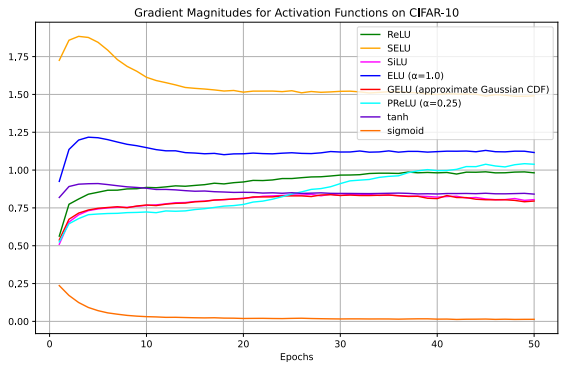

Figure 4.13: Gradient Magnitudes – CIFAR-10

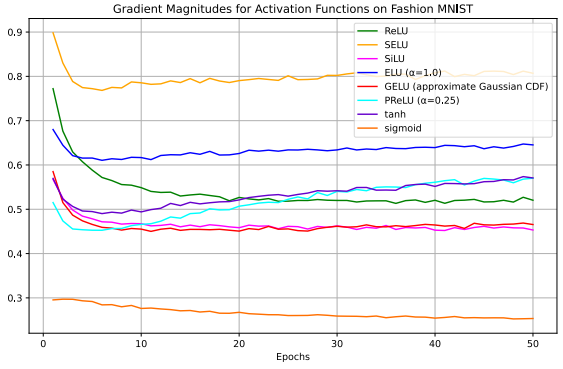

Figure 4.14: Gradient Magnitudes – Fashion MNIST

Among the tested activation functions, tanh and sigmoid demonstrated the slowest convergence rates, and tanh outperformed sigmoid on both datasets. However, these functions exhibited very low overfitting, suggesting better generalization properties. Their slower convergence can be explained by the saturation effect, expressed as small or even vanishing gradients in deeper layers. However, tanh had only a slightly decreasing gradient on CIFAR-10, while on Fashion MNIST the gradient showed increasing trend.

On the other hand, ReLU and activation functions derived from it, such as parametric ReLU, ELU, SELU, SiLU, and GELU, achieved high performance but showed increased overfitting to the training data. All these functions exhibited stable gradients across all epochs, without the vanishing or exploding gradient problem, and the parametric ReLU gradients had a slightly increasing trend.

Additionally, ReLU and its variants, including parametric ReLU, ELU, SiLU, and GELU, demonstrated strong resistance to FGSM attacks. However, SELU performed significantly worse, probably due to its internal normalization mechanism that limits the range of activations. This might have caused the model to be more sensitive to perturbations caused by the FGSM attack. The accuracies of the models after the FGSM attack on the CIFAR-10 dataset are shown in Figure 4.15, and on the Fashion-MNIST dataset in Figure 4.16.

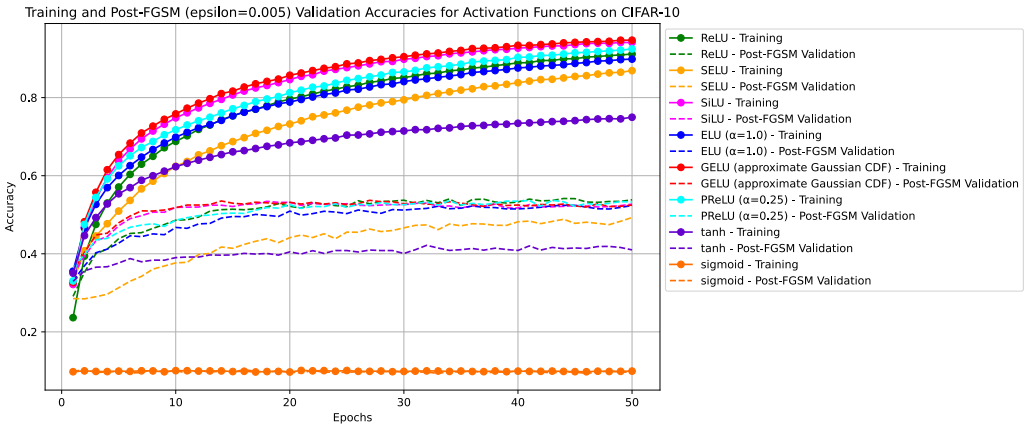

Figure 4.15: Post-FGSM Accuracies for Activation Functions – CIFAR-10

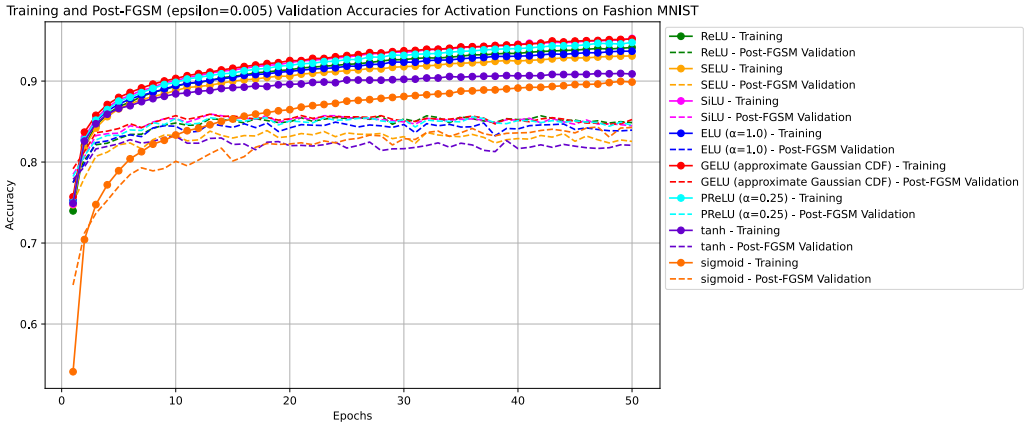

Figure 4.16: Post-FGSM Accuracies for Activation Functions – Fashion MNIST

4.3.2 Experiment with Sinusoidal Activation Function (sin(x))

Motivation and Experimental Design

Periodic functions, such as the sine, are rarely used as activation functions, especially in the context of CNNs. The primary reason is the oscillatory nature of periodic functions, which have infinitely many local minima and can complicate convergence by causing the optimization process to converge to a local minimum.

However, some studies have investigated the potential of $\sin(x)$ as an activation function. In a study by Parascandolo et al., conducted in 2017, it was stated that the sinusoidal function may allow networks to approximate complex functions and enable faster learning compared to traditional monotonic functions in some cases, such as time-series predictions and certain algorithmic tasks. A fully connected model with sin(x) activation in all hidden layers applied to the MNIST dataset also converged effortlessly. However, the network predominantly ignored the periodicity of the sine function and instead utilized its nonperiodic regions, as its performance was similar to that of networks using a truncated sine or the tanh function. [32].

In another study by Ramachandran et al. in 2017, novel activation functions incorporating sinusoidal components, rather than pure sinusoids, were discovered through automated search techniques. The obtained activation functions combined both periodic and nonperiodic elements. These functions were then tested on the CIFAR-10 and CIFAR-100 datasets, and some of them achieved high classification accuracy. These results suggest that while the pure sine function may present challenges for optimization, the integration of sinusoidal components with other characteristics can contribute positively to model performance and enhance complex feature extraction [33].

The purpose of this experiment was to evaluate the behavior of a pure sinusoidal activation function $\sin(x)$ on the CIFAR-10 and the Fashion MNIST datasets and to test to what extent it utilizes the periodicity of the sine function. Similarly to the method used in the research discussed earlier [32], the use of periodicity was evaluated through a comparison of the performance of pure $\sin(x)$ with alternative functions such as a truncated sine $\text{truncsin}(x)$, which limits the periodic range of the sine function, and the monotonic function tanh, which is very similar to $\text{truncsin}(x)$. The $\text{truncsin}(x)$ function is mathematically defined as:

$$\text{truncsin}(x) = \begin{cases} 0, & \text{if } x < -\frac{\pi}{2} \\ \sin(x), & \text{if } -\frac{\pi}{2} \leq x \leq \frac{\pi}{2} \\ 1, & \text{if } x > \frac{\pi}{2} \end{cases}$$

The comparison of the functions described above is shown in Figure 4.17.

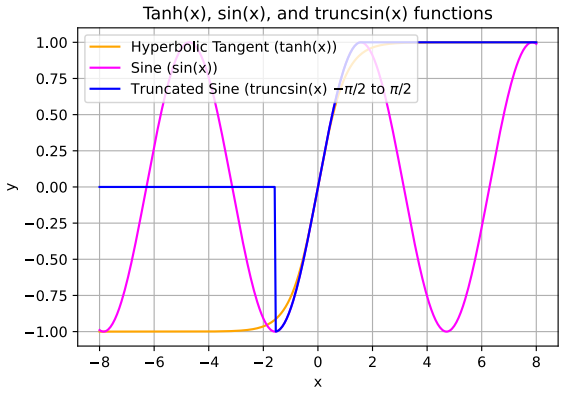

Figure 4.17: Tanh(x), sin(x), and truncsin(x) $-\pi/2$ to $\pi/2$ functions

Results and Analysis

The obtained accuracies are presented in Figures 4.18 and 4.19.

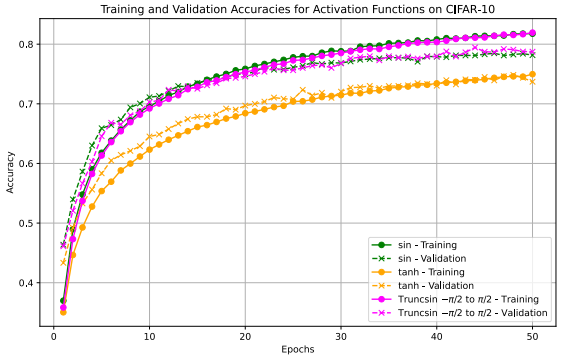

Figure 4.18: Accuracies for Sine and Truncated Sine Functions – CIFAR-10

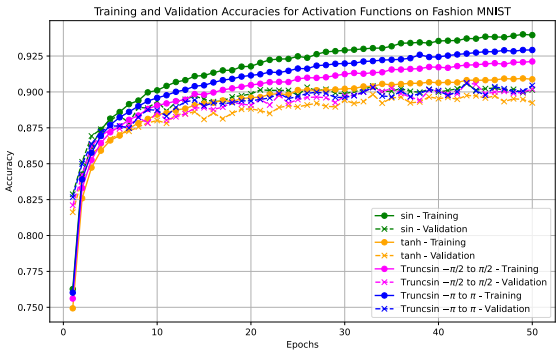

Figure 4.19: Accuracies for Sine and Truncated Sine Functions – Fashion MNIST

The sine function demonstrated good convergence on both datasets and outperformed the tanh function. However, for the CIFAR-10 dataset, the performance of the sine function was very similar to the performance of the truncated version, indicating that the periodicity of the sine function was not utilized effectively. These findings align with previous research, which claimed that the sine function in a fully connected model converged on the MNIST dataset, but did not utilize periodicity [32]. In contrast, on the Fashion MNIST dataset, the periodic sine function outperformed both its truncated version ($[-\pi/2, \pi/2]$) and the tanh function. As a result, the experiment was extended to include another truncated sine function, this time truncated to one full period, as depicted in Figure 4.20. Mathematically, it is defined as:

$$\text{truncsin}(x) = \begin{cases} 0, & \text{if } x < -\pi \\ \sin(x), & \text{if } -\pi \leq x \leq \pi \\ 1, & \text{if } x > \pi \end{cases}$$

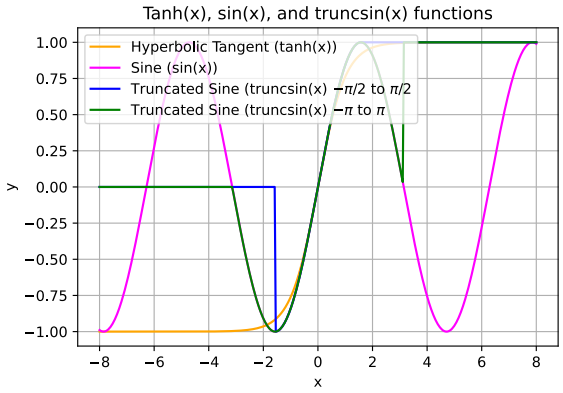

Figure 4.20: Tanh(x), sin(x), truncsin(x) $-\pi/2$ to $\pi/2$, and truncsin(x) $-\pi$ to π

The sine function again slightly outperformed its one-period truncated counterpart, indicating that periodicity was utilized. However, the validation accuracy and loss for all versions of the sine function and its truncated variants eventually saturated, reaching similar values.

The performance under the FGSM attack was less effective in comparison to the activations from the first experiment and the nonperiodic sine activation, as illustrated in Figures 4.21 and 4.22, respectively. This may be caused by the periodicity of the activation function, which potentially makes the activations more predictable and, therefore, easier to manipulate by adversarial perturbations. In contrast, the impact on the Fashion MNIST dataset was less pronounced, as the perturbations were relatively weak for such a simple task, allowing all models to converge effectively even after the attack.

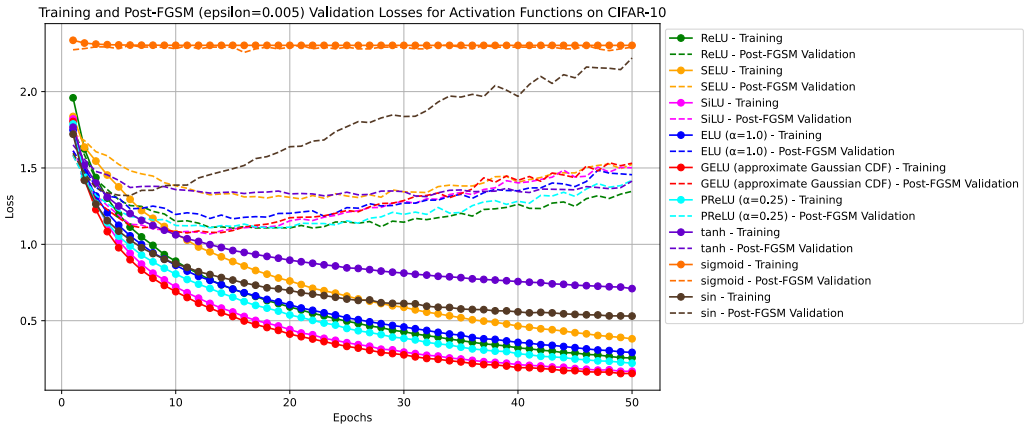

Figure 4.21: Loss Curves for Activation Functions – CIFAR-10

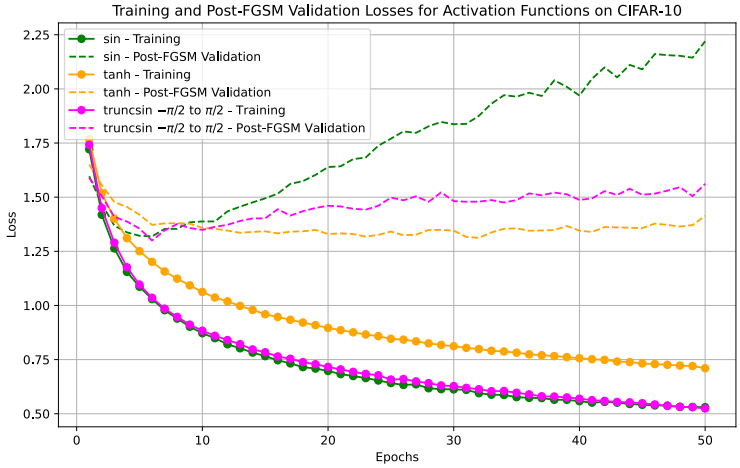

Figure 4.22: Loss Curves for Sine and Truncated Sine Functions – CIFAR-10

The gradients of the sine and truncated sine functions exhibited a strong increasing trend, as shown in Figures 4.23 and 4.24, which could lead to unstable training and exploding gradients at higher epochs.

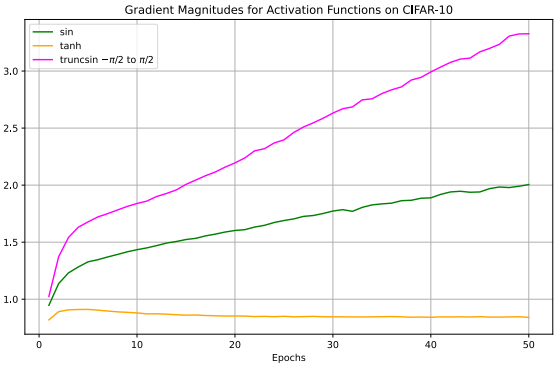

Figure 4.23: Gradient Magnitudes for Sine and Truncated Sine – CIFAR-10

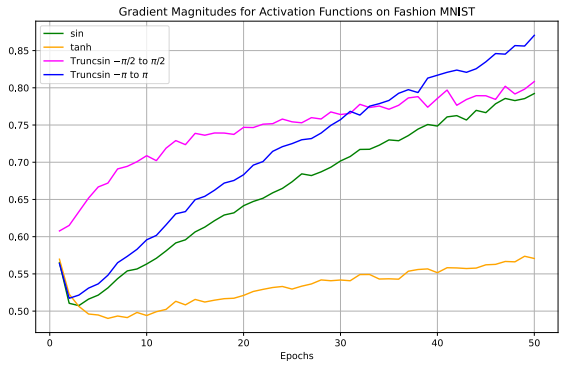

Figure 4.24: Gradient Magnitudes for Sine and Truncated Sine – Fashion MNIST

To address this issue, the experiment was extended to investigate possible solutions for restricting the gradient range. Three techniques were implemented and analyzed for their effectiveness:

- **Gradient Clipping:** This method limits the gradients to a predefined threshold after the gradient computation and before the gradient descent step. Three threshold values were tested.
- **Gradient Scaling:** The input of the activation function is scaled by a predefined factor before applying the sine function.
- **Sine with Batch Normalization:** Batch normalization layers are incorporated after each convolutional and fully connected layer.

All three methods restricted the gradient effectively (Figures 4.25 and 4.26).

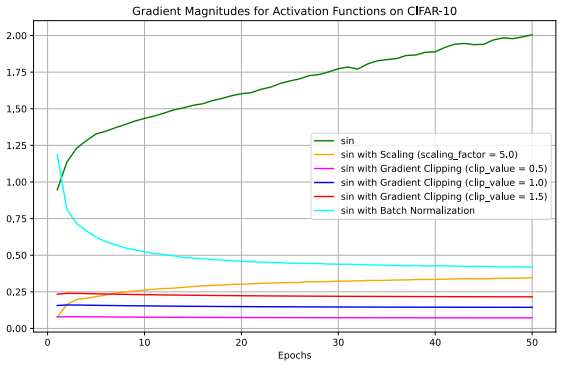

Figure 4.25: Gradient Magnitudes for Sine with Gradient Regulation – CIFAR-10

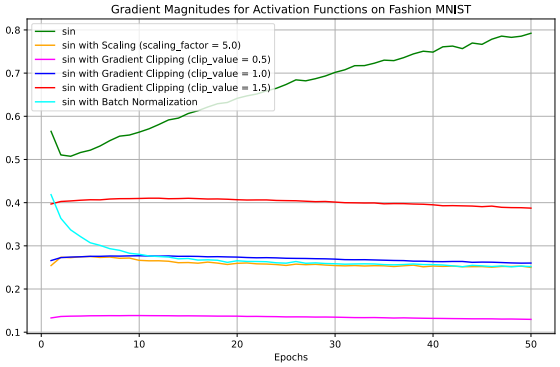

Figure 4.26: Gradient Magnitudes for Sine with Gradient Regulation – Fashion MNIST

The results highlight clear differences in the influence of the gradient regulation techniques on the model's performance. Gradient scaling proved to be the least effective, underperforming the other two methods on both datasets. This suggests that simply rescaling the input to the sine activation may disrupt the model's learning dynamics, making it an unsuitable approach. Meanwhile, models utilizing gradient clipping with all three tested threshold values showed performance nearly identical to that of the pure sine activation without any gradient regulation. In contrast, a model with integrated batch normalization outperformed both gradient clipping and scaling across both datasets, particularly in the later training epochs. Stabilization of activations probably slows down learning slightly in the early epochs, but promotes faster convergence later. These findings are illustrated in Figures 4.27 and 4.28.

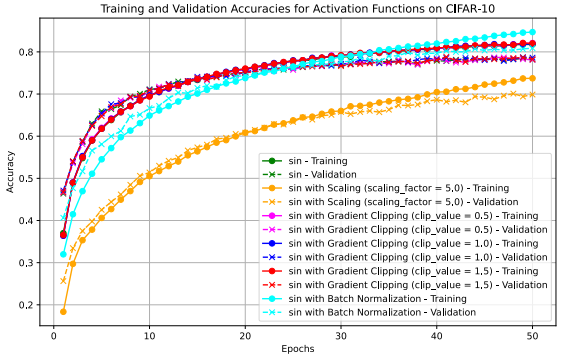

Figure 4.27: Accuracies for Sine with Gradient Regulation – CIFAR-10

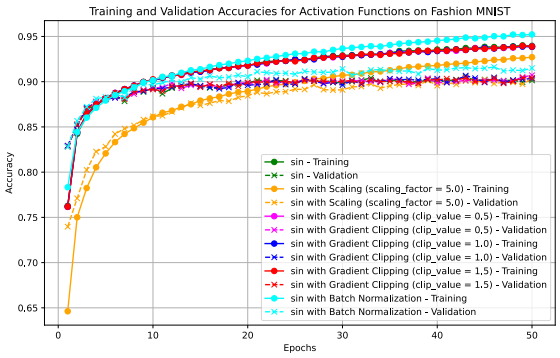

Figure 4.28: Accuracies for Sine with Gradient Regulation – Fashion MNIST

4.3.3 Experiment with Linear Combination Activation Functions with Trainable Parameters ReLU-Sigmoid-Tanh-Linear/-Sin

Motivation and Experimental Design

This experiment involved the implementation of a linear combination (LC) activation function with trainable parameters. This linear combination of multiple activation functions can be achieved by assigning trainable weights to each activation function and computing

a dot product of a vector of weights (initialized as a unit vector at the start of training) and a vector of individual activation functions, divided by the sum of weights:

$$\text{act}(x) = \frac{W \times \text{Acts}}{\sum_{i=1}^{n} w_i} = \frac{[w_1, \ldots, w_n] \times \begin{bmatrix} \text{act}_1(x) \\ \vdots \\ \text{act}_n(x) \end{bmatrix}}{\sum_{i=1}^{n} w_i}$$

Linear combinations of activation functions can potentially improve model performance by dynamically adjusting the contribution of each individual activation function during training, making the model more flexible. A linear combination of multiple activation functions ensures that the network can always replicate any of the simple activation functions by adjusting the corresponding weights. This ensures that it performs at least as well as single activation functions. Furthermore, the gradients of such combined functions are easily computable [28].

As demonstrated in the research by Liao, conducted in 2020, CNNs with trainable linear combination activation functions outperform models with only ReLU on the image classification task on the CIFAR-10 dataset in small, medium, and large models. The activation function used in this study was a linear combination of ReLU, logistic sigmoid, tanh, and linear function:

$$y = \frac{\alpha \cdot \text{ReLU}(x) + \beta \cdot \text{Sigmoid}(x) + \gamma \cdot \text{Tanh}(x) + \delta \cdot x}{\alpha + \beta + \gamma + \delta},$$

where α, β, γ and δ are trainable parameters.

This can be expressed explicitly as:

$$y = \frac{\alpha \cdot \max(0, x) + \beta \cdot \frac{1}{1+e^{-x}} + \gamma \cdot \frac{e^x - e^{-x}}{e^x + e^{-x}} + \delta \cdot x}{\alpha + \beta + \gamma + \delta}, \quad \alpha_0 = 1.0,\ \beta_0 = 1.0,\ \gamma_0 = 1.0,\ \delta_0 = 1.0[28].$$

In this thesis, the same function was implemented to compare this LC function with other common activation functions and evaluate the performance of the model. The ReLU-Sigmoid-Tanh-Linear function is shown in Figure 4.29.

Additionally, the ReLU-sigmoid-tanh-linear activation function was enhanced with both a sinusoidal term and its truncated version for comparative analysis, along with an additional trainable parameter. The idea was to extend the previous experiments and evaluate the impact of a periodic component in a linear combination activation function on the model's performance in an image classification task.

The resulting function with a sine is mathematically defined as:

$$y = \frac{\alpha \cdot \text{ReLU}(x) + \beta \cdot \text{Sigmoid}(x) + \gamma \cdot \text{Tanh}(x) + \delta \cdot x + \epsilon \cdot \sin(x)}{\alpha + \beta + \gamma + \delta + \epsilon},$$

where α, β, γ, δ and ϵ are trainable parameters.

The explicit form is defined as:

$$y = \frac{\alpha \cdot \max(0, x) + \beta \cdot \frac{1}{1+e^{-x}} + \gamma \cdot \frac{e^x - e^{-x}}{e^x + e^{-x}} + \delta \cdot x + \epsilon \cdot \sin(x)}{\alpha + \beta + \gamma + \delta + \epsilon},$$
$$\alpha_0 = 1.0,\ \beta_0 = 1.0,\ \gamma_0 = 1.0,\ \delta_0 = 1.0,\ \epsilon_0 = 1.0.$$

The ReLU-Sigmoid-Tanh-Linear-Sin function is displayed in Figure 4.30.

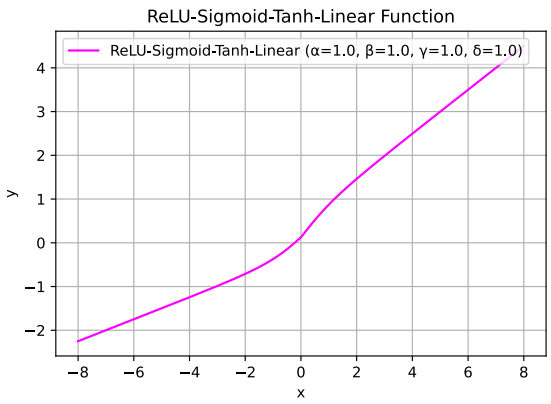

Figure 4.29: ReLU-Sigmoid-Tanh-Linear

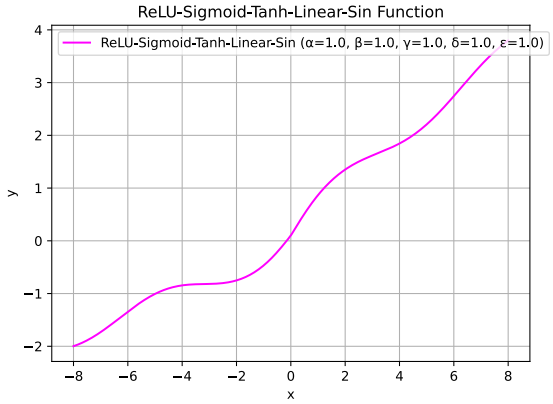

Figure 4.30: ReLU-Sigmoid-Tanh-Lin-Sin

The trainable weights were initialized to units and remained unrestricted, allowing them to reach any values. To assess the impact of activation weight sharing on learning dynamics and generalization ability, three different levels of flexibility were tested:

- **Neuron-Wise:** Each neuron had its own independent set of learnable activation weights, offering maximum flexibility in activation adaptation.
- **Layer-Wise:** Each layer had a set of learnable activation weights, which was shared across all neurons within the same hidden layer.
- **Uniform:** A global set of learnable activation weights was applied uniformly across all neurons in all hidden layers.

Results and Analysis

These obtained accuracies are illustrated in Figures 4.31 and 4.32.

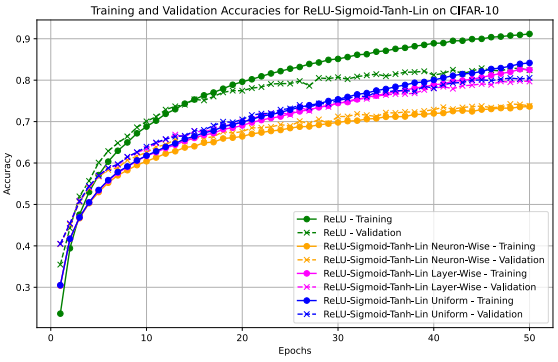

Figure 4.31: Accuracies for ReLU-Sigmoid-Tanh-Linear – CIFAR-10

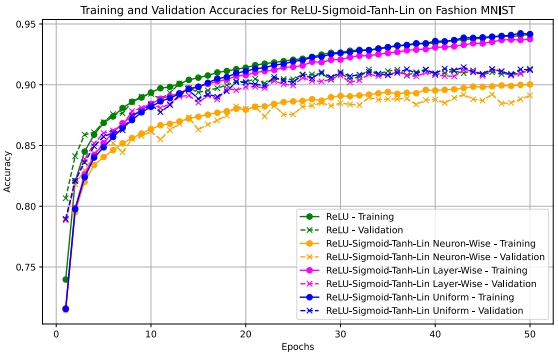

Figure 4.32: Accuracies for ReLU-Sigmoid-Tanh-Linear – Fashion MNIST

The results for the Uniform and Layer-Wise configurations were quite similar on CIFAR-10, showing comparable accuracies, with Uniform slightly outperforming Layer-Wise. Neither configuration was able to surpass the results achieved with ReLU on the CIFAR-10 dataset, although they exhibited much less overfitting compared to ReLU. In contrast,

when applied to the Fashion MNIST dataset, both Uniform and Layer-Wise configurations performed similarly to ReLU, but again did not outperform it.

On the other hand, the Neuron-Wise configuration showed much slower convergence on both datasets, but it exhibited minimal overfitting to the training data.

In general, as the flexibility of the approach increased, the model's resistance to overfitting also increased, though at the cost of slower convergence speed. Given these results, it was decided to increase the number of epochs to 200 for the CIFAR-10 experiments, to provide the models with more training time. The results are shown in Figure 4.33.

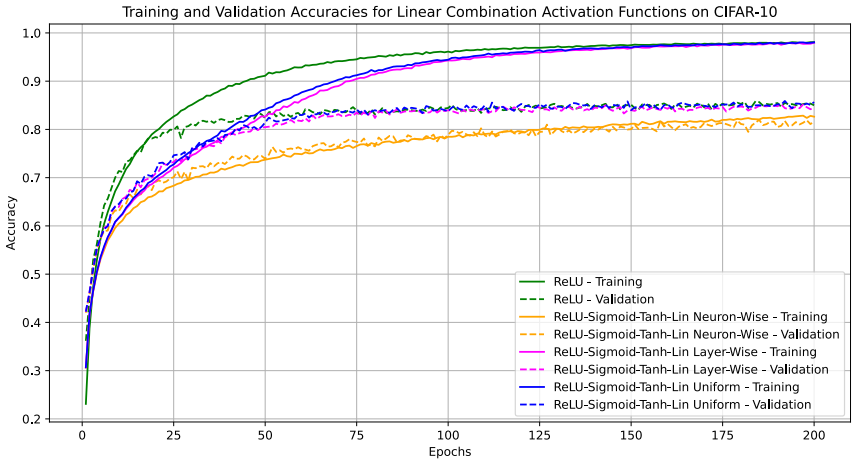

Figure 4.33: Accuracies for ReLU-Sigmoid-Tanh-Linear – CIFAR-10

The Neuron-Wise and Layer-Wise approaches eventually saturated at accuracy levels comparable to ReLU, without surpassing it. The Uniform approach demonstrated minimal overfitting, even in the later epochs. However, its validation accuracy remained lower than that of the other configurations.

The Neuron-Wise configuration exhibited a strong trend of increasing gradient magnitudes in the later epochs, likely due to the high degree of flexibility in learnable activation parameters and the unrestricted nature of the linear function, which allowed gradients to grow without bounds. This effect is illustrated in Figure 4.34. This configuration should be used with gradient regularization techniques to prevent excessive gradient growth.

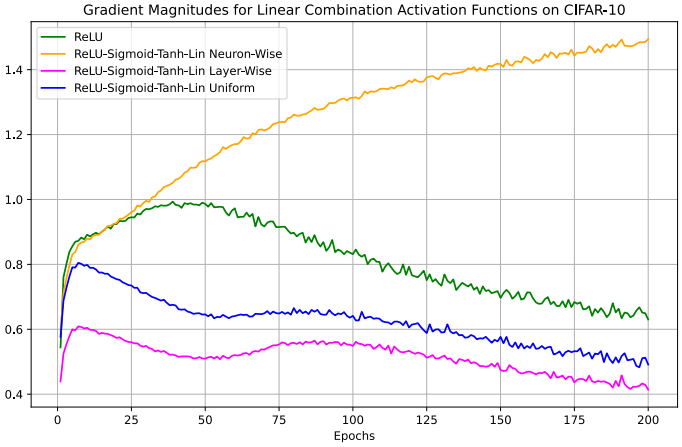

Figure 4.34: Gradient Magnitudes for ReLU-Sigmoid-Tanh-Linear – CIFAR-10

The accuracies obtained from the experiment incorporating sine and truncated sine terms into the linear combination are shown in Figures 4.35 and 4.36.

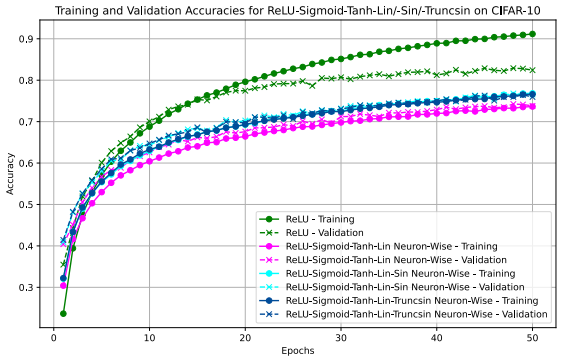

Figure 4.35: Accuracies for ReLU-Sigmoid-Tanh-Linear-Sin – CIFAR-10

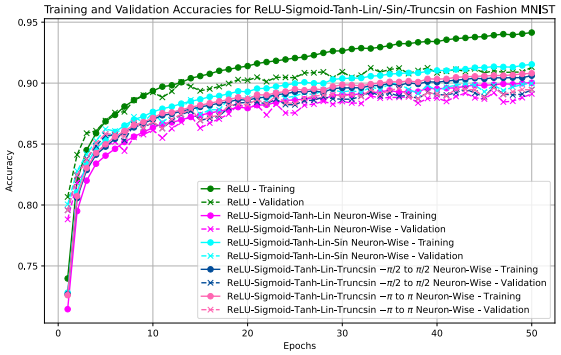

Figure 4.36: Accuracies for ReLU-Sigmoid-Tanh-Linear-Sin – Fashion MNIST

The addition of a sine term slightly improved prediction accuracy and convergence speed on both datasets. As in the previous experiment, the periodicity of the sine function was not effectively utilized on the CIFAR-10 dataset. However, on Fashion MNIST, the function with a full sine term outperformed both of its truncated versions.

Consistent with the previous experiment, where sine as an activation function performed poorly under adversarial attacks, the activation functions incorporating a sine term also exhibited weaker convergence than those without it. These results are illustrated in Figures 4.37 and 4.38.

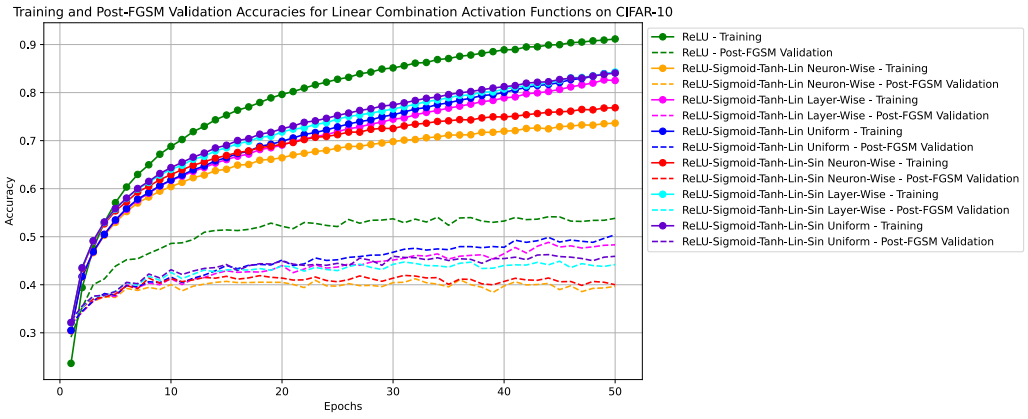

Figure 4.37: Post-FGSM Accuracies for ReLU-Sigmoid-Tanh-Linear/-Sin – CIFAR-10

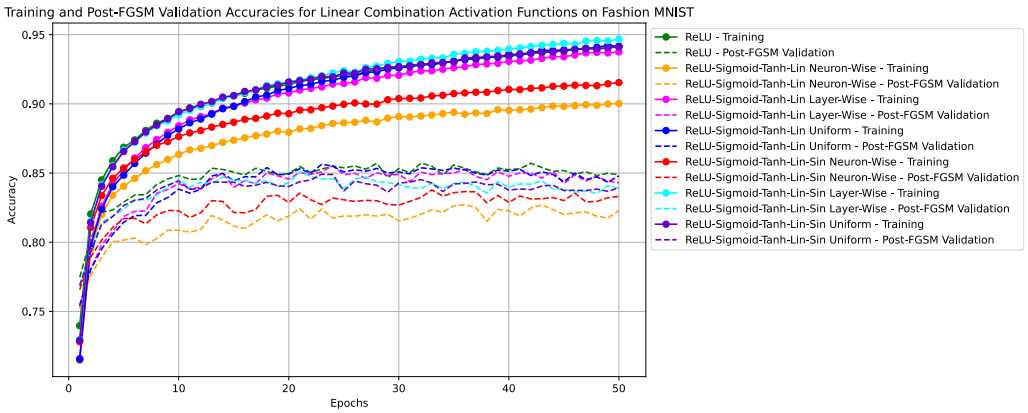

Figure 4.38: Post-FGSM Accuracies for ReLU-Sigmoid-Tanh-Linear/-Sin – F. MNIST

4.3.4 Experiment with LC Activation Function ReLU-Tanh

Motivation and Experimental Design

Continuing the idea of the previous experiments, another LC activation function ReLU-Tanh was tested. This function can combine the efficiency of ReLU with the lower overfitting tendency of tanh, potentially balancing generalization and robustness. This function closely resembles the function from the previous experiment and was among the best-performing in a study by Manessi and Rozza [29], who also researched the LC activation functions. In their work, they automatically combined ReLU, tanh, and linear activation functions during training, comparing the performance of convex (where trainable weights of each activation function are restricted to be non-negative) and affine (where weights are unrestricted) linear combinations across several well-known models and datasets, including Fashion MNIST and CIFAR-10. Their findings demonstrated that LC functions consistently outperformed individual base functions, with the affine approach often outperforming the convex method due to its non-monotonic behaviour.

The mathematical formulation is as follows:

$$y = \frac{\alpha \cdot \text{ReLU}(x) + \beta \cdot \tanh}{\alpha + \beta},$$

where α and β are trainable parameters.

Explicitly, this is expressed as:

$$y = \frac{\alpha \cdot \max(0, x) + \beta \cdot \frac{e^x - e^{-x}}{e^x + e^{-x}}}{\alpha + \beta}, \quad \alpha_0 = 1.0, \beta_0 = 1.0.$$

Figure 4.39 illustrates the ReLU-tanh activation function.

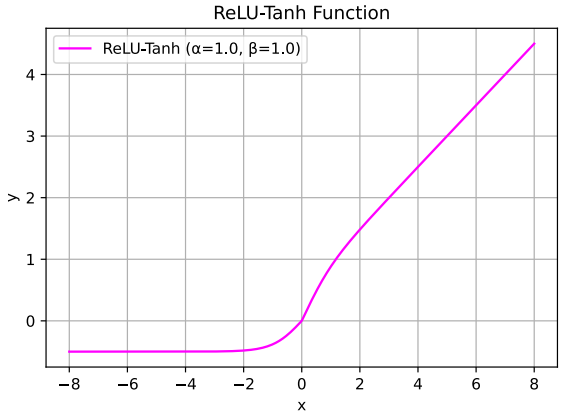

Figure 4.39: ReLU-Tanh Function

Results and Analysis

The obtained accuracies are presented in Figures 4.40 and 4.41.

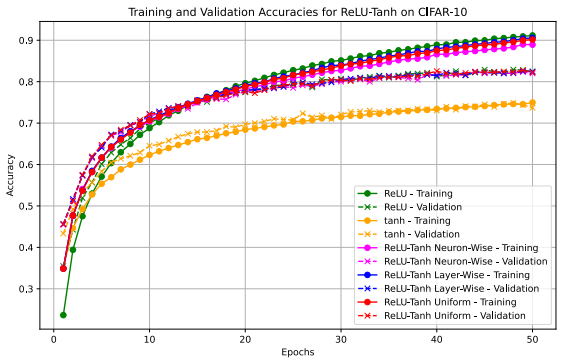

Figure 4.40: Accuracies for ReLU-Tanh – CIFAR-10

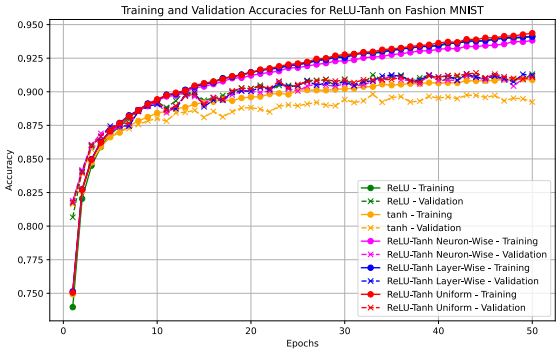

Figure 4.41: Accuracies for ReLU-Tanh – Fashion MNIST

The ReLU-Tanh activation function demonstrated a slightly lower rate of overfitting compared to standard ReLU. However, as in the previous experiment, it did not surpass ReLU in any of the tested configurations on either dataset.

4.3.5 Experiment with Batch Normalization and Self-Normalizing SELU

Motivation and Experimental Design

The objective of this experiment was to examine the internal normalization properties of the SELU function discussed earlier and compare it with the effect of batch normalization incorporated into architectures.

Batch normalization is a method designed to address the challenges caused by the internal covariate shift, which is the fact that the distribution of inputs to each layer changes during training due to parameter updates in previous layers. This phenomenon complicates training and often requires careful initialization and small learning rates. Batch normalization mitigates these issues by normalizing the inputs to each layer within a mini-batch, ensuring that their mean is 0 and variance is 1. This normalization step is integrated into the network architecture. In CNNs, batch normalization is applied to each feature map to adjust the values across all spatial dimensions and mini-batch samples.

In a 2015 research by Ioffe and Szegedy, the findings demonstrated that batch normalization enabled faster convergence, allowed for higher learning rates, stabilized gradient flow, and allowed for the elimination or reduction of dropout and L2 weight regularization without increasing overfitting [22].

It was of interest to compare the effects of explicit batch normalization with the activation function SELU, which inherently includes a form of self-normalization.

Results and Analysis

The comparison of accuracy curves of models with and without batch normalization are presented in Figures 4.42 and 4.43.

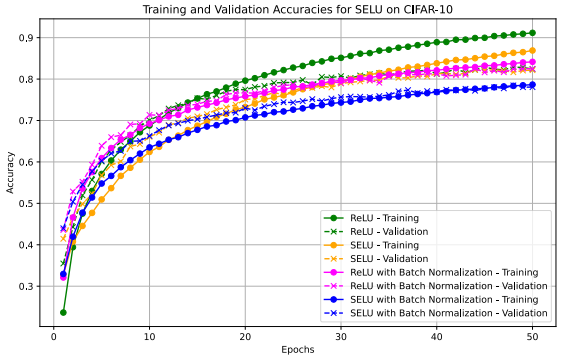

Figure 4.42: Accuracies for SELU – CIFAR-10

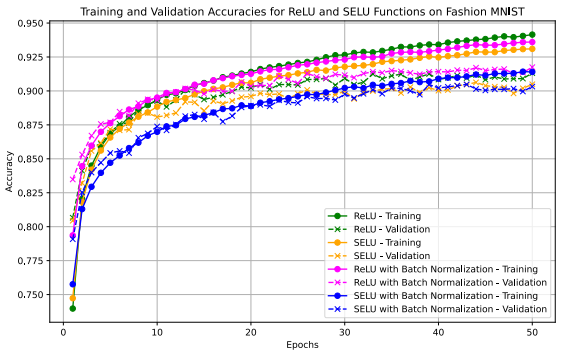

Figure 4.43: Accuracies for SELU – F. MNIST

On the CIFAR-10 dataset, the model utilizing SELU exhibited stable convergence trends, initially progressing more slowly than ReLU but ultimately reaching a comparable validation performance in later stages. The observed effect was in accordance with the previous experiment, where applying batch normalization to a model with a sine activation function produced similar effects. On the Fashion MNIST dataset, however, SELU performed significantly worse. This suggests that in simpler datasets with less complex data features, activation normalization may be unnecessary or even detrimental.

To further investigate the normalization capabilities of SELU, three types of noise were added to the images in the training set of a dataset: Gaussian, Poisson, and uniform. This

experiment aimed to assess whether SELU's self-normalizing properties could help maintain stability in the presence of noisy inputs. The obtained loss trends on CIFAR-10 dataset are shown in Figures 4.44, 4.45, and 4.46, which depict the performance of models in the presence of Gaussian, Poisson, and uniform noise, respectively.

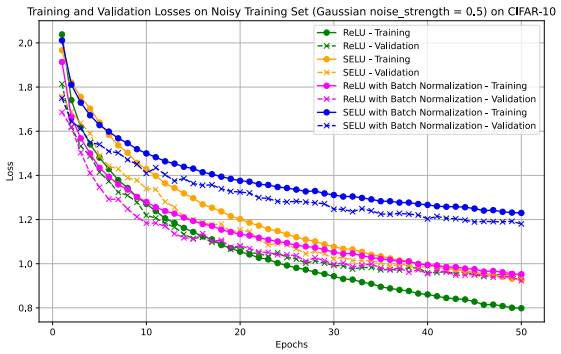

Figure 4.44: Loss Curves under Gaussian Noise – CIFAR-10

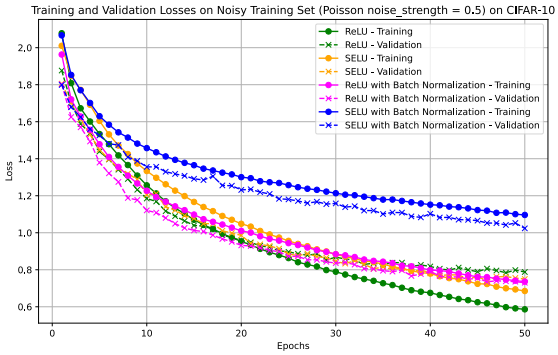

Figure 4.45: Loss Curves under Poisson Noise – CIFAR-10

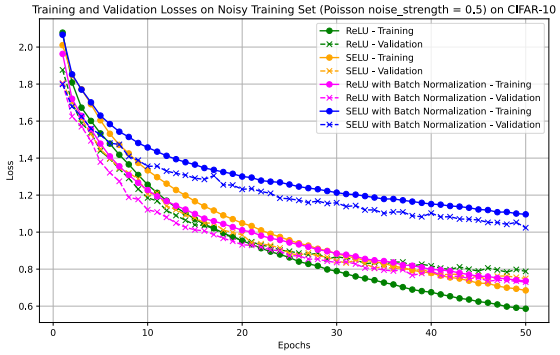

Figure 4.46: Loss Curves under Uniform Noise – CIFAR-10

The results demonstrate that the previously observed effect was further enhanced by adding noise to the images, especially Poisson and uniform, as SELU's inherent self-normalizing properties allowed the network to generalize better to unseen data. This resulted in performance comparable to ReLU with batch normalization. However, applying batch normalization directly to SELU negatively impacted the model's performance. This can be explained by the fact that SELU's self-normalization by driving the mean of activations towards 0 and the standard deviation towards 1 could be disrupted by additional batch normalization.

4.3.6 Experiment with Adding Noise to ReLU for Defense Against Adversarial Attacks

Motivation and Experimental Design

In short preliminary experiments, ReLU demonstrated a relatively strong defense against adversarial attacks. Based on this observation, the idea was to further enhance this capability by adding noise to ReLU.

A study by Byun et al. performed in 2022 indicates that adding small Gaussian noise to the input can effectively disrupt gradient estimation in optimization-based black-box attacks, where attackers estimate the gradient. Even minor Gaussian noise introduces sufficient randomness, complicating the accurate estimation of gradients [4].

For white-box gradient-based attacks, such as FGSM, research also revealed a positive effect of adding noise to input data. A 2022 study by Shi et al. examined the effect of adding Poisson, Gaussian, and uniform noise to images to defend against common adversarial attacks, including FGSM, on both the MNIST and CIFAR-10 datasets. The results revealed that the addition of noise effectively defended against all attacks tested across both datasets. Among the types of noise tested, Poisson noise provided a better defense compared to Gaussian or uniform noise. Furthermore, while the impact of added noise was less noticeable on the MNIST dataset, it was more pronounced and beneficial for the CIFAR-10 dataset [38].

Based on the above, this experiment was designed to test the impact of adding three types of noise, Gaussian, Poisson, and uniform, to the ReLU activation function, unlike previous studies that added noise to images, and analyze how they affect resistance to adversarial attacks. The three types of noise added to ReLU are shown in Figures 4.47, 4.48, and 4.49.

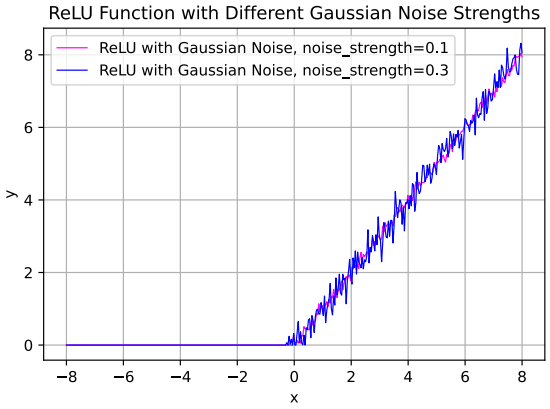

Figure 4.47: ReLU with Gaussian Noise

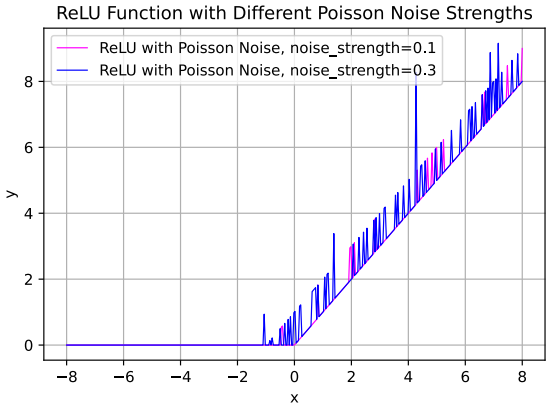

Figure 4.48: ReLU with Poisson Noise

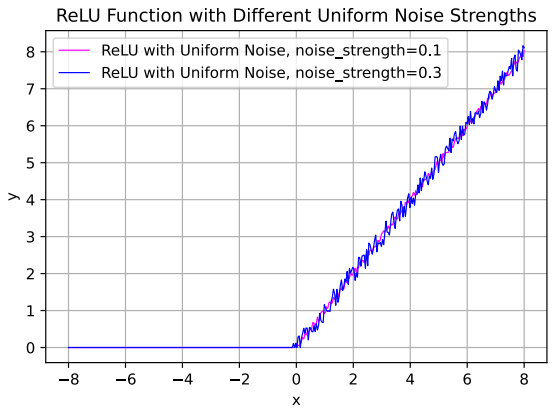

Figure 4.49: ReLU with Uniform Noise

Results and Analysis

Figures 4.50, 4.51, and 4.52 illustrate the loss curves of ReLU with added Gaussian, Poisson, and uniform noise, respectively, on the CIFAR-10 dataset.

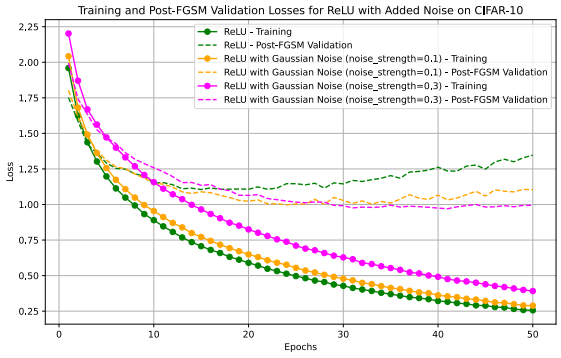

Figure 4.50: Loss Curves for ReLU with Gaussian Noise – CIFAR-10

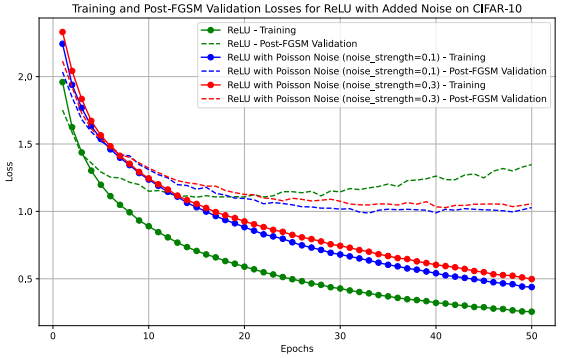

Figure 4.51: Loss Curves for ReLU with Poisson Noise – CIFAR-10

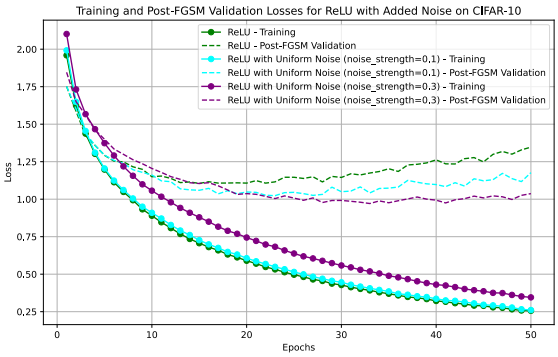

Figure 4.52: Loss Curves for ReLU with Uniform Noise – CIFAR-10

All three types of noise added to ReLU improved convergence under the FGSM attack on the CIFAR-10 dataset. Models using ReLU with Uniform and Gaussian noise demonstrated better generalization as noise strength increased, whereas for Poisson noise, lower noise levels performed slightly better.

For the Fashion MNIST dataset, higher FGSM perturbation strengths were tested, as the models exhibited stable convergence with any activation function under weaker perturbations. The accuracies obtained for ReLU with Gaussian noise on the Fashion MNIST dataset are presented in Figures 4.53, 4.54, and 4.55, corresponding to perturbation strengths of 0.005, 0.01, and 0.02, respectively.

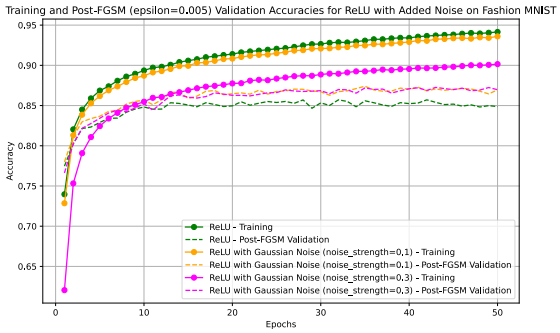

Figure 4.53: Accuracies for ReLU + Gaussian Noise, FGSM 0.005 — F. MNIST

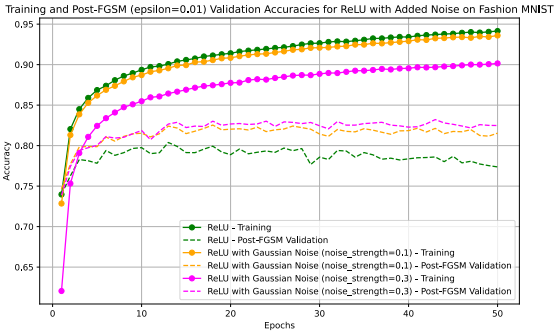

Figure 4.54: Accuracies for ReLU + Gaussian Noise, FGSM 0.01 — F. MNIST

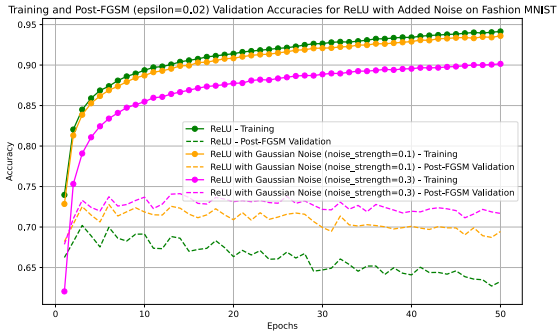

Figure 4.55: Accuracies for ReLU + Gaussian Noise, FGSM 0.02 — F. MNIST

The accuracy curves for ReLU with Poisson noise on the Fashion MNIST dataset are shown in Figures 4.56, 4.57, and 4.58, corresponding to perturbation strengths of 0.005, 0.01, and 0.02, respectively.

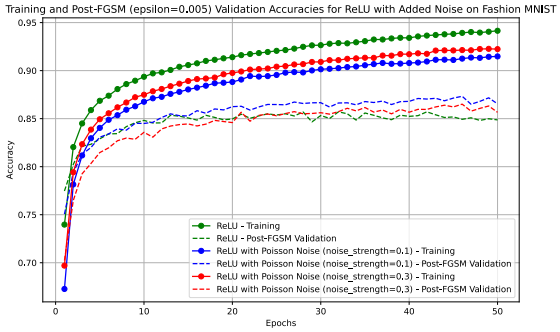

Figure 4.56: Accuracies for ReLU + Poisson Noise, FGSM 0.005 — F. MNIST

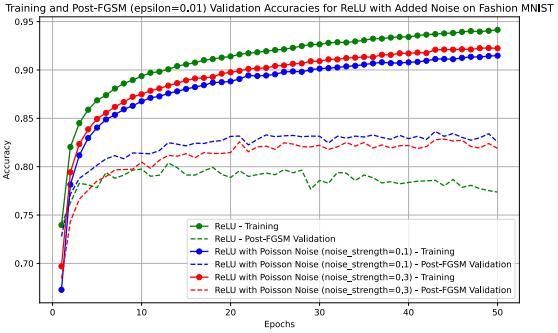

Figure 4.57: Accuracies for ReLU + Poisson Noise, FGSM 0.01 — F. MNIST

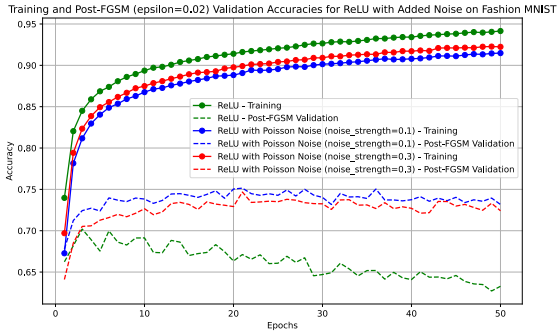

Figure 4.58: Accuracies for ReLU + Poisson Noise, FGSM 0.02 — F. MNIST

The accuracies for ReLU with uniform noise on the Fashion MNIST are shown in Figures 4.59, 4.60, and 4.61.

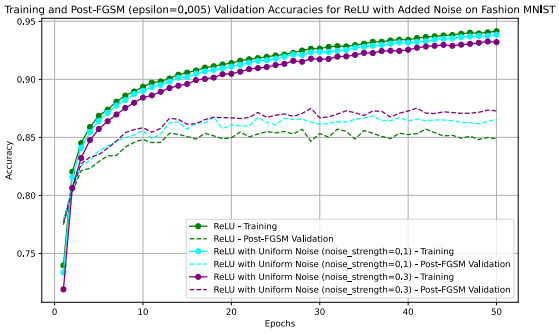

Figure 4.59: Accuracies for ReLU + Uniform Noise, FGSM 0.005 — F. MNIST

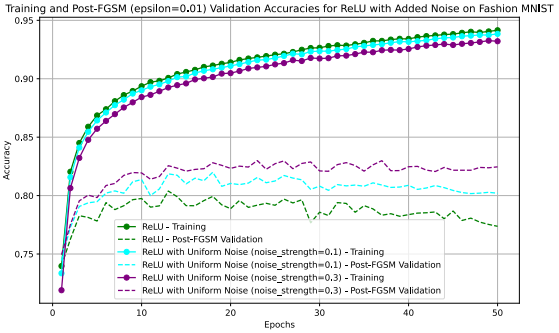

Figure 4.60: Accuracies for ReLU + Uniform Noise, FGSM 0.01 — F. MNIST

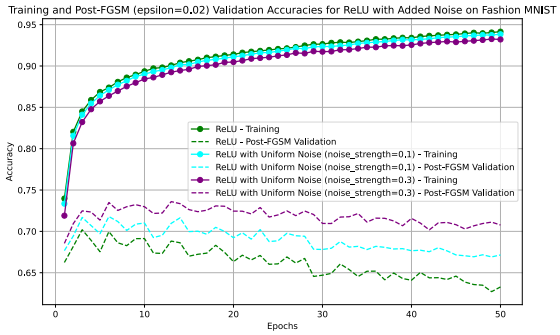

Figure 4.61: Accuracies for ReLU + Uniform Noise, FGSM 0.02 — F. MNIST

As seen from the figures, increasing the perturbation strength led to a growing performance gap between ReLU with all types of noise and standard ReLU. This indicates that adding noise to the activation function enhances resistance to the adversarial attack.

Consistent with the CIFAR-10 results, models using ReLU with uniform and Gaussian noise demonstrated improved generalization at higher noise strengths, whereas models with Poisson noise performed best at lower noise strength. Among all noise types, the models

with Poisson noise achieved the highest performance, with the model at a noise strength of 0.1 performing the best, as shown in Figure 4.62.

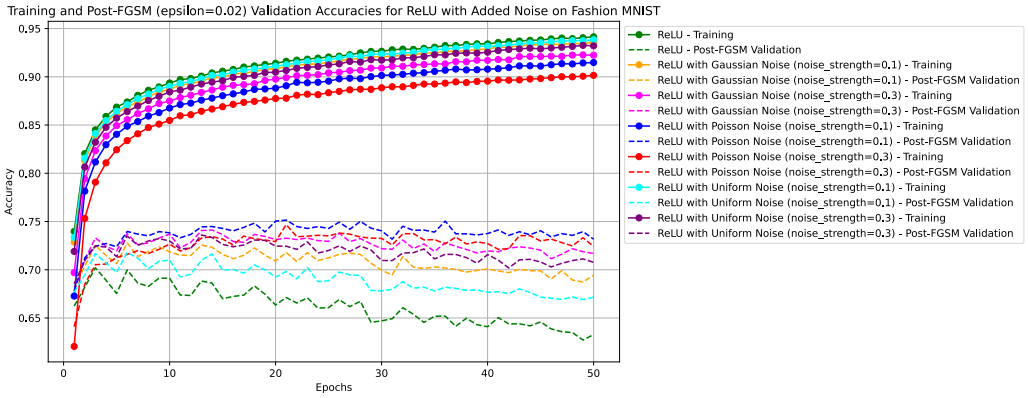

Figure 4.62: Accuracies for ReLU with Added Noise – Fashion MNIST

4.3.7 Experiment with Stochastic Activation Selection for Defense Against Adversarial Attacks

Motivation and Experimental Design

In addition to the previously discussed noise injection to data, several studies suggest that randomness can improve resistance to adversarial attacks. For instance, in a research by Dhillon et al. [9], a method was proposed where a random set of activations (more likely those with smaller magnitudes) is eliminated for each layer during the forward pass, which resulted in improved robustness against adversarial perturbations.

Building upon this evidence, it was hypothesized that introducing randomness into the selection of activation functions could also enhance the model's resistance to FGSM perturbations. This experiment aimed to investigate the effect of randomly selecting the activation function for each neuron from a predefined set during each forward pass. To further increase randomness, the α parameter of the ELU activation function is randomly chosen from a uniform distribution within the range $[0.01, 1.0]$, while the negative slope parameter of leaky ReLU is randomly selected from a uniform distribution within the range $[0.001, 0.1]$.

The activation functions considered in this experiment include:

- ReLU
- ELU, with a randomly selected α parameter in the range $[0.01, 1.0]$
- SiLU
- Tanh
- GELU
- SELU
- Leaky ReLU, with a randomly selected negative slope parameter in the range $[0.001, 0.1]$

Results and Analysis

The accuracy and loss curves obtained on the CIFAR-10 dataset are presented in Figures 4.63 and 4.64.

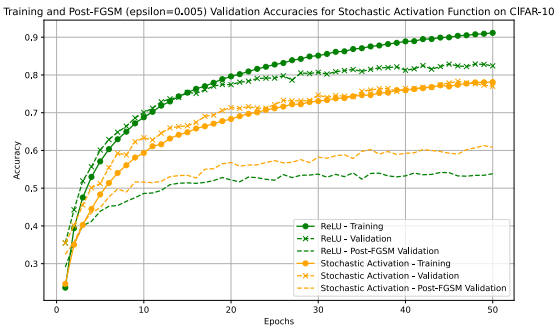

Figure 4.63: Accuracy for Stochastic Activation – CIFAR-10

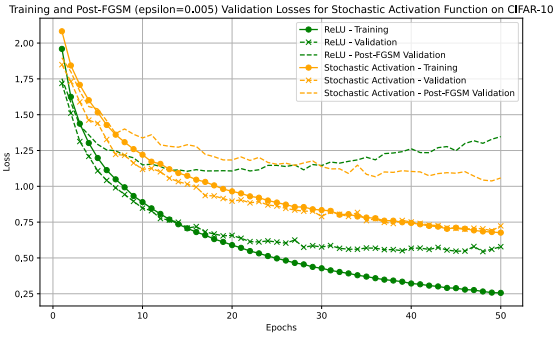

Figure 4.64: Loss for Stochastic Activation – CIFAR-10

The stochastic selection of activation, as expected, led to increased resistance to the FGSM attack. These results align with existing evidence of the positive effect of various forms of randomness on improving robustness against adversarial attacks.

For the Fashion MNIST dataset, higher FGSM perturbation strengths were tested, similarly to the previous experiment. The results are shown in Figures 4.65, 4.66, and 4.67, corresponding to perturbation strengths of 0.005, 0.01, and 0.02, respectively.

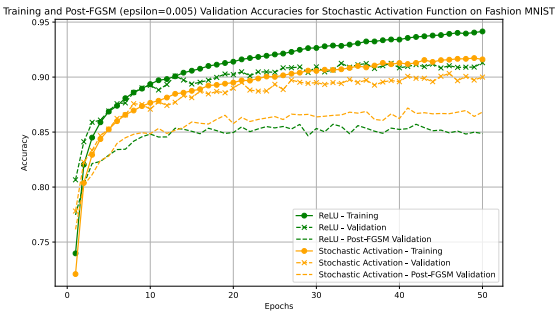

Figure 4.65: Accuracies for Stochastic Activation, FGSM 0.005 — F. MNIST

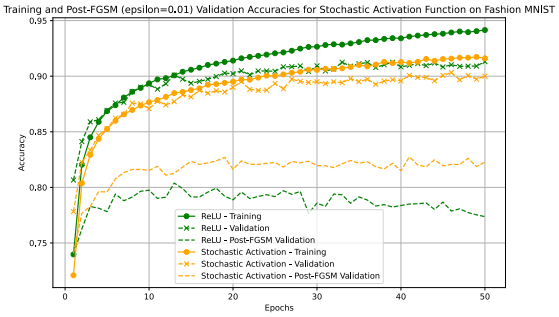

Figure 4.66: Accuracies for Stochastic Activation, FGSM 0.01 — F. MNIST

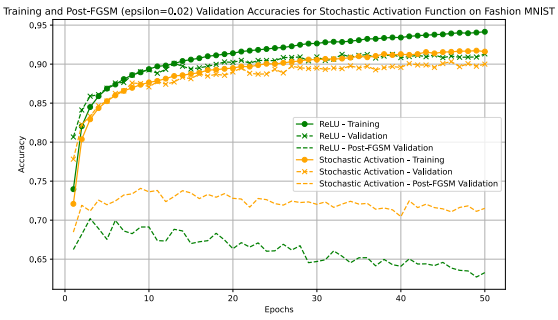

Figure 4.67: Accuracies for Stochastic Activation, FGSM 0.02 — F. MNIST

As in the previous experiment, higher perturbation strengths led to a widening gap between accuracies of the model utilizing ReLU and the model with stochastic activation. This confirms that randomness in activation selection enhances robustness against adversarial perturbations.

Chapter 5

Conclusion

The main objective of this thesis was to study how various activation functions, both widely used and custom-designed, influence the performance of neural networks.

The experiments revealed that the performance of activation functions varies significantly depending on the complexity of the dataset and the architecture of the model. For simpler tasks like Fashion MNIST, the choice of activation function had a less pronounced effect, whereas for more difficult tasks such as CIFAR-10 and deeper architectures, the choice of activation function became essential.

Activation functions like ReLU and its variants exhibited high performance, but with increased risk of overfitting. The sigmoid and tanh functions, although slower to converge, showed promising generalization capabilities. However, the sigmoid function failed to train effectively on the CIFAR-10 dataset due to the vanishing gradient problem.

Using sinusoidal activation functions resulted in effective convergence, yet periodicity was not exploited effectively, particularly on the CIFAR-10 dataset. Moreover, using periodic activations requires the incorporation of gradient regulation techniques, since it has been observed that sinusoidal activation is prone to the problem of exploding gradients.

None of the trainable linear combination activation functions tested, despite experiments with various configuration alternatives, reached the performance levels of a single fixed activation function ReLU.

The SELU with internal normalization performed similarly to ReLU with batch normalization, initially converging slower than the standard ReLU but showing greater stability than ReLU in later epochs. This effect was enhanced when different types of noise were added to the images, highlighting the advantages of using normalization in neural networks.

Adding Gaussian, Poisson, and uniform noise to the ReLU activation function improved its resistance against the FGSM adversarial attack. Among these, Poisson noise provided the most effective defense. This effect was further amplified as the strength of the perturbation increased.

Incorporating randomness in the selection of activation functions during each forward pass also exhibited enhanced resistance to the FGSM attack. As with the noise injection experiment, the effect was further enhanced by increasing the perturbations intensity. These results further confirm that randomness in activations can be a valuable defense mechanism.

The obtained results form a comprehensive picture of how different activation functions affect learning and can be further expanded through more in-depth future research.

Bibliography

[1] *CUDA Toolkit Documentation 12.6 Update 3*. Available at: https://docs.nvidia.com/cuda/index.html. Cited on 2025-01-14.

[2] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer New York, NY, 2006. ISBN 978-0-387-31073-2. Chapter 4.1.7. The perceptron algorithm, pp. 192-196.

[3] Bishop, C. M. and Bishop, H. *Deep Learning: Foundations and Concepts*. Springer, 2024. ISBN 978-3-031-45467-7. Chapter 1.3. A Brief History of Machine Learning pp. 16-21, Chapter 6.2.3. Hidden unit activation functions, pp. 182-185, Chapter 6.4. Error Functions, pp 194-198, Chapter 7.2. Gradient Descent Optimization, pp 213-218.

[4] Byun, J.; Go, H. and Kim, C. On the Effectiveness of Small Input Noise for Defending Against Query-based Black-Box Attacks. In: *Proceedings - 2022 IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022*. Institute of Electrical and Electronics Engineers Inc., 2022, p. 3819–3828.

[5] Chellapilla, K.; Puri, S. and Simard, P. *High Performance Convolutional Neural Networks for Document Processing*. October 2006.

[6] Ciampiconi, L.; Elwood, A.; Leonardi, M.; Mohamed, A. and Rozza, A. *A survey and taxonomy of loss functions in machine learning*. 2024. Available at: https://arxiv.org/abs/2301.05579.

[7] Ciresan, D. C.; Meier, U.; Masci, J.; Gambardella, L. M. and Schmidhuber, J. High-Performance Neural Networks for Visual Object Classification. *CoRR*, 2011, abs/1102.0183. Available at: http://arxiv.org/abs/1102.0183.

[8] Clevert, D.-A.; Unterthiner, T. and Hochreiter, S. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016. Available at: https://arxiv.org/abs/1511.07289.

[9] Dhillon, G. S.; Azizzadenesheli, K.; Lipton, Z. C.; Bernstein, J.; Kossaifi, J. et al. Stochastic Activation Pruning for Robust Adversarial Defense. *CoRR*, 2018, abs/1803.01442. Available at: http://arxiv.org/abs/1803.01442.

[10] Elfwing, S.; Uchibe, E. and Doya, K. Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. *Neural networks : the official journal of the International Neural Network Society*, 2017, vol. 107, p. 3–11.

[11] Florea, A. and Andonie, R. Weighted Random Search for Hyperparameter Optimization. *CoRR*, 2020, abs/2004.01628. Available at: https://arxiv.org/abs/2004.01628.

[12] Fukushima, K. Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements. *IEEE Transactions on Systems Science and Cybernetics*, 1969, vol. 5, no. 4, p. 322–333. DOI 10.1109/TSSC.1969.300225.

[13] Fukushima, K. Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position-Neocognitron-. *IEICE Technical Report, A*, 1979, vol. 62, no. 10, p. 658–665.

[14] Goldberg, Y. A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research*, october 2015, vol. 57. DOI 10.1613/jair.4992.

[15] Goodfellow, I. J.; Shlens, J. and Szegedy, C. *Explaining and Harnessing Adversarial Examples*. 2015. Available at: https://arxiv.org/abs/1412.6572.

[16] Gu, M.; Zhang, Y.; Wen, Y.; Ai, G.; Zhang, H. et al. A lightweight convolutional neural network hardware implementation for wearable heart rate anomaly detection. *Computers in Biology and Medicine*, 2023, vol. 155, p. 106623. ISSN 0010-4825. Available at: https://www.sciencedirect.com/science/article/pii/S0010482523000884.

[17] He, K.; Zhang, X.; Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 770–778. DOI 10.1109/CVPR.2016.90.

[18] Hendrycks, D. and Gimpel, K. *Gaussian Error Linear Units (GELUs)*. 2023. Available at: https://arxiv.org/abs/1606.08415.

[19] Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*, november 1997, vol. 9, no. 8, p. 1735–1780. ISSN 0899-7667. Available at: https://doi.org/10.1162/neco.1997.9.8.1735.

[20] Hossain, M. M.; Talbert, D.; Ghafoor, S. and Kannan, R.-R. *Fawca: A flexible-greedy approach to find well-tuned cnn architecture for image recognition problem*. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2018.

[21] Huang, G.; Liu, Z.; Maaten, L. van der and Weinberger, K. Q. *Densely Connected Convolutional Networks*. July 2017. DOI 10.1109/CVPR.2017.243.

[22] Ioffe, S. and Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. Available at: https://arxiv.org/abs/1502.03167.

[23] Klambauer, G.; Unterthiner, T.; Mayr, A. and Hochreiter, S. *Self-Normalizing Neural Networks*. 2017. Available at: https://arxiv.org/abs/1706.02515.

[24] Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. *Technical report, University of Toronto*, 2009. Available at: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[25] Krizhevsky, A.; Sutskever, I. and Hinton, G. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, january 2012, vol. 25.

[26] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E. et al. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1989, vol. 1, no. 4, p. 541–551. DOI 10.1162/neco.1989.1.4.541.

[27] LeCun, Y.; Bottou, L.; Bengio, Y. and Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, december 1998, vol. 86, p. 2278 – 2324. DOI 10.1109/5.726791.

[28] Liao, Z. Trainable Activation Function Supported CNN in Image Classification. *ArXiv*, 2020, abs/2004.13271. Available at: https://arxiv.org/abs/2004.13271.

[29] Manessi, F. and Rozza, A. Learning Combinations of Activation Functions. *CoRR*, 2018, abs/1801.09403. Available at: http://arxiv.org/abs/1801.09403.

[30] McCulloch, W. S. and Pitts, W. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 1943, vol. 5, no. 4, p. 115–133. DOI 10.1007/bf02478259.

[31] Oh, K.-S. and Jung, K. GPU implementation of neural networks. *Pattern Recognition*, 2004, vol. 37, no. 6, p. 1311–1314. ISSN 0031-3203. DOI https://doi.org/10.1016/j.patcog.2004.01.013.

[32] Parascandolo, G.; Huttunen, H. and Virtanen, T. *Taming the waves: sine as activation function in deep neural networks*. 2017. Available at: https://openreview.net/forum?id=Sks3zF9eg.

[33] Ramachandran, P.; Zoph, B. and Le, Q. V. Searching for Activation Functions. *CoRR*, 2017, abs/1710.05941. Available at: http://arxiv.org/abs/1710.05941.

[34] Rosenblatt, F. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell Aeronautical Laboratory, 1957. Report: Cornell Aeronautical Laboratory.

[35] Rosenblatt, F. *Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, D. C., 1962.

[36] Rumelhart, D. E.; Hinton, G. E. and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 1986, vol. 323, p. 533–536. DOI https://doi.org/10.1038/323533a0.

[37] Sen, J. and Dasgupta, S. *Adversarial Attacks on Image Classification Models: FGSM and Patch Attacks and their Impact*. 2023. Available at: https://arxiv.org/abs/2307.02055.

[38] Shi, L.; Liao, T. and He, J. Defending Adversarial Attacks against DNN Image Classification Models by a Noise-Fusion Method. *Electronics*, 2022, vol. 11, no. 12. ISSN 2079-9292. Available at: https://www.mdpi.com/2079-9292/11/12/1814.

[39] Simonyan, K. and Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, 2014, abs/1409.1556.

[40] Smith, L. N. Cyclical Learning Rates for Training Neural Networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2015, p. 464–472. DOI 10.1109/WACV.2017.58.

[41] Sonoda, S. and Murata, N. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*. Elsevier BV, september 2017, vol. 43, no. 2, p. 233–268. ISSN 1063-5203. Available at: http://dx.doi.org/10.1016/j.acha.2015.12.005.

[42] Srivastava, R. K.; Greff, K. and Schmidhuber, J. Highway Networks. *CoRR*, 2015, abs/1505.00387. Available at: http://arxiv.org/abs/1505.00387.

[43] Steinkrau, D.; Simard, P. Y. and Buck, I. Using GPUs for Machine Learning Algorithms. In: *Proceedings of the Eighth International Conference on Document Analysis and Recognition*. USA: IEEE Computer Society, 2005, p. 1115–1119. ICDAR '05. ISBN 0769524206. Available at: https://doi.org/10.1109/ICDAR.2005.251. DOI 10.1109/ICDAR.2005.251.

[44] Sun, S.; Cao, Z.; Zhu, H. and Zhao, J. A Survey of Optimization Methods from a Machine Learning Perspective. *CoRR*, 2019, abs/1906.06821. Available at: http://arxiv.org/abs/1906.06821.

[45] Szandala, T. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. *CoRR*, 2020, abs/2010.09458. Available at: https://arxiv.org/abs/2010.09458.

[46] Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J. and Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 2818–2826.

[47] Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D. et al. *Intriguing properties of neural networks*. 2014. Available at: https://arxiv.org/abs/1312.6199.

[48] Terven, J.; Cordova Esparza, D. M.; Ramirez Pedraza, A.; Chavez Urbiola, E. A. and Romero Gonzalez, J. A. *Loss Functions and Metrics in Deep Learning*. 2024. Available at: https://arxiv.org/abs/2307.02694.

[49] Waibel, A.; Hanazawa, T.; Hinton, G.; Shikano, K. and Lang, K. Phoneme recognition using time-delay neural networks. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, april 1989, vol. 37, p. 328 – 339. DOI 10.1109/29.21701.

[50] Werbos, P. J. Applications of advances in nonlinear sensitivity analysis. In: *System Modeling and Optimization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, p. 762–770.

[51] Wu, Y. and Liu, L. *Selecting and Composing Learning Rate Policies for Deep Neural Networks*. 2022. Available at: https://arxiv.org/abs/2210.12936.

[52] Xiao, H.; Rasul, K. and Vollgraf, R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*, 2017, abs/1708.07747. Available at: http://arxiv.org/abs/1708.07747.

[53] Zhang, W.; Doi, K.; Giger, M. L.; Wu, Y. C.; Nishikawa, R. M. et al. Computerized detection of clustered microcalcifications in digital mammograms using a shift-invariant artificial neural network. *Medical physics*, 1994, 21 4, p. 517–24.

[54] Zhang, W.; Itoh, K.; Tanida, J. and Ichioka, Y. Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Applied optics*, 1990, 29 32, p. 4790–7.

[55] Zhao, X.; Wang, L.; Zhang, Y.; Han, X.; Deveci, M. et al. A review of convolutional neural networks in computer vision. *Artif. Intell. Rev.*, 2024, vol. 57, p. 99.

Appendix A

Running the Code

The experiments in this project were implemented using Python 3.12.3 and designed to run on systems with an NVIDIA GPU and the CUDA 11.8 toolkit installed.

Dependencies

The core dependencies required to run the code are:

- **PyTorch** — for building and training neural networks (GPU version with CUDA 11.8 support)
- **Matplotlib** — for visualizing results

To install the required packages, the following command should be used:

```
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu118

pip install matplotlib
```

Running Experiments

To run the experiment, the following command should be used:

```
python3.12 experiment_name.py
```

After execution, the averaged results are automatically saved in a text file named `training_results.txt`.