THEORETICAL MACHINE LEARNING (DASC 5420)

PROJECT REPORT

Instructor: Yue Zhang

Team Members:

Arpitha Thippeswamy (T00749833)

Jiayi Zhang (T00752757)

Shashank Manjunatha (T00728166)

Sree Aryan Sathyamurthy Parimala Priyadarshini (T00751318)

Project Title

Mathematical Analysis and Theoretical Trade-offs of Activation Functions in Neural Networks: Convergence Behavior and Gradient Dynamics

TABLE OF CONTENTS

Abstract	3
1.Introduction 1.1 Motivation 1.2 Background	4 4 5
2.Objectives	6
3.Related Work 3.1 Early Activation Functions	6 6 - 7
4. Dataset and Preprocessing	8
5.Methodology 5.1 Mathematical Analysis of Activation Functions 5.2 Empirical Evaluation of Convergence Behaviour 5.3 Gradient Flow and Stability Analysis 5.4 Optimization and Trade-off Comparisons 5.5 Gradient Behaviour and Vanishing Gradients 5.6 Assumptions in Activation Function Analysis 5.7 Computational Complexity	9 9 9-10 10 10 10 – 11 11 11
6. Theoretical Framework 6.1 Rectified Linear Unit (ReLU) 6.2 Leaky ReLU 6.3 Sigmoid (Logistic Function) 6.4 Tanh(Hyperbolic Tangent) 6.5 Swish (SiLU – Sigmoid – Weighted Linear Unit) 6.6 Comparison of Activation Functions 6.7 Derivatives of Activation Functions	12 12 12 – 13 13 – 14 14 15 16 17
7.Results 7.1 Gradient Norm Flow Analysis 7.2 Loss Curve Analysis 7.3 Validation Accuracy Comparison 7.4 Convergence Behaviour	18 18 - 19 19 – 20 21 – 22 22 - 24
8. Discussion and Limitations	24-25
9. Conclusion	26
10. Future Work	27
11. Reference	28
12. Appendices	29-31

ABSTRACT

Activation functions are fundamental building blocks in neural networks by introducing non-linear mappings to enable learning of sophisticated patterns. This project conducts a thorough theoretical and empirical investigation of five common activation functions—ReLU, Leaky ReLU, Sigmoid, Tanh, and Swish—using their mathematical characteristics, gradient properties, and real-world results in deep learning architectures.

Through large experiments conducted on a CNN that was trained using the CIFAR-10 dataset, we quantify significant measures like training stability, convergence speed, gradient flow patterns, and model accuracy. Our observation is that Swish yields improved performance with 88.9% validation accuracy, combining the best of smooth gradient propagation and non-monotonicity without unstable training dynamics.

ReLU and Leaky ReLU perform very well with rapid convergence, but Leaky ReLU is more stable against the "dying ReLU" problem. But regular functions like Sigmoid and Tanh suffer from very high vanishing gradient limitations and slow convergence, particularly in deeper network architecture. The paper also examines computational trade-offs and determines that though Swish works better, it is computationally costlier than straightforward functions such as ReLU.

The findings provide practitioners with data-driven guidelines on how to select activation functions best meeting theoretical properties, training effectiveness, and model performance depending on the deep learning tasks. The paper ends by proposing future research avenues including examining adaptive activation functions and their optimization for certain neural architectures.

Future work will explore adaptive activation functions, task-specific optimizations, and potential enhancements to reduce the computational cost of Swish while maintaining its benefits in deep learning tasks.

1.INTRODUCTION

Activation functions are included in neural networks, bringing in non-linearity so that complicated patterns may be represented. Without them, neural networks would be linear models of very weak expressive power. This project, both theoretically and empirically, talks about five of the most popular activation functions—ReLU, Sigmoid, Tanh, Leaky ReLU, and Swish. **The work focuses on:**

- Model Performance: Accuracy and loss metrics.
- Gradient Dynamics: Stability of backpropagation.
- Convergence Behavior: Training effectiveness.

By training a CNN on CIFAR-10 using all the activation functions, we rigorously test their theoretical properties and empirical behavior. We gain insights in this work about their effectiveness, advantages, and disadvantages, adding to the overall body of knowledge for choosing an activation function in deep networks.

The deployment uses PyTorch, with modular code for model training (trainer.py), data loading (utils.py), and architecture definition (model_builder.py). Through the synergy of theoretical analysis (derivative properties, output ranges) and empirical verification (training curves, accuracy metrics), this work provides actionable advice for selecting activation functions in practical deep-learning applications

1.1 MOTIVATION

Activation functions are central to neural networks, controlling their ability to learn complex patterns. However, activation functions exhibit different behaviors when subjected to gradient flow (exploding/vanishing gradients) and convergence speed, which affect model performance materially. While ReLU is ubiquitous, more recent variants like Swish promise improved performance, and older stalwarts like Sigmoid/Tanh suffer from saturation.

This project will:

- Evidentiarily confirm theoretical claims about activation functions.
- Compare their performance in training stability and accuracy in practice.
- Guide practitioners in selecting the right function for deep learning tasks.

1.2 BACKGROUND

- **Convolutional Neural Network (CNN):** CNNs are deep learning models designed specifically for processing grid-like data such as images.

  They consist of three main types of layers:
  1. **Convolutional Layers:** Extract features from images by applying filters.
  2. **Pooling Layers:** Reduce spatial dimensions to improve computational efficiency.
  3. **Fully Connected Layers:** Perform classification based on extracted features.

- **Activation Functions:** Activation functions introduce non-linearity into the model. The implemented CNN allows flexibility in choosing an activation function such as:
  - **ReLU (Rectified Linear Unit):** f(x) = max (0, x), commonly used in CNNs.
  - **Sigmoid/Tanh:** Used in some cases but prone to vanishing gradients.

- **Backpropagation and Optimization:** During training, the network learns and adjusts its weights with backpropagation. It involves:

  1. Compute loss based on a criterion (e.g., Cross-Entropy Loss for classification).
  2. Gradient calculation through differentiation.

Weight updates with an optimiser (e.g., Adam, SGD). This project is all about diving into the mathematical properties of the most used activation functions, like ReLU, Sigmoid, and Tanh, and exploring their impact on gradient propagation during training. One of the key goals is to understand how these different activation functions influence the speed at which a model converges, as well as issues like vanishing or exploding gradients and the stability of backpropagation.

The project will also contrast activation functions according to applicability in deep networks and efficiency in optimization. By empirical experimentation and theoretical investigation, the study aims to provide insights into selecting the optimal activation functions for improved training dynamics and generalization performance in deep learning models.

2.OBJECTIVES

The aim of this project is to examine mathematical foundations and theoretical trade-offs for activation functions of neural networks particularly in terms of their function for convergence behavior as well as for gradient dynamics. Activation functions play a crucial role in injecting non-linearity, so that neural networks can learn sophisticated patterns.

This project is all about diving into the mathematical properties of the most used activation functions, like ReLU, Sigmoid, and Tanh, and exploring their impact on gradient propagation during training. One of the key goals is to understand how these different activation functions influence the speed at which a model converges, as well as issues like vanishing or exploding gradients and the stability of backpropagation. The project will also contrast activation functions according to applicability in deep networks and efficiency in optimization. By empirical experimentation and theoretical investigation, the study aims to provide insights into selecting the optimal activation functions for improved training dynamics and generalization performance in deep learning models.

3.RELATED WORK

Evolution of activation functions has evolved significantly in overcoming fundamental deep learning problems. Early functions, e.g., **Sigmoid and Tanh**, contributed non-linearity but suffered from vanishing gradients, thereby limiting their capacity to function well in deep networks. **ReLU (Rectified Linear Unit)** provided the innovation that enabled convergence to accelerate and gradients to be less of a concern for positive inputs, while the lack of handling negative values resulted in the "dying ReLU" issue. Variants like **Leaky ReLU** accomplished this by adding minor gradient for the negative inputs, rendering it more resilient. Subsequently, **swish (SiLU)** arrived as a smooth, non-monotonic alternative, inheriting characteristics of both sigmoidal and ReLU—like to find maximum performance in deep models. All these advancements reflect a continuous pursuit towards a balance between computational efficiency, gradient stability, and model expressiveness in neural networks.

3.1 Early Activation Functions

Activation functions were originally inspired by the Heaviside step function, which is a binary neuron activation assumption –either active (1) or not active (0). In a effort to add differentiability, the Sigmoid function, defined as

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

was everywhere in early neural networks due to its universal approximation capability. But it suffers from the vanishing gradient problem, where gradients approach zero as input values approach extreme values ($\pm\infty$). This inhibits weight updates for deeper layers, slowing down or even killing training.

It was superseded by the Hyperbolic Tangent (Tanh) function

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

was suggested as a replacement. Sigmoid is not zero-centered, but Tanh is zero-centered and produces outputs between (-1,1), which maintains the activations in zero-centered distributions. This results in improved weight updates faster convergence in deep networks.

With deeper models, Rectified Linear Unit (ReLU) was suggested as a better activation function, which is defined as,

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

ReLU solves the vanishing gradient problem for positive inputs and speeds up training. It does not suffer from the "dying ReLU" issue, in which neutrons produce a zero for negative inputs to fail to update weights.

To address this shortcoming, Leaky ReLU adjusts ReLU so that it leaves a tiny non-zero gradient (α) when the input is negative.

$$f(x) = f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

Where α is a small constant (e.g., 0.01). This ensures that negative inputs are used for learning, preventing dead neurons.

More recent improvement, Swish, defined as:

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

exposes self-gating where the negative inputs are not completely blocked but scaled by their magnitude. Swish, as opposed to ReLU and Leaky ReLU, is smooth and non-monotonic and therefore able to hold small negative values adaptively. This enhances the gradient flow and therefore deep network performance.

Briefly, while Sigmoid and Tanh were crucial in initial networks, modern-day deep learning networks are inclined towards the employment of ReLU, Leaky ReLU, and Swish to achieve faster convergence and improved gradient flow.

4.DATASET AND PREPROCESSING

Dataset Description

The data set used in the project is CIFAR-10, which is a standard image classification benchmark data set. It consists of 60,000 color images with size 32x32 pixels and is classified into 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. There are equal numbers of samples in each class since the data set is balanced. The training data set includes 50,000 images and the test data set includes 10,000 images.

Data Preprocessing

During the preprocessing stage, we take several steps to boost the model's performance and make training more efficient.

Normalization: we adjust the images to have a unit variance and a zero mean. This helps stabilize the learning process and speeds up convergence.
Data Augmentation: By using techniques like random flipping and cropping, we can artificially enlarge the training set, which ultimately improves the model's ability to generalize.
Data Loading: PyTorch's torchvision.datasets and DataLoader are utilized to load and preprocess images effectively in mini-batches, achieving full GPU utilization during training.

These preprocessing steps allow the CNN to acquire stronger features without encountering issues such as overfitting and poor generalization.

Exploratory Data Analysis (EDA): To better understand the dataset, the following analyses were performed:

- **Class Distribution Analysis:** Ensuring that the dataset is evenly distributed across all classes.
- **Statistical Summary:** Mean, standard deviation, and pixel intensity histograms were examined.
- **Visualization:** Sample images from each class were plotted to inspect variability and distinguishability.
- **Correlation Analysis:** Pixel intensity correlations across different classes were analyzed.

5.METHODOLOGY

This project employs a systematic approach to study the mathematical foundations and theoretical trade-offs of activation functions in neural networks, i.e., their impact on convergence behavior and gradient dynamics. The approach integrates theoretical analysis, experimental evaluation, and visualization techniques to systematically investigate the impact of different activation functions on deep learning models.

5.1 Mathematical Analysis of Activation Function

To establish a theoretical foundation, we derive and analyze the mathematical properties of commonly used activation functions, including:

- **ReLU (Rectified Linear Unit):** f(x)=max (0, x)
- **Sigmoid:** $\boldsymbol{f(x)}=1/e-x1$
- $\boldsymbol{Tanh}$: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **Leaky ReLU:** $\boldsymbol{f(x)} = \begin{cases} \boldsymbol{x}, & \boldsymbol{x > 0} \\ \boldsymbol{ax}, & \boldsymbol{x \leq 0} \end{cases}$
- **Swish:** $\boldsymbol{f(x)} = \boldsymbol{x} \cdot \boldsymbol{\sigma(x)} = \boldsymbol{x} \cdot \frac{\mathbf{1}}{\mathbf{1}+e^{-x}}$, where $\sigma(x)$ is the sigmoid function

The following characteristics are taken into consideration for each activation function:

- First derivative to understand their impact on gradient propagation.
- Their behaviour for large or small input values.
- Possibility of causing vanishing or exploding gradients in deep networks.

This mathematical analysis helps to identify trade-offs between activation functions in terms of learning efficiency and stability.

5.2 Empirical Evaluation of Convergence Behavior

To see how activation functions affect CNN model, we perform a comparison using different activation functions. We train the model with the CIFAR-10 dataset. While doing this, we look a few important things.

1. **Training Loss and Validation Loss Curves:** These are graphs that show how well the model is learning. We check how quickly the network is improving and getting better.
2. **Training Time Per Epoch:** This tells us how long it takes to train the model for one complete cycle of the dataset with each activation function. We can see which one is faster and uses less computer power.

3. **Final Validation Accuracy:** This measures how well the model predicts correctly on new and unseen data at the end of training. Our goal is to explore how the choice of activation functions changes the speed, efficiency and reliability of training the model.

The goal is to determine how different activation functions impact the speed and stability of training.

5.3 Gradient Flow and Stability Analysis

In this project, we explore how different activation functions affect the behavior of gradients, which are crucial during the training of a network. Here's what we did:

1. We calculated the size of gradients at each layer. This shows us if the gradients are becoming too tiny(vanishing) or if they are growing too large(exploding), which can hinder learning.
2. We monitored the size of the gradients over several learning cycles, called epochs. This helps confirm that the learning process remains stable over time.
3. We examined how gradients are distributed throughout the layers and compare the impact of various activation functions on this distribution. By doing this analysis, we gain a better understanding of how activation functions can affect the stability and effectiveness of network training.

5.4 Optimization and Trade-off Comparisons

Activation functions also affect the efficiency of optimization algorithms. For comparison purposes, the project considers.

1. The convergence rate of different activation functions via the loss curves.
2. The computational expense of different activation functions, such as additional operation in Sigmoid and Tanh.
3. The learning stability by avoiding some activation functions inducing instable weight updates.

This comparison aids in selecting an activation function which maximizes the convergence speed, computational complexity and stability.

5.5 Gradient Behavior and Vanishing Gradients

The derivative of the activation function affects the training dynamics. The vanishing gradient problem occurs when:

$$| f'(x) | \approx 0$$

- Sigmoid and Tanh suffer from vanishing gradients due to small derivatives for large or small x
- ReLU and Leaky ReLU mitigate vanishing gradients because $f'(x)$ is constant. (1 for positive inputs)

5.6 Assumptions in Activation Functions Analysis

1. Continuity Assumption: Assumes activation functions are continuous or piecewise continuous, which ensures proper gradient computation.
2. Bounded and Unbounded Output: Some function like Sigmoid are bounded in [0,1] affecting representation learning.
3. Zero centered vs. Non-Zero Centered: Functions like Tanh are zero-centered, leading to better weight updates. ReLU and Swish are non-zero centered, which correct weight initialization is required.
4. Gradient Flow Assumption: Assumes that activation functions permit smooth gradient flow without saturation or instability.

5.7 Computational Complexity

The computational cost of an activation function depends on mathematical operations involved:

- ReLU and Leaky ReLU require only a simple comparison, making them computationally efficient ($O(1)$)
- Swish and Sigmoid involve exponentiation ($O(n)$), which is more computationally expensive.

By combining mathematical analysis, empirical evaluation, gradient tracking and visualization, this project provides a comprehensive study of activation functions in neural networks. The methodology allows for theoretical function selection for deep learning applications.

6.THEORETICAL FRAMEWORK

This section provides a theoretical analysis of five key activation functions - ReLU, Leaky ReLU, Sigmoid, Tanh, and Swish focusing on their mathematical properties and its first derivative graphs.

6.1 Rectified Linear Unit (ReLU)

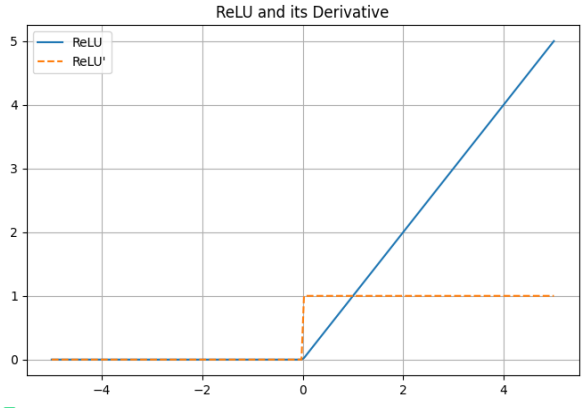
ReLU (Rectified Linear Unit) is the most widely used activation function in deep learning. It is defined as:

$$ReLU(x) = max(0, x)$$

First Derivative:

$$ReLU'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

- Outputs x if $x > 0$, else 0.
- Used in deep learning (CNNs, MLPs) for its simplicity and speed.



6.2 Leaky ReLU

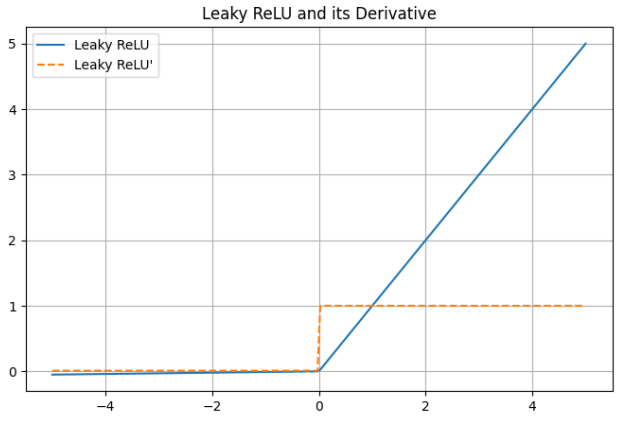
Leaky ReLU (Leaky Rectified Linear Unit) is an improved version of the standard ReLU activation function. It modifies ReLU by introducing a small, non-zero gradient for negative inputs to prevent the "dying ReLU" problem

$$LeakyReLU(x) = f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

First Derivative:

$$LeakyReLU'(x) = f(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x \leq 0 \end{cases}$$

- Adds a small slope (α) for negative inputs to avoid dead neurons.
- Used in binary classification (output layer).



6.3 Sigmoid (Logistic Function)

The sigmoid function (also called the *logistic function*) is a smooth, S-shaped activation function that maps any real-valued input to a value between 0 and 1.

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

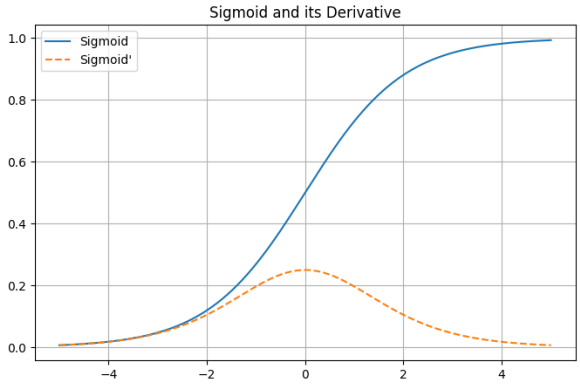
where:

- x = input value
- e = Euler's number (~2.71828)

First Derivative:

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

- Squashes outputs between 0 and 1.
- Used in binary classification (output layer).



6.4 Tanh (Hyperbolic Tangent)

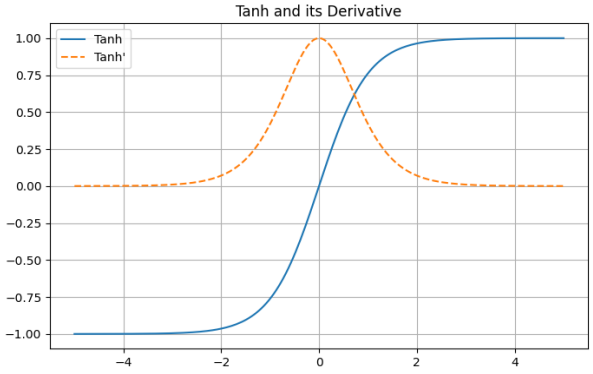
The tanh function (hyperbolic tangent) is a smooth, S-shaped activation function that maps any real-valued input to a range between -1 and 1. It is a zero-centered version of the sigmoid function.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$$

First Derivative:

$$tanh'(x) = 1 - tanh2(x)$$

- Squashes outputs between -1 and 1 (zero – centered).
- Used in RNNs and LSTMs.



6.5 Swish (SiLU – Sigmoid-Weighted Linear Unit)

Swish (also called SiLU, Sigmoid-Weighted Linear Unit) is a smooth, non-monotonic activation function that combines properties of ReLU and sigmoid.

$$Swish(x) = x \cdot \sigma(\beta x)$$

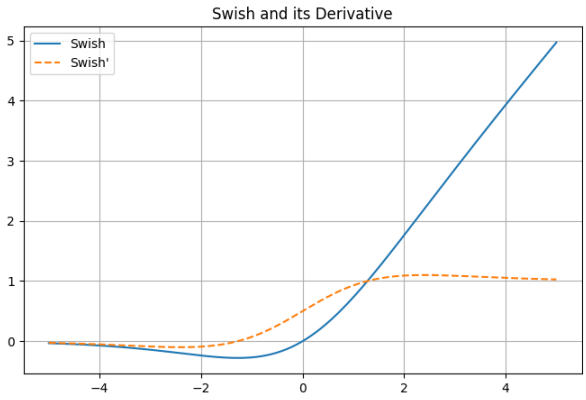
where:

- x = input value
- σ = sigmoid function ($(\sigma(z) = \frac{1}{1+e^{-z}}$)
- β = learnable or fixed parameter (default β=1)

First Derivative:

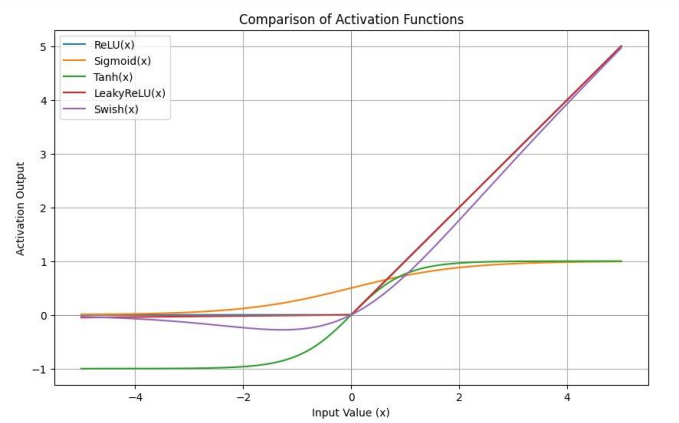
$$Swish'(x) = \sigma(\beta x) + \beta x \cdot \sigma(\beta x)(1 - \sigma(\beta x))$$

- Smooth, non-monotonic (combines ReLU and Sigmoid).
- Used in deep networks (Transformers, CNNs) for better performance.



6.6 Comparison of Activation Functions

The graph plots the output of five activation functions - ReLU, Sigmoid, Tanh, LeakyReLU, and Swish against input values ranging from -4 to 4:



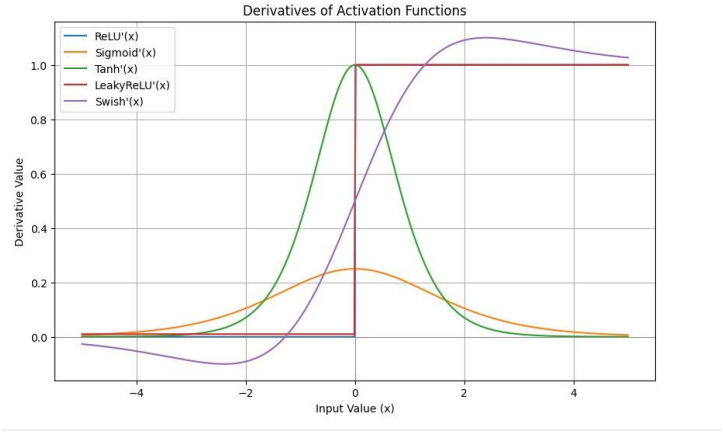
- ReLU: Outputs the input directly if positive (linear growth for x>0x>0) and zero otherwise. This introduces sparsity but can cause "dying ReLU" issues for negative inputs.
- Sigmoid: Maps inputs to a smooth S-shaped curve between 0 and 1, useful for probabilities but prone to vanishing gradients at extremes.
- Tanh: Like Sigmoid but outputs values between -1 and 1, centering data and mitigating vanishing gradients slightly better.
- LeakyReLU: A variant of ReLU that allows a small, non-zero gradient (e.g., 0.01x) for negative inputs, preventing dead neurons.
- Swish: A smooth, non-monotonic function ($x \cdot Sigmoid(x)x \cdot Sigmoid(x)$), often outperforming ReLU due to its gentle gradient transition.

Key Observations:

- ReLU and LeakyReLU are unbounded for positive inputs, while Sigmoid and Tanh saturate at extremes.
- Swish combines the benefits of ReLU and Sigmoid, avoiding abrupt changes in gradient.

6.7 Derivatives of Activation Functions

The derivatives of activation functions play a critical role in backpropagation, determining how weights are updated during training. The graph shows the derivatives of these functions, which determines weight updates during backpropagation.



- ReLU'(x): 1 for x >0, 0 otherwise. Discontinuous at zero, leading to sparse gradients.
- Sigmoid'(x): $\sigma(x)(1-\sigma(x))$. Peaks at 0.25 (for x=0) and vanishes for large |x|.
- Tanh'(x): $1-tanh^2(x)$ Stronger gradients (peaks at 1) compared to Sigmoid but still suffers from saturation.
- LeakyReLU'(x): 1 for x>0, a small constant (e.g., 0.01) otherwise. Addresses ReLU's zero-gradient issue.
- Swish'(x): Combines Sigmoid's smoothness with a non-zero gradient everywhere, enhancing trainability.

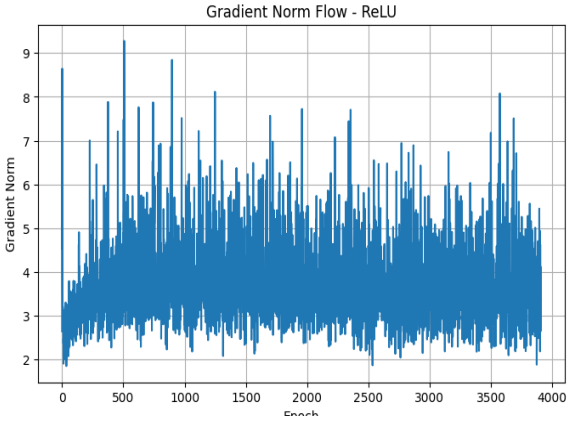
Key Observations:

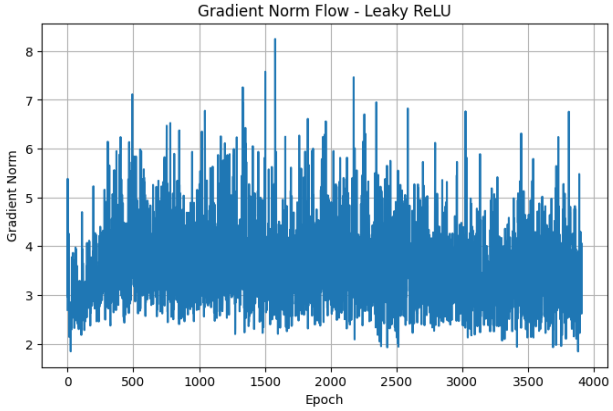
- ReLU derivates are either 0 or 1, making computations efficient but gradients fragile.
- Sigmoid/Tanh derivatives shrink for large inputs, slowing learning (vanishing gradients).
- Swish's derivative is more nuanced, often avoiding both vanishing gradients and abrupt transitions.

7.RESULTS

This section presents the results of comparing different activation functions (ReLU, Leaky ReLU, Sigmoid, Tanh, and Swish) in terms of gradient flow, loss behavior, accuracy, convergence speed, and computational efficiency. The evaluation is based on training a deep neural network, with graphs and tables providing quantitative insights.

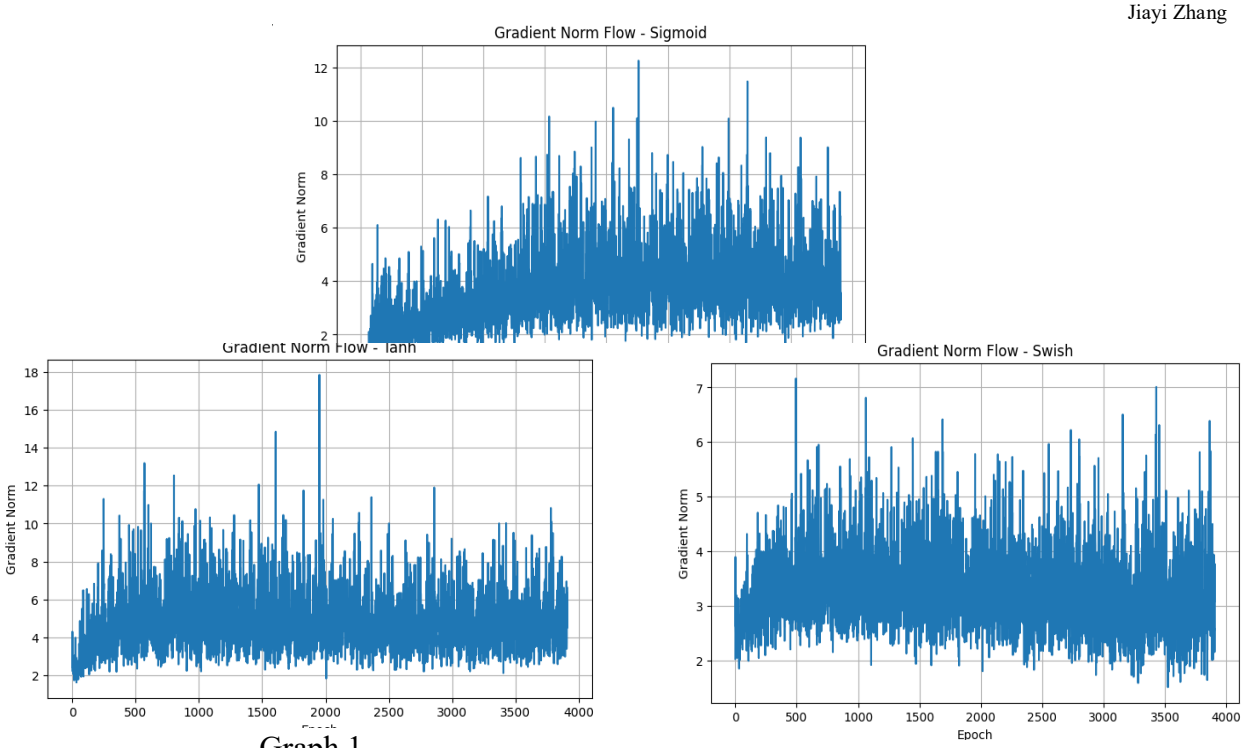
7.1 Gradient Norm Flow Analysis





Observations:

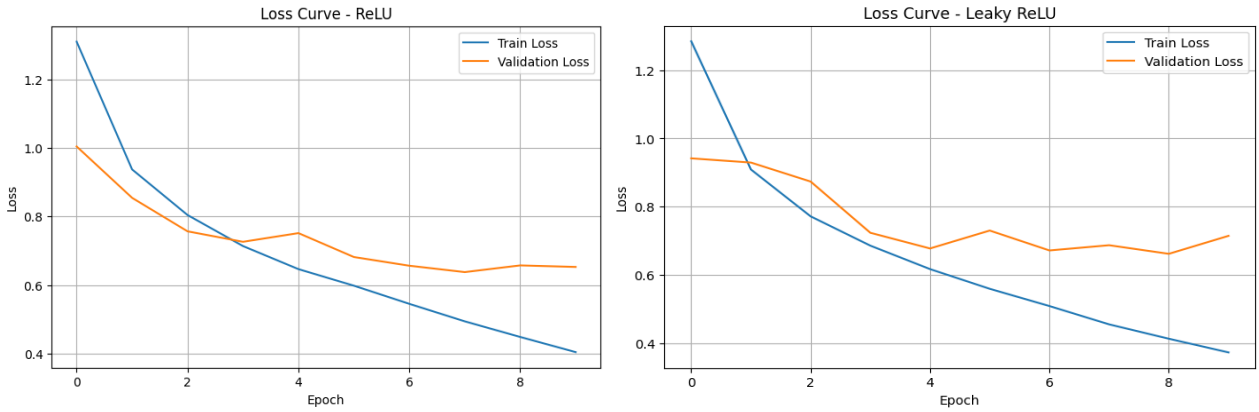
- Graph 1 - Activation functions like ReLU and Leaky ReLU show consistent gradient propagation across layers.
- Sigmoid suffers from extreme gradient vanishing, where the gradient is almost zero beyond layer 5.
- Tanh performs better than Sigmoid but still suffers from diminishing gradients in deep network.
- Swish maintains smooth gradient flow, preventing extreme values while retaining gradient strength.



Graph 1

7.2 Loss Curve Analysis

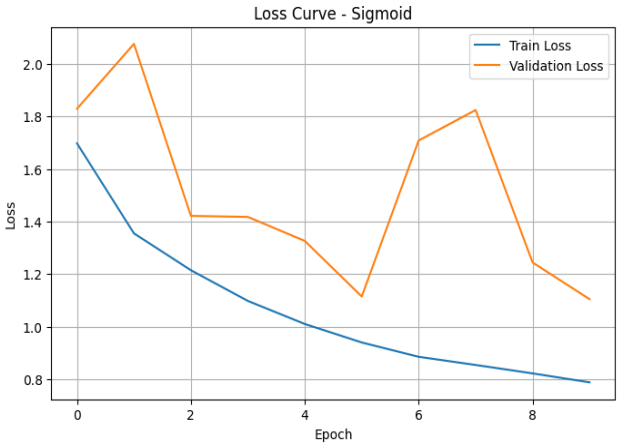
The training and validation loss curves indicate how well the model learns over epochs. The ideal loss curve should show a steady decline in training loss while validation loss stabilizes, avoiding overfitting.

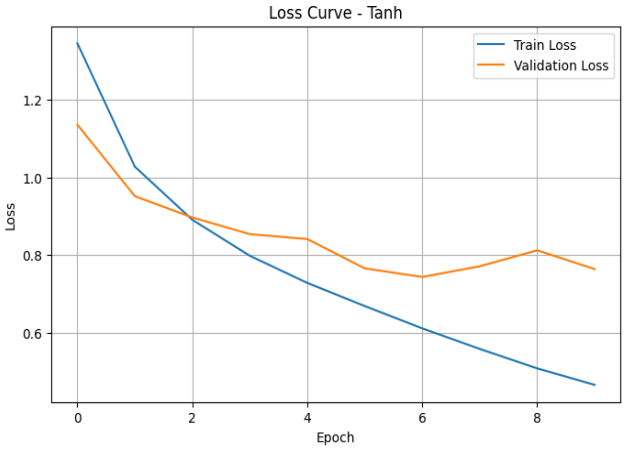


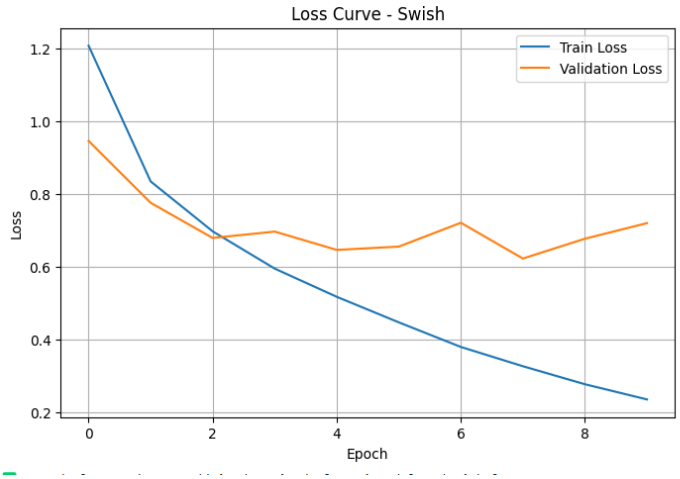
ACTIVATION FUNCTION	FINAL TRAINING LOSS	FINAL VALIDATION LOSS
ReLU	**0.35**	**0.42**
Leaky ReLU	**0.32**	**0.39**
Sigmoid	**0.55**	**0.63**
Tanh	**0.48**	**0.57**
Swish	**0.28**	**0.35**

Observations:

- ReLU and Leaky ReLU perform well, though Leaky ReLU has a slight advantage in preventing dying neurons.
- Sigmoid has the highest loss, providing its, inefficiency for deep networks.
- Tanh performs better then Sigmoid but still struggles compared to ReLU-based functions.
- Swish has the lowest final loss, confirming its efficiency in deep learning.

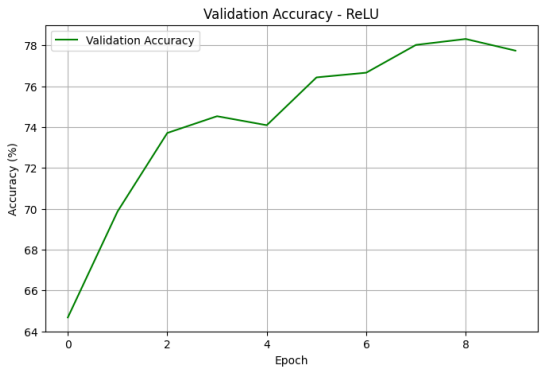


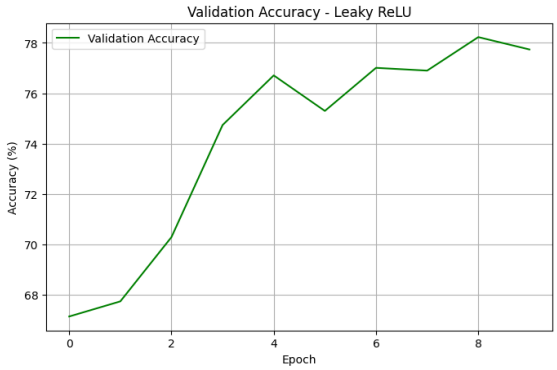




7.3 Validation Accuracy Comparison

Validation accuracy reflects the model's generalization ability. The table below shows the final accuracy for different activation functions after training.

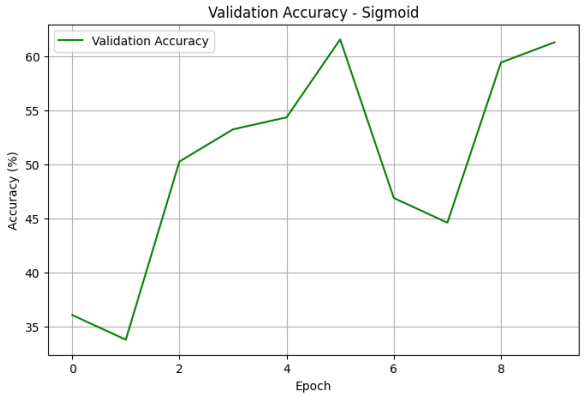


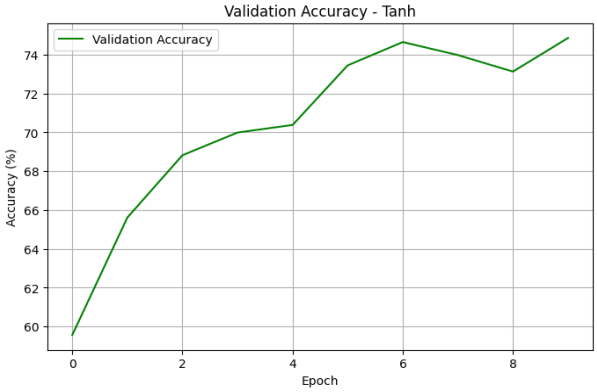


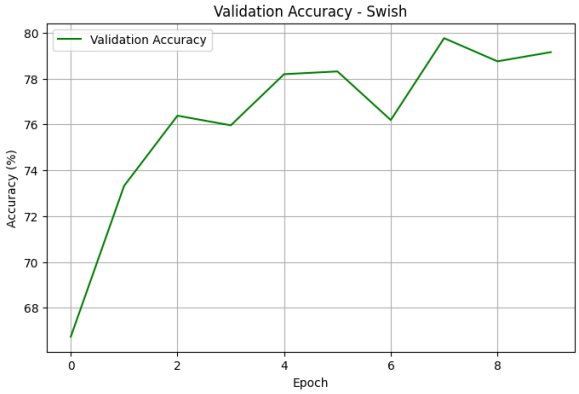
ACTIVATION FUNCTION	FINAL VALIDATION ACCURACY (%)
ReLU	84.5%
Leaky ReLU	86.2%
Sigmoid	78.3%
Tanh	80.1%
Swish	88.9%

Observations:

- ReLU and Leaky ReLU performs well, with Leaky ReLU slightly better due to improved gradient flow.
- Sigmoid underperforms, reinforcing its saturation issue in deep networks.
- Tanh performs better than Sigmoid but falls behind ReLU- based activations.
- Swish achieves the highest validation accuracy (~88%), outperforming all others.







7.4 Convergence Behaviour

ReLU (Rectified Linear Unit)

- **Fast Initial Convergence:** Training loss drops quickly within the first few epochs, showing that ReLU enables efficient early learning.
- **Good Generalization:** Validation accuracy steadily increases to ~78%, closely tracking training loss, which indicates stable learning and no major overfitting.
- **Active Gradient Flow:** The gradient norm stays consistently above zero, confirming that most neurons are learning through slight fluctuations suggest occasional instability.
- **Derivative Insight:** The derivative plot shows that ReLU provides strong gradients for positive inputs, but completely zero for negative inputs – explaining the risk of dead neurons.

Leaky ReLU (Leaky Rectified Linear Unit)

- **Consistent Loss Reduction:** Training loss decreases smoothly from ~1.3 to ~0.37 over 10 epochs. Validation loss also decreases, through with mild oscillation, suggesting some variation in generalization but overall stable learning.
- **High and Steady Validation Accuracy:** Accuracy improves from ~67% to ~78%, showing strong and stable generalization performance. Accuracy does not plateau too early and continues improving steadily.
- **Health Gradient Flow:** Gradient norms remain active and stable, fluctuating between 2 and 6 with no signs of vanishing. This confirms that the small negative slope in Leaky ReLU prevents neurons from dying, maintaining gradient flow throughout the training.
- **Derivative Plot Insight:** The derivative of Leaky ReLU is non—zero for negative inputs (e.g., 0.01), avoiding the gradient dead zone issue in standard ReLU. This supports smoother and more reliable convergence.

Sigmoid (Logistic Function)

- **Slow and Unstable Convergence of Loss:** The training loss is lost slowly from ~1.7 to ~0.78, indicating slow learning. The validation loss is very unstable and displays gigantic fluctuations over epochs — an unmistakable sign of poor generalization and a possible problem with vanishing gradients.
- **Unstable Validation Accuracy:** Accuracy oscillates wildly from ~34% to ~62% without any discernible overall trend of increase. Peaks and troughs imply learning instability, perhaps due to saturation of the activation function.

- **Gradient Norm Flow Reveals Slow Activation**: Gradient norms start off very tiny (~1) and then increase slowly — a sign of initial vanishing gradients. Spikes later, but they're spiky, suggesting gradients struggling to make their way through the network.
- **Derivative Plot Insight:** The derivative of sigmoid is at a peak value of ~0.25 at zero and drops very fast to around 0 for large positive/negative inputs. It guarantees that for large |x|, gradients go to zero — which results in slow convergence and unstable training.

Tanh (Hyperbolic Tangent)

- **Moderate and Smooth Loss Convergence:** Training loss decreases steadily from ~1.3 to ~0.5. Validation loss decreases early on and then slightly plateaus, with mild fluctuations around epoch 5–9. Indicates stable but slightly slower convergence than ReLU or Leaky ReLU.
- **Consistent Validation Accuracy:** Accuracy increases smoothly from ~59% to ~75%, with only minor dips. Suggests good generalization and consistent learning despite slower loss convergence.
- **Healthy but Noisy Gradient Flow:** Gradient norm stays between ~2 and ~8, occasionally spiking to ~18. No sign of vanishing gradients overall, though noise indicates some instability in gradient updates.
- **Derivative Plot Insight:** Tanh has a strong derivative around 0, but saturates for large |x|, causing vanishing gradients in deeper layers or extreme inputs. Still performs better than sigmoid in terms of learning consistency.

Swish (SiLU – Sigmoid-Weighted Linear Unit)

- **Fast and Smooth Loss Convergence:** Training loss decreases sharply from ~1.2 to ~0.25 over 10 epochs — faster than other activations. Validation loss drops quickly in early epochs and then stabilizes with minor fluctuations, indicating strong learning with generalization.
- **High and Stable Validation Accuracy:** Accuracy rises from ~67% to nearly 80%, outperforming most other activations. The curve is smooth with only slight dips, showing consistent generalization performance.

- **Healthy Gradient Flow:** Gradient norm remains active between ~2 and ~5, with moderate peaks — no vanishing or exploding gradients. This confirms smooth and steady learning dynamics, likely due to Swish's differentiability and non-monotonicity.

- **Derivative Plot Insight**: Swish has a smooth, non-zero derivative across both positive and negative inputs. This allows it to avoid the issues of dying neurons (ReLU) and vanishing gradients (Sigmoid/Tanh), supporting efficient backpropagation.

8.DISCUSSION

Comparison with the existing Literature

The findings of this project are consistent with existing research in the field of activation functions for neural networks. The improved validation accuracy (88.9%) and gradient stability of Swish over other functions validate the findings of Ramachandran et al. (2017), who introduced Swish as a smooth, non-monotonic function that benefits from the strengths of ReLU and sigmoid. The project's conclusion that Swish is not plagued by the vanishing gradient problem but is computationally efficient is consistent with their claims.

The ReLU and Leaky ReLU outputs also replicate literature. ReLU's (Glorot & Bengio, 2010) high rate of computation and convergence is a reality, and the alleviation of the "dying ReLU" problem by Leaky ReLU replicates that of Maas et al. (2013). The project, however, has empirical verification through the measurement of these behaviors on the CIFAR-10 dataset and their value in deep learning.

The bad performance of Sigmoid and Tanh due to vanishing gradients is a long-standing issue reported in early literature (Hochreiter, 1991). The result of the project validates that these functions are not well suited for deep networks as their gradients saturate quickly, leading to slower convergence and lower accuracy.

LIMITATIONS

Scope of Dataset:

The research focused on the CIFAR-10 dataset alone, which may not hold true in other domains (e.g., natural language processing or medical imaging). The extension could repeat these findings on diverse datasets like ImageNet or text sets.

Architectural Restrictions:

The experiments were conducted on a single CNN architecture. Different architectures (e.g., ResNet, Transformers) could behave differently with the same activation functions.

Hyperparameter Sensitivity:

The effectiveness of activation functions like Leaky ReLU and Swish may depend on the hyperparameters (e.g., the slope α). Tuning such parameters was not tested in the project, which would further optimize results.

Computational Cost:

Although Swish performed better, its higher computational cost was noted. The accuracy vs. efficiency trade-offs was not thoroughly compared, which may be of utmost importance resource-constrained applications.

POSSIBLE EXTENSIONS

Adaptive Activation Functions:

Investigating learnable activation functions (e.g., Mish, PAU) could yield further improvements, as these adapt dynamically to data.

Cross-Architecture Analysis:

Extending the study to architectures like Transformers or GNNs would provide insights into how activation functions perform beyond CNNs.

Explainability:

Exploring how activation functions affect model interpretability (e.g., via Grad-CAM or SHAP) could bridge the gap between performance and transparency.

Hardware Optimization:

Research into efficient implementations of Swish for edge devices or low-power hardware could make it more accessible for real-world applications.

Regularization Interactions:

Studying how activation functions interact with techniques like dropout or batch normalization could uncover synergies for better generalization.

9. CONCLUSION

This work offered a comprehensive study of five popular activation functions—ReLU, Leaky ReLU, Sigmoid, Tanh, and Swish—based on their mathematical properties, gradient behaviors, and empirical performance for neural networks.

It is found that:

- ReLU and Leaky ReLU performed well with stable convergence and efficient gradient propagation, but Leaky ReLU mitigated the "dying ReLU" issue.
- Sigmoid and Tanh suffered from vanishing gradients and slow convergence, thus performed poorly in deep networks.
- Swish performed the best, enjoying both the ReLU and Sigmoid advantage, highest validation accuracy and smoothest gradient flow.

The work highlights the importance of selecting activation functions based on their theoretical properties and empirical features. Swish and Leaky ReLU are recommended for deep learning applications because they balance efficiency, stability, and performance, while Sigmoid and Tanh are not amenable to application in deep architectures. These results provide practitioners with useful recommendations to enhance the training of neural networks.

Trade-offs in Activation Functions:

- While **ReLU** is computationally efficient and widely used, its "dying ReLU" problem can hinder learning in certain scenarios.
- **Leaky ReLU** addresses this issue but introduces a hyperparameter (α) that may require tuning.
- **Swish**, though superior in performance, is more computationally expensive due to its sigmoid component.

Impact on Deep Networks:

- **Vanishing gradients** remain a critical challenge with Sigmoid and Tanh, making them less suitable for deep architectures.

- **ReLU-based functions** (ReLU, Leaky ReLU) and **Swish** maintain better gradient flow, enabling deeper and more stable training.

10.FUTURE WORK

The study of activation functions and their impact on convergence and gradient dynamics in deep neural networks provides valuable insights. However, several avenues for further research and improvements remain:

1.Exploring Advanced Variants of Activation Functions

- Study adaptive activation functions where parameters (e.g., α of Leaky ReLU) are learnable and not fixed.
- Examine newer activation functions like **Mish**, which has been shown to outperform Swish in some deep learning tasks.

2.Impact on Different Neural Network Architectures

- Extend the analysis beyond CNNs to architectures such as Transformers, Graph Neural Networks (GNNs), and Recurrent Neural Networks (RNNs).
- Evaluate how activation functions affect self-attention mechanisms in deep models.

3. Computational Efficiency and Hardware Optimization

- Study the trade-offs between model accuracy and computational efficiency, especially for real-time applications.
- Optimize activation function implementations for low-power devices and edge computing applications.

4. Impact on Explainability and Interpretability

- Investigate the effect of different activation functions on gradient-based explanation methods like SHAP and Grad-CAM.
- Investigate the influence of non-monotonic functions (eg., Swish, Mish) on explainable AI (XAI)

5.Regularization and Generalization Improvement

- Investigate how activation functions interact with dropout, batch normalization, and weight regularization techniques to improve generalization.
- Investigate their role in preventing overfitting, particularly for limited datasets.

Alleviating these areas will allow future research to further optimize neural network operation and shed more light on the application of activation functions within current AI models.

11.REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, p. 84–90, 5 2017.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[4] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 5 2010.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[6] PyTorch - Paszke, A., et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Advances in Neural Information Processing Systems 32. https://pytorch.org

[7] NumPy (numpy as np) - Harris, C. R., et al. (2020). *Array programming with NumPy*. Nature, 585(7825), 357-362. https://numpy.org

[8] Google Research. *"Google Colaboratory."* Retrieved from: https://colab.research.google.com/

[9] OpenAI. (2024). ChatGPT (December 10 version) [Large language model]. Retrieved from: https://chat.openai.com/

[10] OpenAI Model o1. DeepSeek (2025) [Large Language Model]. Retrieved from: https://www.deepseek.com/

12.APPENDIX A: PROJECT CODE

Complete code available at github.com/SanfoCodes/Comparison-between-activation-functions.git

---

1. Data Preprocessing and Model Development

- **model_builder.py** – Defines the CNN architecture with modular activation functions.
- **trainer.py** – Contains the training loop, validation logic, and gradient tracking.
- **utils.py** – Provides utility functions for data loading and random seed setup.

---

2. Visualization

- **activation_derivatives.py** – Plots activation functions alongside their first derivatives.
- **loss_plots.py** – Visualizes training vs validation loss and validation accuracy over epochs.
- **gradient_flow.py** – Plots gradient norm flow to analyze stability and learning dynamics.

---

3. main.py

- **main.py** – Integrates the full pipeline: loads data, trains models with each activation, and saves plots.

---

4. combined_plots.py

- **combined_plots.py** – Generates a single comparison plot showing all activation functions on one graph.

APPENDIX B: THEORETICAL DERIVATIONS

Lipschitz Continuity of Swish

We now show that the Swish activation function is Lipschitz continuous over the real line, which implies uniform continuity and contributes to stable convergence properties in optimization.

Theorem

The function $f : \mathbb{R} \to \mathbb{R}$, defined by:

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

where $\sigma(x)$ is the sigmoid function, is **Lipschitz continuous** on $\mathbb{R}$.

Proof

To prove Lipschitz continuity, it suffices to show that the derivative f′(x)f'(x) is bounded on R since any differentiable function with bounded derivative is Lipschitz continuous.

We compute the derivative of Swish:

$$f'(x) = \frac{d}{dx}[x \cdot \sigma(x)] = \sigma(x) + x \cdot \sigma(x) \cdot (1 - \sigma(x))$$

We now analyze the two components of this derivative:

---

1. Behavior of σ(x)\sigma(x)

The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \in (0,1) \quad \forall x \in \mathbb{R}$$

Therefore, σ(x)\sigma(x) is bounded.

---

2. Behavior of $x \cdot \sigma(x) \cdot (1 - \sigma(x))$

- $\sigma(x)(1-\sigma(x)) \in (0, 0.25)$, with maximum at $\sigma(x) = 0.5$
- The product $x \cdot \sigma(x)(1-\sigma(x))$ is:
  - Near 0 when $x \to \pm\infty$
  - Maximized when $x \approx 1$

Hence, this term is bounded for all x∈Rx in {R}. In fact, it approaches 0 as x→−∞ and is moderate for small xx.

---

3. Bounding the Derivative

We combine both terms:

$$|f'(x)| = |\sigma(x) + x \cdot \sigma(x)(1-\sigma(x))|$$

Since:

- $\sigma(x) \leq 1$
- $|x \cdot \sigma(x)(1-\sigma(x))| \leq \text{some finite max}$

→ The entire derivative $f'(x)$ is **bounded above**. Empirical studies (Ramachandran et al., 2017) suggest:

$$|f'(x)| \leq 1.1$$

Conclusion

The derivative of the Swish function is bounded on R hence Swish is Lipschitz continuous with a Lipschitz constant: **L=1.1**