Activation Functions and Weight Initializers for CNN: A comparative study

Quy–Hung Xin[1]*, Duc-Tuan Doan[1]
[1]Faculty of Information Technology, University of Science-VNUHCM, Vietnam
*Corresponding author: xqhung23@apcs.fitus.edu.vn

***Abstract*—The weight initialization and activation function are critical factors that influence the performance of deep neural networks during training. A poor selection of activation functions and weight initialization approaches may result in the loss of information during forward propagation and cause gradients to vanish or explode during backpropagation. In this study, we present an empirical study of various combinations of activation functions and weight initialization methods, offering insights into their impact on metrics such as loss convergence, accuracy, and training time. We divide our experiments into two parts: training with and without batch normalization. When the model is trained without batch normalization, our results show that pairings of modern and less common activation functions and weight initializers, such as ELU with LSUV and Tanh with Orthogonal, outperform the traditional pairings like He and ReLU in terms of model performance. In contrast, the model becomes less dependent on the choice of activation functions and weight initializers using batch normalization. Additionally, the optimal pairings differ from those in a model without batch normalization and traditional study. Our findings provide practical insights into the choice of activations and initializers to maximize performance when training different CNN architectures.**

***Index Terms*—Activation functions, initializers, convolutional neural networks, deep neural networks.**

I. Introduction

In recent years, optimizing neural networks, particularly convolutional neural networks (CNNs), has been a major focus in the deep learning field. The primary objectives include enhancing learning capacity, improving accuracy for real-world applications, and ensuring model stability during training while maintaining computational efficiency. One strategy to achieve these objectives is improving key components of CNN models, particularly activation functions (AFs) and weight initializers (WIs).

AFs introduce non-linearity into models, enabling them to learn and represent complex patterns in data. Other architectural or regularization techniques for CNNs, such as Batch Normalization [10], Skip connections [6], and Dropout [21], while they can improve model performance, often add more additional complexity to the model. In contrast, better AFs can act as a lightweight improvement to the model. In addition, they play a direct role in shaping the training process and significantly influence the learning dynamics, particularly in deep architectures, where models are more prone to vanishing and exploding gradient problems [4]. In fact, the depth of CNN architectures has increased significantly in the past decade (One of the most popular CNN models, ResNet, [6] has up to 152 hidden layers in its largest variant while DenseNet [9] extends to as many as 201 hidden layers), making it important to develop AFs that maintain stable gradient flow, accelerate convergence, and enhance model efficiency.

WIs, on the other hand, determine the initial values of the weights in a neural network before training begins. A poor choice of WI can lead to common issues such as vanishing or exploding gradients, slow convergence, and unstable optimization. Moreover, the combination of AFs and WIs should be carefully considered, as it significantly impacts model performance. For instance, ReLU typically performs better with He initialization than with Xavier initialization [5]. Together, AFs and WIs play a pivotal role in determining the training efficiency, stability, and overall effectiveness of CNNs.

This study aims to conduct a comparative analysis of AF and WI pairings by evaluating their practical performance. For activations, we compare popular AFs, including Tanh, ReLU, ELU [1], GELU [7], SiLU, and Mish [15]. For initializers, we examine widely used WI methods, such as Xavier and He, as well as those with promising performance, including Orthogonal [20] and Layer-Sequential Unit-Variance (LSUV) [14].

To empirically assess the effectiveness of various AF-WI combinations, we conduct experiments using modified VGG19 models on the CIFAR-10 dataset. These experiments evaluate the behavior of each combination in networks with and without using any normalization techniques. Our assessment criteria include both accuracy and convergence speed.

Our contributions are two-fold:

- We provide a comprehensive analysis of state-of-the-art AFs and WIs, offering a deeper understanding of their underlying mathematical foundations. This analysis helps bridge the gap between observed performance and theoretical principles, ultimately contributing to the development of more reliable and efficient models.
- Through our experiments, we systematically evaluate the empirical effectiveness of different AF-WI pairings with different model architectures. Our findings offer valuable insights into the trade-offs associated with different initialization techniques, as well as how the interaction between AFs and WIs impacts training dynamics, convergence speed, and overall model performance across various network architectures and datasets.

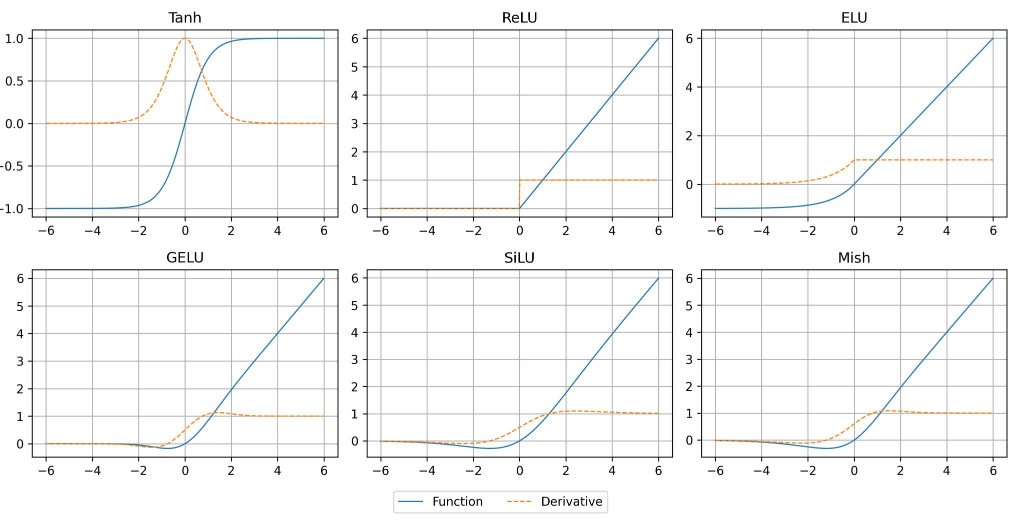

Fig. 1. Graph of Activation Functions and their Derivatives

II. Related work

Previous work mainly focused on enhancing AFs. To address the vanishing and exploding gradient problems associated with early AFs (e.g., Sigmoid and Tanh), Nair and Hinton introduced the ReLU activation function, which has since become one of the most widely used AFs [16]. To further enhance network performance, numerous ReLU variants have been proposed. For example, Leaky ReLU [13] introduces a small negative slope to mitigate the dying ReLU problem, while PReLU [5] and ELU [1] generalize traditional ReLU by incorporating learnable parameters, improving a model's ability to learn complex representations. More recently, advanced AFs, such as Swish [18], Mish [15], and GELU [7], have been introduced, showcasing improved gradient flow and generalization in deep networks.

Meanwhile, the impact of WIs was also explored with He et al. [5] introducing He initialization, which was shown to outperform the earlier Xavier initializer proposed by Glorot and Bengio [3] on networks using the ReLU function. Meanwhile, Xiao et al. also reintroduced Orthogonal initialization when discussing its significance in maintaining *dynamic isometry* (a property where, during training, the singular values of the Jacobian matrix of the network remain close to 1, ensuring stable gradients and effective learning). In an effort to stabilize training, Mishkin and Matas introduced LSUV initialization [14], examining its interaction with different AFs. However, their study was limited to traditional AFs, such as Sigmoid, Tanh, ReLU, and PReLU, leaving the impact of modern AFs largely unexplored.

While previous research has extensively examined AFs and WIs, the interaction between them remains an area of ongoing exploration. Tomasz Szanda [22] investigated the performance of multiple AFs, but did not analyze their compatibility with different weight initialization strategies. More recently, Kit Wong, Rolf Dornberger, and Thomas Hanne [23] studied various AF-WI combinations in a simple feedforward neural network (FNN) [19]. However, the simplicity of their chosen datasets and models may not fully capture the potential of AF-WI interactions in real-world applications.

III. Activation Functions

AFs are an essential component of neural networks, particularly during the training process. Without AFs, a neural network would reduce to a linear combination of inputs, making it insufficient to learn complex patterns and hierarchical representations.

An efficient activation function should exhibit several key properties, including non-linearity, differentiability, computational efficiency, and smoothness for stable gradient flow [17]. Though there are many ways to classify AFs, we are interested in the two characteristics of AFs: saturating and non-saturating.

- Saturating functions approach a fixed limit as the input becomes large (positive or negative), leading to vanishing gradients that hinder training. Examples include Sigmoid and Tanh.
- Non-saturating functions, on the other hand, do not suffer from this issue and are generally preferred in deep learning due to their better gradient flow and faster convergence.

In this section, we will analyze all the AFs used in the experiments

A. Tanh

The hyperbolic tangent (tanh) function is defined as:

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

Tanh was one of the earliest AFs introduced in neural networks during the 1980s, alongside the Sigmoid function. However, it is a saturating function, which can make training more difficult compared to non-saturating AFs such as ReLU [5]

Recently, Tanh has regained attention for training CNNs after Xiao et al. [24] proved that Tanh is a key condition for achieving dynamical isometry in deep networks when combined with orthogonal initialization. We choose to investigate Tanh not only as a representative of saturating functions but also to explore its potential to maintain dynamical isometry.

B. ReLU

The Rectified Linear Unit (ReLU) is one of the most widely used activation functions in deep neural networks. It introduces non-linearity to the network while maintaining computational efficiency. ReLU is defined as follows:

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0. \end{cases}$$

However, ReLU has a major drawback known as the dying ReLU problem, where neurons become inactive and output only zero. This issue occurs when large negative inputs result in a zero gradient, preventing affected neurons from learning further.

C. ELU

The Exponential Linear Unit, also known as ELU, was proposed by Clevert et al. [1]. It generalizes the idea of extending the ReLU function of its predecessors (i.e. LeakyReLU and PReLU). The ELU is defined as follows:

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

where α is the learnable parameter. And the derivative of this function is given as:

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

Unlike ReLU, which outputs zero for negative inputs, ELU allows small negative values, helping reduce shifts and improve learning rate.

D. GELU

The Gaussian Error Linear Unit (GELU) was introduced by Hendrycks & Gimpel in 2016 [7]. GELU applies a Gaussian-based gating mechanism to decide how much input should pass through. It is defined as follows:

$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

where $\Phi(x)$ is the standard Gaussian cumulative distribution function:

$$\Phi(x) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

E. SiLU

The SiLU (or also known as Swish) function was introduced by Ramachandran et al. (2017) [18] as a self-gated activation function. It is defined as follows:

$$\mathrm{SiLU}(x) = x \cdot \mathrm{Sigmoid}(x) = \frac{x}{e^x - 1}$$

Like aforementioned functions, SiLU also allows small negative values, preventing the Dying ReLU problem. As a result, this function can outperform ReLU in deep networks by smoothing activation behavior, making optimization easier.

F. Mish

The Mish activation function was introduced by Misra (2019) [15] as an improvement of Swish and ReLU AFs. Its expression is defined as:

$$\mathrm{Mish}(x) = x \cdot \mathrm{Tanh}\,\mathrm{Softplus}(x)$$

where

$$\mathrm{Softplus}(x) = \ln(1 + e^x)$$

Mish provides smooth gradients, leading to better feature representation in deep networks. In practice, Mish achieves better accuracy compared to ReLU and Swish in image classification tasks [15]. However, the increased complexity of its formula results in higher computational cost, making it less efficient than ReLU for real-time applications or resource-constrained environments.

IV. Weight Initializers

WIs also play a crucial role in neural networks by determining the initial values of parameters before training. The choice of WI directly impacts gradient flow, convergence speed, and overall model stability. [2]

The naive approaches are to initialize all weights to a constant value (e.g., zero) and assign them random small values. However, these methods often result in poor model performance, leading to issues such as symmetry problems or unstable gradient propagation. Thus, more advanced initialization techniques should be used to ensure effective learning and stability.

This section will investigate all the WIs used in experiments

A. Xavier Initialization

Proposed by Xavier Glorot and Yoshua Bengio [3], Xavier initialization was one of the earliest weight initialization methods designed to improve the stability of deep neural networks. It remains widely used today due to its ability to maintain variance across different layers during both the forward and backward pass, reducing the risk of vanishing and exploding gradients.

There are two variants of Xavier initialization:

- Xavior Uniform Initialization:

$$W \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right)$$

- Xavior Normal Initialization:

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}} + n_{\text{out}}}\right)$$

where n_{in} and n_{out} represent the number of input and output neurons in the layer, respectively. By scaling the initial weights appropriately, Xavier initialization helps preserve stable gradient flow, enabling more efficient training of deep networks.

However, it should be noted that Xavier initialization was originally designed for AFs like Sigmoid and Tanh, which were widely used at the time of its proposal. These functions are approximately linear near zero and symmetric about the origin. Xavier initialization assumes that the variance of activations is preserved across layers under these conditions. However, for non-saturating AFs, such as ReLU, these assumptions no longer hold. For instance, ReLU produces only non-negative outputs, breaking the zero-mean assumption and leading to a mean shift in activations. As a result, Xavier initialization often leads to suboptimal weight scaling for ReLU-based networks, especially in deep networks [12].

B. He initialization

In 2015, He et al. introduced an alternative WI method that addressed the limitations of Glorot and Bengio's (Xavier) initialization [5]. Their method accounts for the non-linearity of AFs, making it one of the most widely used initialization techniques in deep learning.

Similarly, there are two variants of He initialization:

- He uniform initialization:

$$W \sim U\left(-\sqrt{\frac{6}{n_{\text{in}}}}, \sqrt{\frac{6}{n_{\text{in}}}}\right)$$

- He normal initialization:

$$W \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right)$$

where n_{in} represents the number of inputs.

Despite its effectiveness for modern non-saturating AFs, a model may still suffer from issues such as dying ReLU neurons and vanishing gradients in very deep networks. To address these challenges, several approaches have been proposed, including more sophisticated WI techniques.

C. Orthogonal initialization

Orthogonal initialization was reintroduced by Saxe et al. for recurrent neural networks (RNNs) [20]. However, it has also been shown to improve performance in deep networks by preserving the variance of activations across layers.

The initialization process consists of the following steps:

1) Generate a random matrix $A \in \mathbb{R}^{m \times n}$, where each element follows a normal distribution:

$$A_{ij} \sim \mathcal{N}(0, 1)$$

where m and n denote the input and output dimensions of the layer.

2) Compute the QR decomposition of A:

$$A = QR$$

where Q is an orthogonal matrix and R is an upper triangular matrix.

3) Adjust the sign of Q to maintain consistency:

$$W = Q \cdot \text{sign}(\text{diag}(R))$$

4) Apply a scaling factor γ accordingly to the AF to adjust the variance:

$$W' = \gamma W$$

Later, Xiao et al. [24] and Wei Hu et al. [8] proved that orthogonal initialization can help a network achieve dynamical isometry in deep neural networks, enabling effective training even with as many as 4000 layers. This result is comparable to the performance of widely-used non-saturating AFs when paired with He initialization.

D. LSUV initialization

Layer-Sequential Unit-Variance (LSUV) initialization was introduced by Mishkin & Matas (2015) [14]. It is a data-driven method that pre-trains the weights based on the data.

1) Initialize the weight matrix W using He initialization:
2) Perform a forward pass with a mini-batch of input data.
3) Compute the variance σ_l^2 of the activations for layer l:

$$\sigma_l^2 = \frac{1}{N} \sum_{i=1}^{N} (h_i^{(l)})^2$$

where N is the batch size and $h_i^{(l)}$ are the activations at layer l.

4) Adjust the weight matrix to normalize the variance:

$$W^{(l)} \leftarrow \frac{W^{(l)}}{\sqrt{\sigma_l^2}}$$

5) Repeat steps 2–4 for each layer until $|\sigma_l^2 - 1| < \epsilon$ (e.g., $\epsilon = 0.01$).

LSUV dynamically adjusts weights to maintain unit-variance activations before training begins, making it a powerful alternative to He initialization for deep networks. However, its extra computational cost means it is best suited for very deep architectures.

V. Experimental design

In this section, we will conduct experiments to evaluate the combinations of AFs and WIs. We are interested in the following questions:

RQ1. *Is there an optimal combination of AF and WI for CNNs?*

RQ2. *How do different pairings of AF and WI influence the CNNs' performance?*

A. Experimental setup

All experiments were conducted on a machine equipped with an Intel(R) Core(TM) i7 processor, 32 GB of RAM, and an NVIDIA RTX 5070 GPU. We employ Python and PyTorch as the primary deep learning framework for implementing and evaluating the models. To evaluate the models' performance, we will apply accuracy, loss, and convergence speed as main metrics.

In addition, for Xavier and He initialization methods, we adopt their respective normal (Gaussian) formulas. In the case of LSUV initialization, we set a maximum of 10 iterations for the orthonormal adjustment procedure.

B. Dataset and models

CIFAR-10 is a widely used benchmark dataset for image classification in deep learning and computer vision [11] [5]. It consists of 60,000 color images of size 32×32, categorized into 10 classes. As a medium-scale dataset, CIFAR-10 remains computationally efficient while being suitable for evaluating deep learning models.

TABLE I
VGG19-Based Configuration for CIFAR-10: With vs. Without Batch Normalization

VGG19 without BatchNorm	**VGG19 with BatchNorm**
Input (32 × 32 RGB Image)	
conv3-64, Act conv3-64, Act	conv3-64, BN, Act conv3-64, BN, Act
MaxPool 2×2	MaxPool 2×2
conv3-128, Act conv3-128, Act	conv3-128, BN, Act conv3-128, BN, Act
MaxPool 2×2	MaxPool 2×2
conv3-256, Act conv3-256, Act conv3-256, Act conv3-256, Act	conv3-256, BN, Act conv3-256, BN, Act conv3-256, BN, Act conv3-256, BN, Act
MaxPool 2×2	MaxPool 2×2
conv3-512, Act conv3-512, Act conv3-512, Act conv3-512, Act	conv3-512, BN, Act conv3-512, BN, Act conv3-512, BN, Act conv3-512, BN, Act
MaxPool 2×2	MaxPool 2×2
conv3-512, Act conv3-512, Act conv3-512, Act conv3-512, Act	conv3-512, BN, Act conv3-512, BN, Act conv3-512, BN, Act conv3-512, BN, Act
MaxPool 2×2	MaxPool 2×2
AvgPool 1×1	AvgPool 1×1
Linear $512 \to 10$	Linear $512 \to 10$

For the vanilla version, the network is modified as follows:

- The default AF used in the original architecture (ReLU) is replaced with the experimented AF.
- Since VGG19 was originally designed for input images of size 224×224, we adjust the layer configurations to accommodate the 32×32 input size of CIFAR-10.

For our experiments on CIFAR-10, we aim to compare the results of a vanilla network with those of a network that incorporates normalization techniques. We choose Batch Normalization as such technique since it is widely used in CNNs and it significantly impacts the dependency of the model on WIs and AFs [10]. We adopt VGG19 [6] as the base model and conduct experiments on two modified versions: one without any normalization and one with batch normalization.

The batch-normalized version of VGG19 follows the same architecture, with the addition of batch normalization layers after each convolutional block. Refer to Table I for architectural details.

Additionally, we apply basic data augmentation techniques, including *RandomCrop*, *RandomRotate* and *RandomErasing* which are built in Pytorch, to enhance generalization. We use Cross-Entropy Loss as the objective function and train the model with the SGD optimizer, setting the momentum to 0.9 and the weight decay to 5×10^{-4} since they are one of the most popular choices.

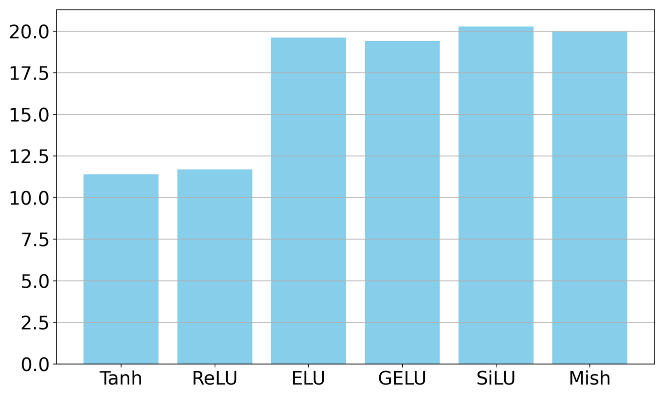

Fig. 2. Running time performance per activation (in minutes)

VI. Results

This section presents the results of the experiments and further discussion. The performance is evaluated based on training loss trends over training epochs, the final accuracy and training time.

A. Running time performance

Figure 2 illustrates the running time of each AF in a VGG19 model without further normalization techniques. As observed, Tanh and ReLU activations stand out for their superior computational efficiency, with each requiring approximately 11 minutes to complete the training process. Nevertheless, other AFs, such as ELU, GELU, SiLU and Mish, show significantly slower performance, with training time nearly double that of Tanh and ReLU, reaching around 20 minutes.

This phenomenon can be attributed to the fact that Tanh and ReLU are computationally lightweight, as they involve relatively simple mathematical operations, allowing for faster training times. In contrast, other AFs, such as GELU, require more complex calculations, like the Gaussian error function, and ELU, which has trainable parameters, which increases the computational load. As a result, these activation functions exhibit significantly slower performance compared to Tanh and ReLU.

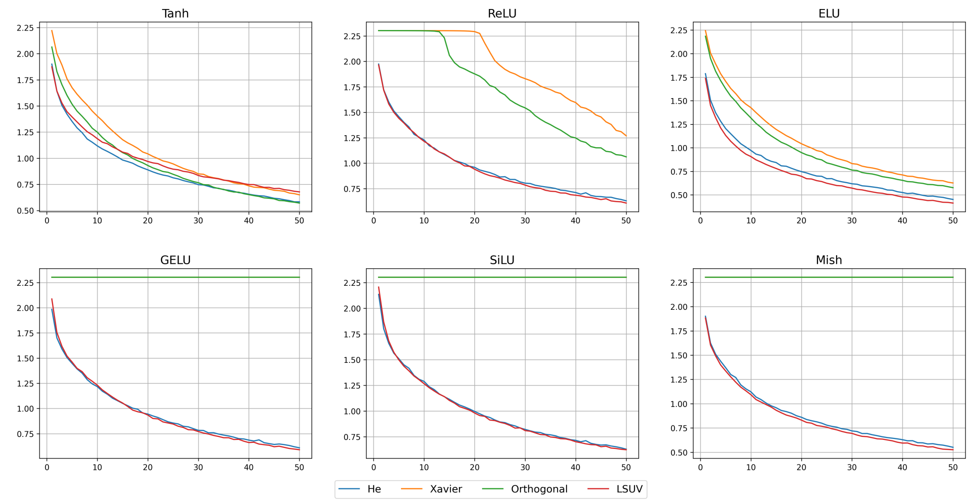

Fig. 3. Training loss of different AF-WI pairings after 50 epochs without batch normalization

B. *Without Batch normalization*

Figure 3 presents the training loss curves across 50 epochs for different AFs paired with various WI methods and Table II shows the final test accuracy after training.

TABLE II
FINAL TEST ACCURACY OF VARIOUS AF-WI PAIRINGS WITHOUT NORMALIZATION (IN PERCENTAGE)

Activation Functions	He	Xavier	Orthogonal	LSUV
Tanh	79.3	80.4	81.7	77.0
ReLU	79.7	56.8	65.7	79.7
ELU	82.2	80.8	81.9	83.2
GELU	79.8	10.0	10.0	79.5
SiLU	79.1	10.0	10.0	78.0
Mish	81.3	10.0	10.0	80.1

As expected, Xavier initialization demonstrates the weakest performance across all AFs, except for Tanh. Its poor performance becomes more significant when combined with modern AFs such as SiLU, Mish, and GeLU, where the training loss remains nearly constant throughout the training process. For ReLU, the loss stagnates during the initial 20 epochs. With ELU, the model learns properly for all epochs, but at a slower rate compared to other initializers.

As observed in Table II, Orthogonal initialization, when paired with Tanh, shows comparable results to He initialization when combined with modern AFs. However, when paired with other AFs, Orthogonal yields poor performance, outperforming only Xavier. For newer AFs, its behavior is similar to that of Xavier, with no learning progress observed during training.

He initialization proves to be a robust choice, as it consistently enables convergence across all AFs. Although LSUV requires a pre-training stabilization procedure, it shows little to no improvement over He initialization. Additionally, LSUV performs poorly when paired with Tanh.

While traditional pairings, such as ReLU with He initialization, remain effective, the best results are observed when ELU is paired with LSUV (0.832) and He (0.822). Tanh with Orthogonal also shows impressive results (0.817).

C. *With Batch normalization*

TABLE III
FINAL TEST ACCURACY OF VARIOUS AF-WI PAIRINGS WITH NORMALIZATION (IN PERCENTAGE)

Activation Functions	He	Xavier	Orthogonal	LSUV
Tanh	77.8	80.2	81.6	75.7
ReLU	80.6	82.5	85.0	80.9
ELU	82.6	84.8	87.6	82.8
GELU	81.0	84.9	85.6	81.2
SiLU	84.6	86.8	88.5	83.5
Mish	82.6	84.5	87.2	81.6

Figure 4 shows the training loss over 50 epochs, and Table III presents the final test accuracy for different AF and WI combinations with batch normalization. It is observed that, when batch normalization is applied, all pairings are able to converge, and those combinations that previously failed to learn now yield comparable results.

The results showed in indicate that Orthogonal initialization yields the highest accuracy across all AFs, with the best performance observed for SiLU (0.885), ELU (0.876), and Mish (0.872). Interestingly, Xavier, which is typically considered to perform worse than the He initializer, now outperforms it across all AFs. For LSUV, similar to the case without batch normalization, there is little improvement despite the pre-training process.

VII. DICUSSIONS

In this section, we discuss and interpret the results presented in the previous section while addressing the research questions of interest. We then show the limitations of our approach.

A. *Answers to research questions*

The research questions in this study were focused on the effect of AFs and WIs on the model. Through the experiments, the results have shown valuable insights to answer the questions.

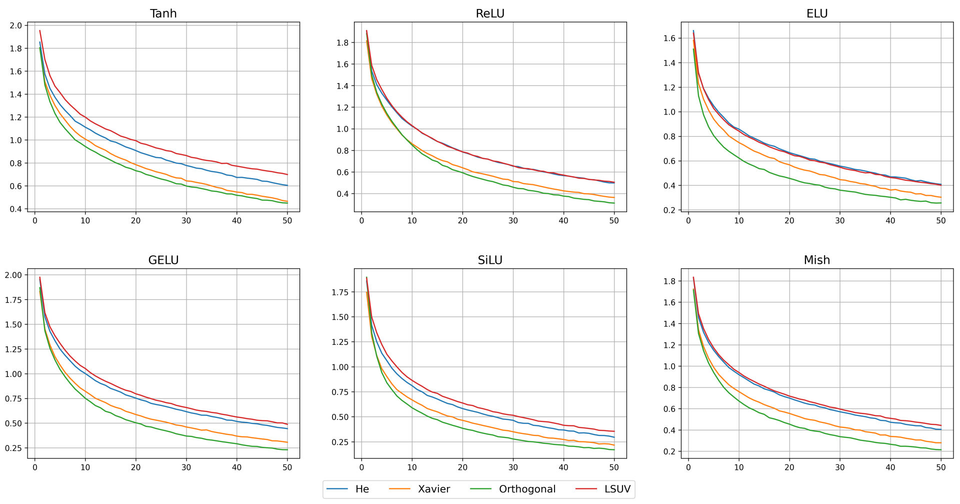

Fig. 4. Training loss of different AF-WI pairings after 50 epochs with batch normalization

RQ1. *Is there an optimal combination of AF and WI for CNNs?*

Research question RQ1 is addressed by conducting experiments on two different model architectures: one without batch normalization and one with batch normalization. The results indicate that there is no single optimal combination of AF and WI for CNNs, as the two CNN models showed different outcomes. For example, in the VGG19 model without batch normalization, ELU paired with LSUV yielded the best results, while some AFs, such as ELU, SiLU, and Mish, failed to learn when paired with Orthogonal initialization. However, in the case of the batch-normalized VGG19, Orthogonal initialization emerged as the best choice across all AFs. Therefore, the choice of AF and WI should depend on the specific model architecture and techniques applied.

RQ2. *How do different pairings of AF and WI influence the CNNs' performance?*

Research question RQ2 is addressed by analyzing the results across various AF and WI pairings. For a vanilla model, the combination of AF and WI is crucial because poor selections can prevent the model from learning effectively. Furthermore, some combinations lead to faster convergence than others. In the case of batch normalization, although all pairings result in convergence, the speed of convergence varies depending on the pairing. Hence, selecting the appropriate AF and WI can significantly impact the model's performance.

B. Models without Batch normalization

As observed in the previous section, the behavior of AFs and WIs differs significantly depending on whether batch normalization is used. In networks without batch normalization, the choice of AF and WI becomes especially critical, as performance varies more significantly. For non-saturating AFs, such as ReLU or Mish, Xavier initialization is generally unsuitable, as suggested by prior studies [5], [12].

Orthogonal initialization method when paired with Tanh activation has been reported to offer benefits for training deep neural networks [24], particularly by supporting dynamical isometry. However, when it is paired with different AFs, such as SiLU, GeLU and Mish, the performance was significantly worse. It can be because the properties of other AFs do not support the dynamic isometry condition, making the model unstable to train.

Our results suggest that the combination of AF and WI significantly impacts the performance of the model in the absence of batch normalization. A poor choice of AF-WI pairing can lead to issues such as slow convergence or failure to learn, as shown in our results. The results also indicate that newer AF-WI pairings can outperform traditional combinations. For instance, the pairing of Mish with LSUV yields the best performance in this setup. It shows the potential of training a vanilla model without relying on additional normalization techniques, thus reducing the overall complexity.

In addition, there is a clear trade-off between accuracy and training time. Modern activation functions, such as Mish and GeLU, often require more complex mathematical operations, leading to slower training times. In contrast, simpler activation functions, such as Tanh and ReLU, accelerate the training process, as demonstrated in Figure 2.

C. Models with Batch normalization

In contrast, in networks with batch normalization, the model learns effectively across all AF-WI combinations. This indicates that batch normalization reduces sensitivity to the specific choice of AF and WI. As a result, the selection of AF and WI becomes less critical, allowing other aspects of the model, such as adding more layers or incorporating additional techniques, to be prioritized for improvement.

In addition, the behaviors of the pairings are different from when there is no batch norm. For the case of orthogonal, it outperforms all WIs across all AFs. It can be the case that batch norm can also help maintain dynamic isometry when all the parameters are normalized across layers.

Furthermore, Xavier can surpass He even in models with non-saturating functions. Xavier initialization is designed to maintain a balance between the variance of activations and gradients, assuming both forward and backward signals should have similar magnitudes. When combined with batch norm, Xavier's more balanced scaling can work well because batch norm prevents activations from growing too large. On the other hand, He initialization scales weights to compensate for ReLU's zeroing-out of negative values, helping to maintain a stable variance of activations. However, with batch norm in place, activations are explicitly normalized, so the need for He's aggressive scaling is diminished. In some cases, it may even introduce unnecessary variance early in training.

D. Limitations

Due to resource and time constraints, our study employs a relatively simple model and dataset, which limits the exploration of potential WI and AF pairings in more complex models and with different techniques used in CNNs. More complex models with greater depth could reveal more pronounced differences in the performance of AF-WI pairings. Additionally, the exploration of various optimization techniques (beyond Batch Normalization) could lead to the discovery of other optimal pairings, providing further insights into the performance of different AF and WI combinations.

Despite these limitations, the study offers valuable theoretical and empirical insights into training CNN models, with findings that can be applied to deeper and more complex architectures.

VIII. Conclusion

In this study, we have investigated each AF and WI theoretically, ranging from traditional ones (e.g., Tanh, ReLU, He, and Xavier) to more modern ones (e.g., GeLU, Mish, and LSUV). We then conducted experiments testing the performance of AF and WI combinations on a modified VGG19 model, with and without batch normalization, using the CIFAR-10 dataset.

Our results have showed a noticeable difference between the two settings. When a model is trained without batch normalization, some combinations achieve remarkable results, such as ELU paired with He and LSUV, and Tanh paired with Orthogonal. However, the performance of certain pairs is suboptimal, particularly for AFs like GeLU, SiLU, and Mish. When paired with Xavier and Orthogonal initializers, these AFs lead to extremely poor performance, where the model fails to learn.

Using batch normalization, all the combinations can help the model learn at different rates. Additionally, the performance of the pairings becomes different from our traditional study. For example, when there is no batch norm, orthogonal initialization makes the model unable to learn. But when batch norm is applied, it outperforms all WIs across all AFs.

Our study suggests that the training performance of our approach, including accuracy and training time, is influenced by the choice of AF across various model architectures. An optimal selection of AF can lead to faster convergence, improving the model's ability to learn efficiently within fewer epochs. This not only enhances the model's accuracy but also reduces computational cost by minimizing the time spent in training. Furthermore, the impact of the AF choice is more pronounced in certain architectures, where specific AFs help the model navigate through complex patterns in the data more effectively. Consequently, selecting the right AF becomes a key factor in optimizing both the learning speed and the overall performance of the model, allowing for more effective deployment in real-world applications.

Future research could extend this work to more complex CNN architectures, such as ResNet or DenseNet, with more techniques (e.g., Dropout and Skip connection) to gain a broader perspective on AF-WI interactions. Additionally, using larger and more complex datasets would provide more relevant insights for real-world applications, such as those found in image recognition, natural language processing, or autonomous systems.

References

[1] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[2] Chitra Desai and C Desai. Impact of weight initialization techniques on neural network efficiency and performance: A case study with mnist dataset. *International Journal Of Engineering And Computer Science*, 13(04), 2024.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[4] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? *Advances in neural information processing systems*, 31, 2018.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[8] Wei Hu, Lechao Xiao, and Jeffrey Pennington. Provable benefit of orthogonal initialization in optimizing deep linear networks. *arXiv preprint arXiv:2001.05992*, 2020.

[9] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[11] Alex Krizhevsky, Geoff Hinton, et al. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7):1–9, 2010.

[12] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*, 2017.

[13] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[14] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[15] Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

[16] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[17] Gaurav Kumar Pandey and Sumit Srivastava. Resnet-18 comparative analysis of various activation functions for image classification. In *2023 International Conference on Inventive Computation Technologies (ICICT)*, pages 595–601. IEEE, 2023.

[18] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Swish: a self-gated activation function. *arXiv: Neural and Evolutionary Computing*, 2017.

[19] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[20] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[22] Tomasz Szandała. Review and comparison of commonly used activation functions for deep neural networks. *Bio-inspired neurocomputing*, pages 203–224, 2021.

[23] Kit Wong, Rolf Dornberger, and Thomas Hanne. An analysis of weight initialization methods in connection with different activation functions for feedforward neural networks. *Evolutionary Intelligence*, 17(3):2081–2089, 2024.

[24] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402. PMLR, 2018.