Weight Initialisation Optimisation

Aadit Pani*, Ayush†, Megh Mavani‡, Mehul Goyal§
*1RVU23CSE002, †1RVU23CSE094, ‡1RVU23CSE260, §1RVU23CSE267
School of Computer Science and Engineering, RV University, Bengaluru, India

I. Introduction

Weight initialization plays a critical role in the performance and stability of deep neural networks. In the early stages of training, the choice of initialization can influence whether the network converges, how fast it learns, and the quality of the minima it reaches. Improper initialization may lead to vanishing or exploding gradients, especially in deeper networks, significantly hindering the training process. Over the years, several initialization schemes have been proposed, such as Xavier (Glorot) and He initialization, each tailored to specific activation functions and architectural constraints. These methods aim to preserve signal propagation and gradient stability across layers, laying a foundation for effective optimization.

Despite the success of these methods, the field lacks a unified understanding of how initialization methods interact with different architectural choices, activation functions, and data distributions. Moreover, much of the existing literature focuses on applying a single initialization strategy throughout the network, leaving the potential benefits of hybrid and distributional combinations underexplored.

This study is motivated by the hypothesis that better performance may be achieved not necessarily through the invention of new initialization paradigms, but by revisiting, combining, and systematically evaluating existing techniques. We explore how various initialization strategies—including but not limited to Xavier, He, LeCun, and their uniform and normal variants—perform when applied in different distributions and hybridized across layers or units. Our experimentation spans multiple neural architectures, datasets, and activation functions, maintaining constant model configurations to isolate the effects of initialization.

Through this research, we aim to empirically identify promising configurations that minimize cost functions more consistently and converge more efficiently during training. Furthermore, we attempt to provide mathematical intuition behind our best-performing combinations, and investigate the relationship between initialization choices and activation dynamics. While we do not propose a novel method per se, our goal is to offer insight into more optimal use of existing tools and contribute to a deeper understanding of weight initialization as a practical and theoretical concern.

Here is the GitHub repository of a working demonstration of our project: https://github.com/TheRevanite/Weight-Initialisation-Research

II. Literature Survey

A. What is Initialization

Weight initialization is a critical aspect of deep learning that sets the starting values of a network's weights before training. It significantly influences training stability, speed, and performance. Poor initialization can lead to problems like vanishing/exploding gradients or neuron symmetry, making learning ineffective.

Different initialization methods suit different architectures and activation functions. For instance, Xavier and LeCun initialization work well with sigmoid or tanh activations, while He initialization is ideal for ReLU. RNNs often require orthogonal initialization to maintain gradient flow, and transformers depend on careful scaling due to their depth.

Good initialization helps models converge faster, reduces computational cost, and improves learning. It works hand-in-hand with optimizers, learning rates, and model architecture, making it a fundamental component of deep learning design — not just a setup step.

1) Zero Initialization: Zero initialization sets all the weights in a neural network to zero. While this might seem like a simple and harmless choice, it fails in practice because it doesn't break symmetry between neurons. If all weights start with the same value, every neuron in a layer will compute the same output and receive the same gradient during backpropagation. This means that all neurons will learn in exactly the same way, rendering the network ineffective. However, it is still commonly used to initialize bias terms.

2) Random Initialization: Random initialization involves assigning weights randomly, usually from a uniform or normal distribution. It breaks the symmetry between neurons, enabling each neuron to learn different features. Without appropriate scaling, it may lead to vanishing or exploding gradients in deep networks.

3) He Initialization (Kaiming Initialization): He initialization is designed for networks using ReLU or its variants. It scales the weights using a factor of $\sqrt{2/n_{in}}$, where n_{in} is the number of input units to a layer. This maintains activation variance across layers and stabilizes training.

4) LeCun Initialization: Tailored for sigmoid and tanh activations, LeCun initialization scales weights to preserve activation variance. It prevents saturation and improves convergence, and is recommended for use with SELU activations.

5) Constant Initialization: In this method, weights are set to a fixed constant (e.g., 0.1 or 1). This does not break symmetry between neurons and thus is not used in practice for weights. It may be applicable in bias initialization.

6) Xavier Initialization (Glorot Initialization): Xavier initialization maintains signal magnitude across layers in both forward and backward passes. It sets the variance of weights to $\frac{2}{n_{in}+n_{out}}$ and is effective with sigmoid and tanh activations.

7) Standard Normal/Uniform Initialization: This method samples weights from $N(0,1)$ or $U(-1,1)$ distributions. It lacks scaling for layer dimensions, which may cause instability in deep networks. It's now mostly used for benchmarking.

8) All Zero Initialization: Setting all weights and biases to zero eliminates variability among neurons. This prevents learning, as all neurons behave identically. While biases can be zero-initialized, weights should not.

9) LeCun Uniform (LCUN) Initialization: LCUN samples weights from a uniform distribution, optimized for sigmoid, tanh, and SELU activations. It preserves activation variance across layers and helps prevent neuron saturation. It is well-suited for self-normalizing networks.

B. Literature Review

This section highlights key insights from selected works that shaped our understanding of weight initialization strategies in deep learning.

1) Weight Initialization Techniques for Deep Learning Algorithms in Remote Sensing: Recent Trends and Future Perspectives: Boulila et al. examine weight initialization techniques used in deep learning for remote sensing (RS) tasks such as land cover detection and disaster prediction. Reviewing 17 studies, they analyze methods including random, Xavier, He, and LeCun initialization. The study finds that He initialization works well with non-linear activations like ReLU, whereas Xavier and LeCun are better for differentiable functions like Sigmoid. While standard methods are widely used, the paper highlights the lack of RS-specific strategies and calls for theoretical models tailored to high-dimensional RS data. Autoencoders are suggested as a promising direction for handling uncertainty and missing data.

2) An Analysis of Weight Initialization Methods in Connection with Different Activation Functions for Feedforward Neural Networks: Wong et al. investigate the interplay between initialization techniques and activation functions in feedforward neural networks (FNNs). They compare random, Xavier, and Nguyen-Widrow initialization across functions like ReLU, tanh, log-sigmoid, and GELU. The results show that Xavier with linear activation provides the most consistent results, while Nguyen-Widrow performs well with tanh on non-linearly separable datasets. Random initialization consistently underperforms. The study concludes that weight initialization should align with both activation function and dataset characteristics, and suggests exploring modern activation functions like Swish or Mish in future research.

3) A Review on Weight Initialization Strategies for Neural Networks: Narkhede et al. present a detailed categorization of weight initialization strategies, including random, data-driven, hybrid, and pre-training-based approaches. Emphasis is placed on the popularity of variance scaling techniques (e.g., Xavier, He) for preserving activation stability. Pre-training methods like stacked autoencoders are shown to improve performance on small datasets. The study also explores PCA-based and k-means initialization, and highlights the use of evolutionary algorithms such as genetic algorithms and particle swarm optimization for non-iterative training models. Future work is encouraged in adaptive initialization methods and their application to architectures like transformers and graph neural networks.

4) Weight Initialization of Deep Neural Networks Using Data Statistics: Koturwar and Merchant propose a data-driven initialization method aimed at overcoming the limitations of generic techniques like Xavier and He, especially in the context of imbalanced datasets. Their approach aligns weights with dominant features in the data while preserving variance characteristics. Results show superior performance and faster convergence on datasets such as the Bee Image Dataset and Statefarm Distracted Driver Detection. Hybrid approaches combining data statistics with variance scaling are also discussed. The authors advocate for extending such methods to self-supervised learning and transformer-based architectures.

III. Finalising Parameters

A. Finalising Dataset

We selected the CIFAR-10 dataset, comprising 60,000 color images of size 32×32 across 10 classes. It balances complexity and computational feasibility, making it ideal for benchmarking weight initialization strategies. Its widespread use in the deep learning community also facilitates easier comparison with prior work.

B. Finalising Model

We adopted a Convolutional Neural Network (CNN) architecture due to its effectiveness in visual feature extraction and image classification. Following multiple iterations, a 9-layer CNN was finalized, offering a practical balance between depth, performance, and training efficiency. This architecture supports fine-grained control over kernel size, stride, and activation functions, aligning well with the goals of our study.

C. Evaluation Metrics Used

To evaluate the effectiveness of different weight initialization techniques, we utilized the following metrics:

1) Training Loss: Calculated using cross-entropy, training loss was monitored per epoch to assess how well the model minimized classification error.

2) Validation Accuracy: Tracked at each epoch, validation accuracy measured the model's ability to generalize. It also informed our early stopping mechanism to avoid overfitting.

3) Test Accuracy: After training, the best-performing model based on validation accuracy was tested on unseen data to obtain a final performance benchmark.

4) Early Stopping Criteria: Training was halted if validation accuracy failed to improve over a fixed number of epochs. This reduced overfitting and saved computational resources.

IV. Comparison of Initialization Methods

Table I at the end of this document summarizes the characteristics of various weight initialization techniques, their recommended activation functions, and typical model use cases. This comparison provides a useful overview of different initialization strategies, helping to identify the best methods based on the type of model and task.

V. Experimentation

Having reviewed existing initialization methods and their strengths and limitations, we proceed to explore novel approaches aimed at improving model convergence and performance across different architectures.

A. Poisson Scaled Initializer

The Poisson Scaled Initializer was inspired by drawing weights from a discrete distribution while controlling their variance using fan-in scaling. We sampled weights from a Poisson distribution and subtracted the mean (λ) to center the values around zero, aiming to optimize activation functions like ReLU or Mish, which benefit from zero-centered inputs.

$$W = (\text{Poisson}(\lambda = \text{fanin}) - \text{fanin}) \times \frac{1}{\text{fanin}}$$

The probability mass function for Poisson is:

$$P(x) = \frac{e^{-\lambda}\lambda^x}{x!}$$

Where x represents the number of occurrences. However, this method faced issues due to the non-symmetric, skewed nature of the Poisson distribution, particularly at lower λ values. A large portion of the initialized weights were positive, which biased the activations during forward propagation. For ReLU activations, this caused neurons to stay active continuously, limiting gradient diversity and degrading model performance. The discrete nature of Poisson values also lacked the fine-grained distribution found in other successful initializers.

B. Scaled Uniform Initializer

Next, we explored a custom Scaled Uniform Initializer. This method assigns initial values to weights using a uniform distribution dynamically scaled based on the input layer size (fan-in). The weights are generated within the range:

$$U\left(-\text{scale} \times \text{fanin}, \text{scale} \times \text{fanin}\right)$$

Where fan-in is the product of all input dimensions except the output dimension. This approach aimed to preserve activation variance and prevent vanishing or exploding gradients. The scale parameter allows flexibility, making this initializer tunable across architectures and activation functions.

Despite its theoretical soundness, this method underperformed in practice. The uniform distribution, while balanced, lacked the heavy tails that can help escape poor local minima in high-dimensional spaces. Moreover, its lack of directionality (unlike orthogonal matrices) made it less effective in deeper networks where gradient conditioning is crucial.

C. From Custom Distributions to Hybrid Strategies

After encountering the limitations of the above approaches, we reconsidered our strategy. Instead of relying on entirely new distributions, we combined existing, successful techniques in innovative ways. Specifically, we combined He Initialization and Orthogonal Initialization for their complementary benefits.

We found that:

- **He Initialization** worked well in the shallow layers of the network, ensuring the preservation of activation variance for signal propagation through ReLU functions. - **Orthogonal Initialization** proved beneficial in deeper layers, improving gradient flow and stabilizing convergence. Its full-rank matrices and unit condition number helped prevent vanishing or exploding gradients.

This hybrid strategy — utilizing He Initialization in shallow layers and Orthogonal Initialization in deeper layers — showed promising empirical results. To the best of our knowledge, this combination has not been explicitly documented in prior research.

D. Use of Orthogonal + He Initialization

To address the limitations of individual initialization methods, we explored the motivation behind combining He and Orthogonal initialization.

1) He Initialization (Kaiming): He Initialization is primarily suited for layers with ReLU activations. It preserves the variance of inputs during the forward pass, compensating for the sparsity introduced when ReLU sets negative outputs to zero.

Limitation: He Initialization does not enforce any structure or directionality in the weight matrix, which can lead to unstable Jacobians in deep networks.

2) Orthogonal Initialization: Orthogonal Initialization initializes weights such that the weight matrix is orthogonal, with a condition number of 1. This property promotes consistent gradient flow across layers and helps prevent vanishing or exploding gradients.

Limitation: Orthogonal Initialization does not consider the activation function and, as such, lacks variance adaptation.

3) Combined Benefits: - **He Scaling** maintains appropriate variance tailored to ReLU activations, ensuring that the signal variance remains consistent through the forward pass. - **Orthogonal Directionality** improves the conditioning of the weight matrix, ensuring stable gradient propagation through deep networks.

This hybrid approach seeks to improve training dynamics, particularly in deep networks, where both variance and directionality play a critical role in model performance.

VI. Mathematical Justification

A. He Initialization

He Initialization was designed to address a key issue with ReLU-based networks: preserving variance across layers despite ReLU cutting off half of the activations (i.e., setting negative values to zero).

Let's assume:

- x is the input to a layer.
- w is the weight.
- $y = w \cdot x$.

If x has variance σ^2, then the output variance becomes:

$$\text{Var}(y) = \text{Var}(w \cdot x) = E[w^2] \cdot \text{Var}(x)$$

We solve for $E[w^2] = \frac{1}{n}$ for standard activations, but for ReLU, since it kills half the signal, we compensate by multiplying by a factor of 2:

$$E[w^2] = \frac{2}{\text{fanin}}$$

For the normal version, the distribution is:

$$w \sim N(0, \frac{2}{\text{fanin}})$$

For the uniform version (derived from setting uniform variance equal to normal):

$$w \sim U\left(-\sqrt{\frac{6}{\text{fanin}}}, \sqrt{\frac{6}{\text{fanin}}}\right)$$

B. Orthogonal Initialization

In deep networks, even if variance is preserved, correlations between neurons can cause issues. Orthogonal matrices address this by maintaining directionality and independence. A matrix W is orthogonal if $W^T W = I$, meaning it preserves both length and angles, which aids in stable signal flow.

Suppose we initialize weights as a random matrix, then perform QR decomposition, $W = QR$. This decomposition is useful because the matrix Q retains direction and norm properties, which is exactly why it's used in orthogonal weight initialization in neural networks.

Let's assume you have a matrix A_R made up of n linearly independent column vectors $a_1, a_2, \ldots, a_n$. Here's how the QR decomposition works via classical Gram-Schmidt:

1) Initialize the process and normalize it to get the first orthonormal vector q_1:

$$q_1 = \frac{a_1}{\|a_1\|}$$

2) Orthogonalize each new column. For the second column, subtract its projection on q_1:

$$u_2 = a_2 - \text{proj}_{q_1}(a_2) = a_2 - (q_1^T a_2)q_1$$

Then normalize:

$$q_2 = \frac{u_2}{\|u_2\|}$$

3) Repeat this orthogonalization and normalization for all remaining vectors $a_3, a_4, \ldots, a_n$:

$$u_k = a_k - \sum_{j=1}^{k-1}(q_j^T a_k)q_j$$

and

$$q_k = \frac{u_k}{\|u_k\|}$$

4) Construct the matrix Q and R:

$$Q = [q_1, q_2, \ldots, q_n], \quad R_{ij} = q_i^T a_j \quad \text{(for all } i, j)$$

Where Q is an orthogonal matrix, and optionally, we scale Q by a gain factor:

- 2 for ReLU,
- $\sqrt{5}$ for tanh,
- 1 by default.

VII. Expansion of Testing

Our method combines statistical scaling with structured directionality, ensuring both stable variance and improved signal propagation. We tested it across multiple architectures and datasets, evaluating training stability, convergence speed, and final accuracy.

A. Changing Datasets

Model: 9-layer CNN with BatchNorm, Dropout, ReLU
Initialization:

- He Initialization $\rightarrow$ Shallow layers (Conv1–Conv4)
- Orthogonal Initialization $\rightarrow$ Deep layers (Conv5–Conv6, FC1)

Optimizer: Adam (lr=0.001, weight_decay=1e-4)
Early Stopping: Enabled (patience=7)
Epoch Limit: 50
Hardware: CUDA-enabled GPU

Table II at the end of this document summarizes the validation accuracy of different models across various datasets. This table highlights how our model performs when trained and tested on popular datasets like CIFAR-10, CIFAR-100, SVHN, QMNIST and STL-10.

B. Changing Models

Dataset: CIFAR-10
Initialization:

- He Initialization $\rightarrow$ Shallow layers
- Orthogonal Initialization $\rightarrow$ Deep layers

Optimizer: Adam (lr=0.001, weight_decay=1e-4)
Early Stopping: Enabled (patience=7)
Epoch Limit: 50
Hardware: CUDA-enabled GPU

The performance of various models is summarized in Table III at the end of this document. This table provides insights into how different architectures such as perform on the dataset, including their validation accuracy and notable characteristics.

C. Changing both Dataset and Model

Initialization:

- He Initialization $\rightarrow$ Shallow layers (Conv1–Conv4)
- Orthogonal Initialization (gain=$\sqrt{2}$) $\rightarrow$ Deep layers (Conv5+, FC)

Optimizer: Adam (lr=0.001, weight_decay=1e-4)
Early Stopping: Enabled (patience=7)
Epoch Limit: 50
Activation: ReLU
Normalization: BatchNorm applied where possible
Hardware: CUDA-enabled NVIDIA GPU

The performance of different models on multiple datasets is shown in Table IV at the end of this paper. This table provides an overview of how different architectures, including GoogleNet, ResNet-18, and CaffeNet, perform on datasets including STL-10, QMNIST, and CIFAR-100, with notable results observations for each combination.

The hybrid initialization strategy demonstrated strong cross-architecture consistency, enabling rapid and stable convergence across various CNN models, including GoogleNet, ResNet18, VGG16, and CaffeNet, with minimal tuning. By combining He initialization for localized filters and Orthogonal initialization for deeper layers, it preserved effective weight scales and gradient flow, even in complex network architectures. Models showed early performance plateaus—often within 20 epochs—and maintained stable training curves without relying on complex activations or learning rate tricks. Notably, older or deeper architectures like VGG16 benefited significantly in deeper layers, while models with skip connections, like ResNet18, synergized well with the initialization scheme. Overall, this approach proved robust and transferable, offering a plug-and-play solution for stable training across diverse architectures.

VIII. Conclusion

In this study, we proposed a novel hybrid initialization scheme—combining He and Orthogonal initialization—designed for deep convolutional neural networks utilizing ReLU activations. This approach effectively balances variance preservation with stable gradient propagation by employing He initialization in shallow layers and orthogonal matrices in deeper layers. The hybrid method addresses critical issues in deep networks, such as vanishing/exploding gradients and poor convergence behavior.

We validated the method through comprehensive experiments across benchmark datasets including CIFAR-10, CIFAR-100, SVHN, QMNIST, and STL-10. Evaluation was also performed across various architectures such as our custom 9-layer CNN, VGG-16, ResNet-50, GoogLeNet, and CaffeNet. Empirical results demonstrate that our approach consistently enhances training stability, speeds up convergence, and yields competitive or superior performance in comparison to conventional initialization strategies. Furthermore, a mathematical justification was provided, explaining how this hybrid technique preserves gradient norms and improves conditioning in deep layers.

IX. Future Work

This work lays the foundation for several promising research directions:

- **New Hybrid Initializations:** We intend to investigate alternative combinations tailored to different activation functions or architectural designs.
- **Data-Aware Initialization:** Exploring initializations conditioned on dataset statistics or learned representations from early layers.
- **Task-Specific Initialization:** Extending the approach to domains beyond image classification, such as NLP and reinforcement learning.
- **Learnable Initialization:** Investigating frameworks where initialization parameters are learned through meta-training or differentiable search.

Overall, this study highlights the critical yet underexplored role of initialization design in achieving stable and efficient deep network training.

References

[1] W. Boulila, M. Driss, E. Alshanqiti, M. Al-Sarem, F. Saeed, and M. Krichen, "Weight initialization techniques for deep learning algorithms in remote sensing: Recent trends and future perspectives," *Advances on Smart and Soft Computing: Proceedings of ICACIn 2021*, pp. 477–484, 2022.

[2] K. Wong, R. Dornberger, and T. Hanne, "An analysis of weight initialization methods in connection with different activation functions for feedforward neural networks," *Evolutionary Intelligence*, vol. 17, no. 3, pp. 2081–2089, 2024.

[3] M. V. Narkhede, P. P. Bartakke, and M. S. Sutaone, "A review on weight initialization strategies for neural networks," *Artificial Intelligence Review*, vol. 55, no. 1, pp. 291–322, 2022.

[4] S. Koturwar and S. Merchant, "Weight initialization of deep neural networks (DNNs) using data statistics," *arXiv preprint arXiv:1710.10570*, 2017.

[5] V. Kunc and J. Kléma, "Three decades of activations: A comprehensive survey of 400 activation functions for neural networks," *arXiv preprint arXiv:2402.09092*, 2024.

[6] S. Yadav, "Weight initialization techniques in neural networks," *Towards Data Science*, Nov. 9, 2018. [Online]. Available: https://towardsdatascience.com/

[7] P. Kashyap, "Mastering weight initialization in neural networks: A beginner's guide," *Medium*, Nov. 2, 2024. [Online]. Available: https://medium.com/

[8] W. Boulila, E. Alshanqiti, A. Alzahem, A. Koubaa, and N. Mlaiki, "An effective weight initialization method for deep learning: Application to satellite image classification," *Expert Systems with Applications*, vol. 254, 124344, 2024.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[10] D. Mishkin and J. Matas, "All you need is a good init," *arXiv preprint arXiv:1511.06422*, 2015.

[11] P. Khurana, "Activation functions and initialization methods," *Medium*, Feb. 4, 2020. [Online]. Available: https://medium.com/

[12] Wikipedia contributors, "Activation function," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Activation]_function

[13] Google Developers, "Neural networks: Activation functions," *Google for Developers*. [Online]. Available: https://developers.google.com/

[14] L. Datta, "A survey on activation functions and their relation with Xavier and He normal initialization," *arXiv preprint arXiv:2004.06632*, 2020.

TABLE I
COMPARISON OF WEIGHT INITIALIZATION METHODS

Initialization Method	**Activation Function**	**Models / Usage Notes**
Zero Initialization	None	Not recommended due to symmetry problem. Sometimes used in simple models like linear or logistic regression.
Random Initialization	Any (ReLU, Sigmoid, Tanh, etc.)	Used as a baseline across all models including MLPs, CNNs, RNNs, GANs, and Transformers.
He Initialization	ReLU, Leaky ReLU, ELU	Ideal for deep networks (e.g., VGG, ResNet, EfficientNet, MLPs, ViT) where ReLU helps avoid dead neurons.
LeCun Initialization	Sigmoid, Tanh, SELU	Used in shallow networks, LeNet, RNNs, and self-normalizing networks with SELU.
Constant Initialization	ReLU, Softmax, Sigmoid (Biases)	Rarely used for weights; sometimes applied for bias initialization (e.g., 0.01 in CNNs/MLPs).
Xavier Initialization	Sigmoid, Tanh, Softmax	Effective in MLPs, LSTMs, CNNs, RNNs. Seen in AlexNet, early ResNets, and architectures with saturating activations.
Standard Normal / Uniform	Any (ReLU, Sigmoid, Tanh)	Simple initialization for older or toy models. Less effective than He/Xavier.
All Zero Initialization	None	Not recommended due to symmetry issues. May be used for debugging in linear/logistic regression.

TABLE II
PERFORMANCE ACROSS DIFFERENT DATASETS

Dataset	**Validation Accuracy**	**Notable Observations**
SVHN	93.0%	Achieved consistently high accuracy with fast convergence. Validation accuracy remained in the 90s, indicating robust generalization for digit classification.
CIFAR-10	98.51%	Significant boost in performance. Model reached above 90% accuracy within the first 15–20 epochs.
CIFAR-100	62.1%	Moderate performance, expected due to class granularity. Hybrid initialization kept gradients stable in more complex hierarchies.
QMNIST	92.6%	Strong performance, considering grayscale nature. He initialization handled edge features, while Orthogonal layers captured structural variation.
STL-10	78.4%	Robust performance with fewer training samples per class and larger image sizes. Hybrid initialization maintained gradient flow.

TABLE III
PERFORMANCE ACROSS DIFFERENT MODELS ON CIFAR-10

Model	**Validation Accuracy**	**Notable Characteristics**
CNN (Custom)	91%	Simpler architecture with fewer layers, reduced overfitting risk, and effective tuning for CIFAR-10 dataset.
VGG-16	89%	Deep architecture with a large number of parameters. Slightly lower performance due to lack of skip connections and higher overfitting tendency.
ResNet-50	93%	Highest validation accuracy among tested models. Residual connections help alleviate vanishing gradient problems and improve gradient flow.

TABLE IV
PERFORMANCE ACROSS DIFFERENT MODELS AND DATASETS

Model	**Dataset**	**Validation Accuracy**	**Observations**
GoogleNet	STL-10	74.82%	Efficient training; Orthogonal-initialized inception blocks aided generalization despite limited samples.
GoogleNet	QMNIST	99.01%	Excellent on grayscale digit data. Early layers captured edges; deep layers learned structure.
ResNet18	STL-10	75.42%	Skip connections maintained sharp gradients on large input sizes.
ResNet18	QMNIST	99.39%	Smooth learning curve with rapid convergence.
CaffeNet	STL-10	75.4%	Stable convergence with Orthogonal + He initialization supporting gradient flow.
CaffeNet	QMNIST	91.3%	Slightly behind deeper models but effective for simpler architecture.
9-layer CNN	STL-10	78.4%	Hybrid initialization performed well with minimal tuning and stable training.
9-layer CNN	QMNIST	92.6%	Solid accuracy on digit classification.
ResNet18	CIFAR-10	92.7%	High accuracy and fast convergence due to residual connections.
9-layer CNN	CIFAR-10	82.4%	Reliable training and convergence; effective with hybrid initialization.
VGG-16	CIFAR-10	88.7%	Strong performance, but prone to overfitting without regularization.
GoogleNet	CIFAR-10	83.3%	Deep inception layers enabled robust generalization with orthogonal init.
CaffeNet	CIFAR-10	84.8%	Solid results despite shallower depth, helped by effective initialization.
ResNet18	CIFAR-100	72.5%	Handled 100 classes well with efficient gradient propagation.
VGG-16	CIFAR-100	69.9%	Competitive accuracy; could benefit from skip connections or deeper regularization.
ResNet18	SVHN	98.5%	Performed well on high-res digit data; fast convergence.
VGG-16	SVHN	96.6%	High accuracy, though training was comparatively slower.
9-layer CNN	SVHN	97.2%	Hybrid initialization helped maintain gradient flow; strong result.