Research Article GU J Sci, Part A, 13(1): 269-305 (2026) 10.54287/gujsa.1840916

Gazi University
Journal of Science
PART A: ENGINEERING AND INNOVATION
http://dergipark.org.tr/gujsa

The Effect of Si-CL Loss Function and Different Optimization Algorithms in Improving CNN Performance

Yasin OZKAN[1*] Pakize ERDOGMUS[2]

[1] Department of Computer Technologies, Zonguldak Bulent Ecevit University, Zonguldak, Türkiye
[2] Department of Computer Engineering, Faculty of Engineering, Duzce University, Duzce, Türkiye

Keywords

Classification
Loss Function
MNIST
Optimization Algorithms

Abstract

Convolutional Neural Networks (CNNs) used for image classification often have complex architectures involving large images, time-costly training processes, and a large number of layers and hyperparameters. Therefore, improving the accuracy of CNN is a challenging process that requires time, resources and specialized knowledge. In this study, to improve the performance of CNN models, experiments were conducted on the MNIST, EMNIST, and Fashion-MNIST datasets using different optimization algorithms and a loss function (Si-CL) from the literature. The findings of the study reveal the effects of loss functions and optimization algorithms on model performance in detail. The SGDM, Adam, RMSProp, RMSProp, AdaMax, AdaDelta and AdaGrad optimization algorithms used during the experiments are examined and the results show that the Adam algorithm performs the best in terms of both training accuracy and test accuracy. The SGDM algorithm was particularly effective at larger batch sizes and low learning rates, but required longer training times compared to the Adam algorithm. The Si-CL loss function used in the study performed better than the traditional cross entropy loss. The model trained with the Si-CL loss function achieved higher results in terms of both training and test accuracy, reduced training time and lower loss value. This allowed the model to learn faster and more efficiently.

Cite

Ozkan, Y., & Erdoğmus, P. (2026). The Effect of Si-CL Loss Function and Different Optimization Algorithms in Improving CNN Performance. *GU J Sci, Part A*, *13*(1), 269-305. doi:10.54287/gujsa.1840916

Author ID (ORCID Number)

0000-0002-2029-0856 Yasin OZKAN
0000-0003-2172-5767 Pakize ERDOGMUS

Article Process

Submission Date 12.12.2025
Revision Date 20.01.2026
Accepted Date 02.03.2026
Published Date 31.03.2026

1. INTRODUCTION

Today, deep learning has made significant advances, particularly in the fields of image processing and computer vision. Much of this progress is based on the development of artificial neural networks called CNNs, which work specifically with image data. CNNs, proposed by LeCun and colleagues, are used in a wide range of applications due to their high success in feature extraction and classification stages (LeCun et al., 1998). CNNs demonstrate superior performance in many areas such as image classification, object detection, face recognition, and medical image analysis (LeCun et al., 1998; Krizhevsky et al., 2012).

The training process of CNNs is based on the updating of weights, which is performed by optimization algorithms. Optimization algorithms iteratively update the network's parameters to minimize the loss function. This process directly affects the model's overall performance and training time. Commonly used optimization

*Corresponding Author, e-mail: yasin.ozkan@beun.edu.tr

This work is licensed under Creative Commons Attribution-ShareAlike 4.0 International.

algorithms include methods such as Stochastic Gradient Descent (SGD), Adam, RMSprop, and AdaGrad (Goodfellow et al., 2016).

The training of CNNs relies on optimization algorithms that iteratively update network parameters to minimize the loss function. Commonly used optimizers include SGD and its variants, as well as adaptive methods such as Adam, RMSProp, and AdaGrad. Although these algorithms differ in their update mechanisms, their performance may vary depending on the loss function, dataset characteristics, and model architecture.

The Adam (Adaptive Moment Estimation) algorithm attempts to overcome optimization problems through adaptive learning rates. Introduced by Kingma and Ba (2014), Adam provides a more effective and stable convergence process by adaptively adjusting learning rates for each parameter and using moment estimates. Adam calculates both first-moment (mean) and second-moment (variance) statistics, which allows the algorithm to better manage variability and noise during the learning process. Especially with large datasets and complex models, the Adam algorithm generally produces faster and more reliable results (Reddi et al., 2018).

RMSprop (Root Mean Square Propagation) and AdaGrad (Adaptive Gradient Algorithm) algorithms are designed specifically for scenarios requiring irregular data sets and variable learning rates. RMSprop was proposed by Tieleman and Hinton and offers a more balanced update process by adjusting the learning rate separately for each parameter (Tieleman & Hinton, 2012). This method ensures that parameters are updated more effectively and prevents excessive fluctuations in the learning rate. AdaGrad facilitates the learning of rare features that are frequently encountered by using adaptive learning rates for each parameter (Duchi et al., 2011). Other widely used adaptive optimization algorithms, such as AdaDelta and AdaMax, have also been introduced to address specific limitations of earlier methods. AdaDelta improves upon AdaGrad by limiting the accumulation of past gradients and eliminating the need for a manually defined learning rate, thereby providing more stable learning dynamics. AdaMax, a variant of Adam based on the infinity norm, offers improved numerical stability in certain optimization scenarios.

Despite extensive research on optimization algorithms and loss functions, comprehensive and controlled comparisons of widely used optimizers under identical CNN architectures remain limited. In particular, the interaction between optimization strategies and recently proposed composite loss functions has not been systematically analyzed. This study aims to address this gap by providing a controlled comparative evaluation of popular optimization algorithms under fixed architectural settings and by examining the behavior of the Si-CL loss function in multi-class classification tasks.

Additionally, the proposed Si-CL function is discussed in detail, its adaptability to multi-class classification problems is theoretically justified and experimentally validated. The contribution of Si-CL to the model's convergence speed, class separation power, and generalization performance is examined in detail.

Model performance is not limited to the MNIST dataset; the methods are also tested on the EMNIST dataset to evaluate the model's external validity and robustness. These additional analyses demonstrate that the Si-CL loss and the optimization strategies used work consistently across different data structures.

In conclusion, this study not only systematically reveals the effects of optimization algorithms on CNN training but also provides a unique contribution to the literature by theoretically and experimentally supporting the potential of a new loss function for multi-class problems.

The remainder of the paper is organized as follows: Section 2 briefly presents related work in the field and information about CNNs. Section 3 describes the dataset used and the types of optimization algorithms employed. The results of the experiments conducted in the fourth chapter are presented. The fifth chapter concludes the article by providing information on the results of this study and what can be done next.

2. BACKGROUND AND LITERATURE REVIEW

CNNs and optimization algorithms are cornerstones of the deep learning field. This section summarizes studies on the performance, advantages, and limitations of various optimization algorithms used in CNNs, along with a general overview of neural networks.

2.1. Overview of Neural Network Optimization

Over the years, research on improving and developing new optimization algorithms has played a vital role in improving deep learning architectures. Neural network training is an optimization problem that aims to find the optimal parameters by using gradient descent algorithms to transform the inputs in the training dataset into output labels with minimum loss. The optimization problem in neural networks is a process aimed at minimizing the loss function to achieve the best performance during the model's learning process. The loss function represents the difference between the model's predicted results and the actual results, and this difference must be minimized to optimize the model's accuracy. The optimization problem of neural networks is mathematically presented in Equation (1) (Kingma & Ba, 2014):

$$\theta^* = argmin\, L(\theta; X, Y) \tag{1}$$

Here, Θ represents the learnable parameters of the neural network model, L represents the loss function as the error between the model's predicted values and the actual values, X represents the input data, Y represents the actual labels, and Θ* represents the optimal parameters that minimize the loss function.

Optimization is one of the fundamental components of deep learning that helps train a model better during backpropagation when weights are adjusted to minimize the loss error, while also addressing the dimensionality problem (Goodfellow et al., 2016). The most popular optimization algorithms used in deep learning libraries such as Caffe, Lasagne, and Keras are: SGD (Robbins & Monro, 1951), RMSProp (Hinton et al., 2012), AdaGrad (Zeiler, 2012), AdaDelta (Sutskever et al., 2013), Adam (Kingma & Ba, 2014), AdaMax

(Kingma & Ba, 2014). Researchers continue to develop optimizers to achieve better generalization, as seen in Yazan and Talu (2017).

2.2. Literature Review

Gradient-based optimization algorithms have long been the backbone of deep learning training. Early comparative studies by Yazan and Talu (2017) evaluated five widely used optimizers—SGD, Adam, AdaGrad, AdaDelta, and RMSProp—based on convergence speed, numerical stability, and parameter update behavior on a stacked denoising autoencoder architecture, reporting faster convergence for AdaDelta, although dataset details were not specified. Similarly, Arsenault et al. (2014) compared gradient descent, stochastic gradient descent, semi-stochastic gradient descent, and stochastic average descent on synthetic data and the MNIST dataset using logistic and softmax regression, showing that SGD generally outperformed standard GD, while hybrid variants achieved higher accuracy within reasonable training times. Hallen (2017) further investigated GD and SGD on MNIST for linear and multinomial logistic regression, demonstrating notable differences in accuracy and convergence efficiency between the two approaches.

Beyond empirical comparisons, Ruder (2016) provided a comprehensive and intuitive overview of modern gradient descent optimization algorithms, emphasizing the importance of adaptive learning rate methods for deep and complex neural networks while also highlighting challenges such as unstable convergence and poor generalization. Complementing this perspective, Shalev-Shwartz et al. (2017) analyzed the root causes of gradient descent failures in deep learning, attributing them to factors such as low signal-to-noise ratio, vanishing gradients caused by poor conditioning, and slow training dynamics. These foundational studies collectively indicate that a unified and comprehensive evaluation of popular optimizers on CNN-based image classification tasks remains limited.

Among adaptive optimizers, Adam has received significant attention due to its ability to combine momentum and adaptive learning rates. Kingma and Ba (2014) introduced Adam and demonstrated its faster convergence and improved stability across various deep learning tasks, including CNNs. Reddi et al. (2018) later examined both theoretical and practical limitations of Adam, showing that while it offers stable and fast convergence, its performance can be further improved through refined hyperparameter settings and modified variants. Building on this line of work, Liu et al. (2023) proposed an enhanced Adam variant incorporating adaptive coefficients, composite gradients, and randomized block coordinate descent, achieving faster convergence, higher accuracy, and reduced computational cost in both benchmark classification tasks and real-world regression problems.

Recent studies have increasingly focused on hybrid and enhanced optimization strategies to overcome local optimum stagnation. Yassine et al. (2024) introduced a hybrid optimizer that integrates gradient-based search with the Multi-Verse Optimizer (MVO), demonstrating superior convergence speed and classification performance in breast cancer diagnosis compared to classical gradient-based and metaheuristic approaches. Similarly, Mokhtar et al. (2025) proposed Dual Enhanced SGD (DESGD), which adaptively adjusts

momentum and step size, outperforming SGDM and achieving competitive results with Adam on benchmark functions and the MNIST dataset.

In parallel with optimizer development, loss-function innovation has emerged as a critical research direction for improving robustness and generalization. Chen et al. (2022) proposed a kernel-based MSE loss to better capture nonlinear error structures in wind speed forecasting, significantly improving robustness to outliers compared to standard MSE. Nouri and Seyyedsalehi (2023) introduced a novel loss function for iterated autoencoders that constrains eigenvalues to ensure stable attractor learning, achieving superior robustness and associative memory performance on MNIST. Akhtar et al. (2025) presented HawkEye, a robust loss function integrating boundedness, smoothness, and an insensitive zone, leading to improved noise robustness and generalization across multiple benchmark datasets. Yu et al. (2025) proposed an adjustable loss function for photovoltaic power forecasting, demonstrating consistent performance gains over MAE, MSE, Huber, and Log-Cosh losses across multiple prediction horizons. Finally, Lin et al. (2025) introduced a structure-oriented loss that encodes spatial and positional relationships for point cloud semantic segmentation, significantly improving segmentation accuracy and mean IoU through the incorporation of structural priors.

2.3. Optimization Algorithms

Optimization algorithms are crucial in training deep neural networks, including CNNs. These algorithms adjust the network's weights to minimize the loss function and improve the network's performance. This study provides an overview of the SGDM, Adam, RMSProp, AdaMax, AdaDelta and AdaGrad optimization algorithms, highlighting their key features and benefits.

2.3.1. Stochastic Gradient Descent with Momentum (SGDM)

SGD updates the model's weights using only one example or a small group of examples at each step. This reduces computational costs when working with large datasets and enables the model to learn quickly. The advantages of SGD are that it eliminates the need to keep all the data in memory and can accelerate the learning process by updating the model parameters more frequently. The disadvantage of this optimization algorithm is noise. During each update in this algorithm, the network may contain a certain amount of noise, which can destabilize the learning process (Bottou, 2010). The SGDM algorithm is an improved version of the classic SGD algorithm. The momentum term helps prevent getting stuck in local minima during the optimization process and aids in faster convergence. Momentum corrects the update direction by adding the weighted average of previous gradients. The update of the SGDM algorithm is presented in Equations (2) and (3).

$$v_t = \beta . v_{t-1} + a . g_t \tag{2}$$

$$\theta_t = \theta_{t-1} - v_t \tag{3}$$

In the Equation (2) and (3), β represents the momentum coefficent, v_t and v_{t-1}reprsent current and previous velocity respectively, θ_t and

θ_{t-1} represent the current and previous parameters, g_t represent the gradients of parameters and lastly α represent the learning rate.

2.3.2. AdaGrad

AdaGrad (Adaptive Gradient Algorithm) is an optimization algorithm used in machine learning and deep learning. Its primary function is to make the model training process more effective by adapting the learning rate for each parameter. This adaptation aligns the learning rate of each parameter by utilizing past information from the gradients. This approach is particularly useful in situations where data sets are insufficient and rare features are important. AdaGrad dynamically adjusts the learning rate by calculating the cumulative sum of the squares of the gradients' past values. As the magnitude of this cumulative sum increases for each parameter, the learning rate decreases; this leads to faster reduction in the learning rates of frequently updated parameters. One of AdaGrad's greatest advantages is that it automatically adjusts the learning rate for parameters. This feature ensures that parameters are updated at more appropriate speeds according to gradients of different magnitudes. It particularly helps rare features learn faster in under-sampled datasets. This enables the model to learn such rare but important features more effectively (Kingma & Ba, 2014).

However, AdaGrad also has some disadvantages. Particularly in long training processes, the cumulative sum of the squares of the gradients can increase rapidly, causing the learning rate to decrease very quickly. This can lead to a slowdown in the model's learning process and ultimately cause the learning rate to drop to very low levels. Consequently, AdaGrad may lose its effectiveness in some cases during longer training processes. The mathematical formulation of the AdaGrad algorithm shows how learning rates are dynamically updated in this manner and is presented in Equations (4) and (5).

$$G_t = G_{t-1} + g_t^2 \tag{4}$$

$$\theta_t = \theta_{t-1} - \frac{a}{\sqrt{G_t + \epsilon}} . g_t \tag{5}$$

In the equations, Θt represents the value of the parameters at iteration t, a represents the learning rate, Gt represents the sum of the gradient squares, gt represents the gradient calculated at iteration t, and ϵ represents a small constant for numerical stability for division by zero.

2.3.3. RMSprop

RMSprop (Root Mean Square Propagation) is an optimization algorithm widely used in deep learning and machine learning applications. RMSprop dynamically adjusts the learning rate for each parameter by calculating the moving average of the squares of the gradients. This approach can be particularly effective when gradients change rapidly and have high variance. The basic principle of RMSprop is to use the cumulative average of the squares of the gradients when updating the learning rate for each parameter. This provides a more stable and balanced learning process by taking into account the magnitude of the gradients in the past. The average of the squares of the gradients reduces the variability of the gradients over time and

adapts the learning rate according to this variability. This stabilizes the learning process, especially with gradients that have high variance and rapidly changing gradients.

However, RMSprop also has some disadvantages. It can be difficult to properly adjust the learning rate and other hyperparameters. The RMSprop algorithm contains several hyperparameters that must be carefully adjusted to achieve good performance. The correct selection of these hyperparameters during model training can directly affect the effectiveness of the algorithm (Tieleman & Hinton, 2012). The mathematical formulation of the RMSprop algorithm shows how this dynamic learning rate adjustment is performed and how gradient variance is controlled, and is presented in Equations (6) and (7).

$$s = \beta . s + (1 - \beta) . g_t^2 \tag{6}$$

$$\theta_t = \theta_{t-1} - a . \frac{g_t}{\sqrt{s} + \epsilon} \tag{7}$$

In the equations, Ɵt represents the value of the parameters at iteration t, a is the learning rate, s is weighted average of gradient squares, β is the moving average coefficient, gt is the gradient calculated at iteration t, and ϵ is a small constant for numerical stability.

2.3.4. Adam

The Adam (Adaptive Moment Estimation) algorithm is an optimization method widely used in deep learning and machine learning. Adam makes the model training process more efficient by dynamically adjusting the learning rate for each parameter. This algorithm improves the learning process by using cumulative estimates of both the squares of the gradients and the moments of the gradients. The basic principle of Adam is to use two separate estimates that move on both the first moment (mean) and the second moment (mean of the squares) of the gradients. The first-moment estimator represents the average of the gradients over time, while the second-moment estimator represents the average of the squares of the gradients. These estimators dynamically adjust the learning rate based on the past values of the gradients. As a result, this method adapts the learning rate for each parameter according to both the magnitude and variance of the gradients.

One of the key advantages of the Adam algorithm is that it advances the learning process quickly and efficiently. The combined use of first and second moment estimates of gradients enables the learning rate to be adjusted more effectively based on the variability of the gradients. This feature helps the algorithm deliver a generally faster and more stable learning process. However, Adam also has some disadvantages. In particular, it can sometimes be difficult to set the hyperparameters appropriately, which directly affects the algorithm's performance. The Adam algorithm can produce different results depending on the values of the hyperparameters, which may require careful hyperparameter tuning during the model training process. These hyperparameters must be carefully selected to optimize the algorithm's performance (Kingma & Ba, 2014). The mathematical formulation of the Adam algorithm is presented in Equations (8), (9), (10), (11), and (12).

$$m_t = \beta_1 . m_{t-1} + (1 - \beta_1) . g_t \quad (8)$$

$$v_t = \beta_2 . v_{t-1} + (1 - \beta_2) . g_t^2 \quad (9)$$

$$m_t = \frac{m_t}{1 - \beta_1^t} \quad (10)$$

$$v_t = \frac{v_t}{1 - \beta_2^t} \quad (11)$$

$$\theta_t = \theta_{t-1} - a . \frac{m_t}{\sqrt{v_t} + \epsilon} \quad (12)$$

In the equations, Θt represents the value of the parameters at iteration t, a is the learning rate, m_t is the first moment estimate, v_t is the second moment estimate, β_1 and β_2 are hyperparameters, g_t is the gradient calculated at iteration t, and ϵ is a small constant for numerical stability.

2.3.5. AdaDelta

AdaDelta is an improved version of AdaGrad and offers a more dynamic approach, particularly to prevent the learning rate from decreasing rapidly. In the AdaGrad algorithm, the learning rate is adjusted using the cumulative sum of the squares of the gradients, but this approach can cause the learning rate to decrease rapidly during long training processes. AdaDelta aims to address this issue by providing a more balanced and efficient learning process.

The key innovation of AdaDelta is to prevent the learning rate from decreasing excessively by dynamically updating the past information of the gradients. Unlike AdaGrad, AdaDelta uses the moving average of gradients' squares instead of the cumulative sum of gradients' squares. This moving average manages the variance of gradients more effectively, thereby preventing the learning rate from decreasing rapidly. AdaDelta determines the appropriate learning rate for each parameter by taking into account the variability of the gradients, but instead of keeping the cumulative sum of the gradients' squares, it only considers the latest gradient information. This makes the learning process more dynamic and memory-efficient (Sutskever et al., 2013).

One of AdaDelta's biggest advantages is that it prevents the learning rate from dropping too quickly. This ensures that the model's training process takes longer and yields better results. However, the AdaDelta algorithm requires careful tuning of hyperparameters that can affect performance. In particular, selecting appropriate values for hyperparameters can directly affect the algorithm's effectiveness, and these settings may need to be carefully adjusted. The mathematical formulation of the AdaDelta algorithm shows how it dynamically updates gradient information and manages the learning rate. This formulation is presented in Equations (13), (14), (15), and (16).

$$g_t = \nabla_\theta J\,(\theta_{t-1}) \quad (13)$$

$$E[g^2]_t = p . E[g^2]_{t-1} + (1 - p) . g_t^2 \quad (14)$$

$$\Delta\theta_t = -\frac{\sqrt{E[\Delta\theta^2]_{t-1}+\epsilon}}{\sqrt{E[g^2]_t+\epsilon}}.g_t \quad (15)$$

$$E[\Delta\theta^2]_t = p.E[\Delta\theta^2]_{t-1} + (1-p).(\Delta\theta_t)^2 \quad (16)$$

In the equations, gt denotes the gradient calculated at iteration t, $E[g^2]t$ denotes the exponential moving average of the gradient squared, $E[\Theta^2]t$ denotes the exponential moving average of the parameter updates squared, p denotes the exponential moving average coefficient, and ϵ denotes a small constant for numerical stability.

2.3.6. AdaMax

AdaMax is an optimization method developed as an extension of the Adam algorithm, and this extension aims to overcome some limitations of the Adam algorithm. Specifically, AdaMax aims to provide a more stable and reliable learning process by using the L2 norm (Euclidean norm) instead of the infinity norm used in Adam. The Adam algorithm dynamically adjusts learning rates using estimates of the first and second moments of the gradients. This determines the learning rate for each parameter adaptively based on the past information of the gradients. However, the infinity norm used in Adam is a type of norm used to limit large values of gradients, and in some cases, this approach can lead to instability (Kingma & Ba, 2014).

AdaMax uses the L2 norm instead of the infinity norm to eliminate these instabilities. The L2 norm is based on the average of the squares of the gradients and generally provides a more stable measurement. This change ensures that AdaMax offers a more reliable and stable optimization process. These features of AdaMax help to manage large gradients more effectively and thus contribute to obtaining more consistent results during the model training process (Sutskever et al., 2013). The mathematical formula for AdaMax is presented in Equations (17), (18), and (19).

$$m_t = \beta_1.m_{t-1} + (1-\beta_1).g_t \quad (17)$$

$$u_t = \max(\beta_2.u_{t-1}, |g_t|) \quad (18)$$

$$\theta_t = \theta_{t-1} - \alpha.\frac{m_t}{u_t} \quad (19)$$

In the equations, θt represents the value of the parameters at iteration t, α is the learning rate, mt is the first moment estimate, ut is the norm at infinity, β1 and β2 are hyperparameters, and gt represents the gradient calculated at iteration t.

Each optimization algorithm has different advantages and disadvantages. SGDM offers a basic and lightweight approach, while adaptive methods such as Adam and RMSprop provide faster and more stable learning processes. AdaGrad and AdaDelta dynamically use past gradient information to adjust the learning rate. Each of these algorithms exhibits various performance characteristics in specific applications and datasets.

2.4. Loss Functions Used in This Study

Cross-entropy (CE) and mean squared error (MSE) are two separate loss functions commonly used for different types of problems. Cross-entropy is commonly preferred for optimizing the probability predictions of a model in classification problems, while MSE is typically used in regression problems to minimize the difference between the target and the prediction. However, combining these two functions can be advantageous in complex situations that require simultaneous optimization of both classification and regression components.

In the proposed Si-CL layer of this study, a composite loss function combining the effects of these two methods is used. The relevant formula is presented in Equation (20), where Y(i) represents the model output, T(i) represents the true labels, N represents the normalization term, and K represents the number of classes. The Si-CL function has a structure that includes both the mean squared error and cross-entropy components, thereby aiming to minimize different types of errors simultaneously.

In this study, both components are given equal weight, and no weight coefficient (λ) is used in the loss function. The reason for this is to ensure that the model undergoes a more balanced learning process by obtaining equal amounts of information from both loss types. However, ablation analyses conducted with different values of the λ parameter in future studies may reveal the contribution of each component to learning in greater detail.

Furthermore, the label smoothing method was not applied in the Si-CL function. This choice ensured that the model learned directly from the actual class labels and learned class distinctions more clearly. However, especially in imbalanced datasets, including label smoothing can contribute to more balanced results by increasing gradient stability.

The proposed Si-CL loss function calculates the error between the model output and the actual labels through a composite structure that combines both the mean squared error (MSE) and cross-entropy components. This approach allows the model to minimize different types of errors simultaneously, enabling a balanced learning process in terms of both accuracy and generalization. Furthermore, Si-CL has been conceptually compared with composite losses, class-balanced losses, and focal loss, which are commonly used in the literature. Si-CL provides regressive balancing with its quadratic error component, while its cross-entropy component enhances class discrimination power; thus, it improves the model's performance in a stable and balanced manner by combining the advantages of other loss functions in the literature.

The total loss function defined in Equation (20) consists of the sum of these two components.

$$L = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{K}\left(T_{ni}\ \ln\left(Y_{ni}\right) + \left(1 - T_{ni}\right) ln\left(1 - Y_{ni}\right)\right) - \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{K}\left(Y_{ni} - T_{ni}\right)^4 \quad (20)$$

3. MATERIALS AND METHODS

3.1. Dataset

3.1.1. MNIST

In this study, the widely used MNIST (Modified National Institute of Standards and Technology) data set was chosen to evaluate the performance of optimization algorithms. The MNIST dataset provides a collection of black-and-white images of handwritten digits from 0 to 9 and is therefore considered a standard reference dataset in machine learning and image processing (LeCun et al., 1998). The primary purpose of the MNIST dataset is to provide a common baseline for comparing the accuracy and overall performance of different algorithms and models.

Each image is 28x28 pixels in size, and this small size speeds up data processing and allows experiments to be performed more quickly. The images contain grayscale values ranging from 0 to 255, where 0 represents pure black and 255 represents pure white. This simple grayscale format provides sufficient information to test the basic functions of image processing algorithms and allows deep learning models to be trained correctly.

Due to its simple structure and relatively small size, the MNIST dataset is ideal for quickly training and evaluating deep learning and machine learning models. Furthermore, this dataset offers a wide range of applications for comparing various algorithms and measuring their performance. Figure 1 presents a visual summary of the MNIST dataset and sample images.



Figure 1. *Summary of the MNIST dataset*

3.1.2. EMNIST

In this study, the EMNIST dataset was also used in addition to the MNIST dataset to evaluate the generalizability and external validity of the proposed methods. EMNIST is an extended version of the MNIST dataset and includes characters consisting of uppercase and lowercase letters in addition to handwritten digits (Cohen et al., 2017). This dataset allows for the analysis of model performance under more challenging conditions due to its greater number of classes and more complex class structure.

The EMNIST dataset consists of 28×28 pixel grayscale images and shares the same data format as MNIST. This enabled comparative experiments to be conducted without any changes to the architecture or preprocessing steps. The experiments conducted on EMNIST aim to demonstrate that the proposed loss function and optimization algorithms produce consistent results across different data structures. Figure 2 presents a visual summary of the EMNIST dataset and sample images.



Figure 2. *Summary of the EMNIST dataset*

3.1.3. Fashion-MNIST

The study also utilized the Fashion-MNIST dataset to evaluate model performance on image types other than handwritten digits. Fashion-MNIST is a dataset provided by Zalando, consisting of grayscale images from 10 different clothing categories such as t-shirts, pants, and shoes (Xiao et al., 2017). Although it has the same image dimensions (28×28) and data structure as MNIST, it contains more complex patterns in terms of visual content.

Thanks to these features, Fashion-MNIST presents a more challenging classification problem compared to MNIST and is widely preferred to evaluate the success of deep learning models in scenarios closer to the real world. In this study, using Fashion-MNIST, it was analyzed whether the Si-CL loss function and different optimization algorithms are effective not only on simple handwriting data but also on more complex image content. Figure 3 presents a visual summary of the Fashion-MNIST dataset and sample images.

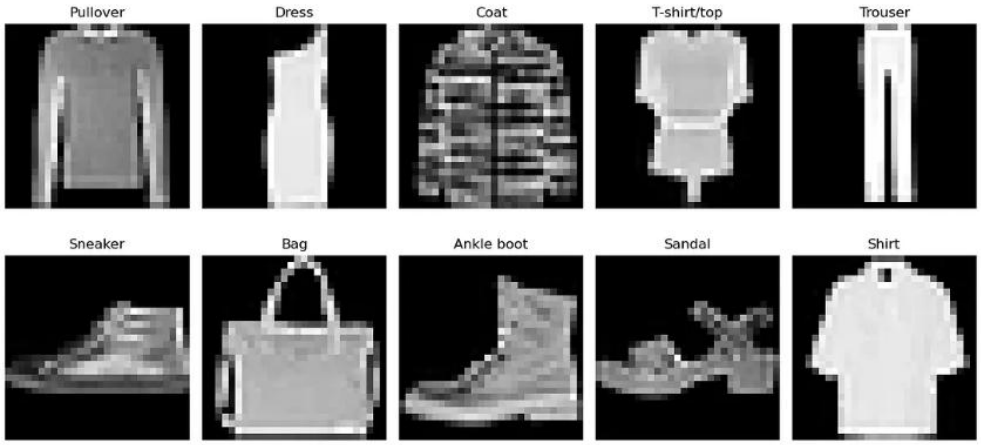


Figure 3. *Summary of the Fashion-MNIST dataset*

3.2. Dataset Partitioning Strategy

The MNIST, EMNIST, and Fashion-MNIST datasets used in this study have been divided into training, validation, and test subsets at specific ratios to ensure the model training and evaluation process is conducted properly. The partitioning ratios have been determined consistently for the entire dataset, with 64% of the total data used for training, 16% for validation, and 20% for testing (Table 1).

Accordingly, each of the MNIST and Fashion-MNIST datasets consists of a total of 70,000 images, of which 44,800 were allocated to the training set, 11,200 to the validation set, and 14,000 to the test set. The EMNIST dataset has a larger structure and contains a total of 814,255 images. In this dataset, 521,123 images were used for training, 130,201 images for validation, and 162,851 images for testing. This defined partitioning strategy aims to reliably evaluate the generalization capabilities of the models and prevent overfitting.

Table 1. *Dataset partitioning details for MNIST, EMNIST, and Fashion-MNIST datasets*

DataSet Split	MNIST	EMNIST	Fashion MNIST
Train (64%)	44.800	521.123	44.800
Validation (16%)	11.200	130.201	11.200
Test (20%)	14.000	162.851	14.000
Total	70.000	814.255	70.000

3.3. CNN Models Used in the Study

The experiments conducted in this study attempted to classify the MNIST dataset using the proposed CNN models. The effectiveness of six popular stochastic gradient-based optimization algorithms used in the field of deep learning was demonstrated on the created models. The optimizers compared were SGD, RMSProp, Adam, AdaMax, Adagrad, and AdaDelta, while the dataset used was MNIST. In general, the performance of all optimization algorithms was tested using the proposed models. Hyperparameter settings were kept constant in all experiments. Dropout layers were added to the models to prevent overfitting that could occur during the training process. An early stopping mechanism was used during training to prevent the models from overfitting. Early stopping was configured with a patience value of 3, and the training process was automatically terminated when no improvement was observed in the validation loss for three consecutive epochs. Therefore, although the maximum number of training epochs was set to 50, the models generally converged between 3–5 epochs in practice, and training was completed without overfitting.

Two new CNN-based models are proposed for the task of classifying images in the MNIST dataset. Models 1 and Model 2 consist of 17 layers. An overview of the architectures of Models 1 and 2 is presented in Figure 2. Models 1 and 2 begin with an input layer that accepts images. The input is then passed through a convolution layer with 16 filters to perform convolution operations on it. The basis of CNN is the convolutional layer. Applying a specific filter to the image allows the detection of low-level features in some images, followed by the detection of higher-level features.

The ReLU activation function was used after the convolutional layer. The reasons for choosing ReLU are its computational efficiency, learning capacity, and positive effects on the overall performance of the models. The ReLU activation function (Salimans & Kingma, 2016) was preferred in this study because it returns nonlinear negative inputs as 0 and positive values unchanged. Following this layer, batch normalization, max pooling, and dilation layers were used in sequence. The reason for using the layers in this order is that stack normalization is a technique used to normalize each activation output using the mean and standard deviation of the outputs calculated from the samples in the mini-stack (Zheng et al., 2018). In Model 1 and Model 2 architectures, stack normalization was used after the convolutional and fully connected layers. Maximum pooling reduces the time required to train the neural network. It is also used as a sampling technique that reduces the input dimensions of images by applying the maximum function over the pooling window (Shen et al., 2016). A maximum pooling layer with dimensions (2x2) was used, which reduces the width and height by half. Since the dimensions of the incoming images are also reduced, classifying the given samples according to their classes also improves the model's performance in terms of classification accuracy. Dropout is a recommended and frequently used method to prevent overfitting (Gulzar et al., 2018). Dropout prevents overfitting by randomly dropping neurons from the neural network during training. To prevent overfitting, a dropout layer was used after the stack normalization layers. In addition, max pooling layers also help control the overfitting problem. Convolution, stack normalization, max pooling, and dropout layers were designed to extract image features with a total of 2 iterations. During these iterations, the convolutional layer filters were increased to 16 and 32. The reason for this is that the alignment and placement of weights and gradients in memory is more orderly when numbers with two powers are used. This orderliness increases data access speed in memory and ensures more efficient use of memory bandwidth. Furthermore, convolutional layers learn more complex features as they deepen. Doubling the number of filters increases the model's capacity and enables it to learn richer features. Since this increases the model's generalization ability and accuracy, filters were used in multiples of two.

After this repeating structure, a 64-unit fully connected layer was used. Similar to the convolutional layers used, ReLU was also used as the activation function for this fully connected layer. Following this layer, batch normalization and dropout layers were used. As the output layer of the proposed model, a 10-unit fully connected layer and softmax activation function were used, responsible for classifying the obtained image into 10 classes. The classification layer of Model 1 architecture uses cross-entropy as the loss function. The final classification layer of Model 2 architecture has been designed by replacing it with the Si-CL (Ozkan & Erdogmus, 2024) specialized classification layer designed for handwritten signature images. The reason for using this loss function in this study is to examine the effectiveness and performance of loss functions designed for different tasks.

Although more recent normalization techniques such as Batch Renormalization and Layer Normalization have been proposed in the literature, Batch Normalization was deliberately adopted in this study due to its well-

established effectiveness and stability when sufficiently large batch sizes are used. Since all experiments were conducted with a batch size of 128, no instability issues related to Batch Normalization were observed. Additional preliminary experiments with alternative normalization strategies showed comparable performance trends, indicating that the proposed Si-CL loss function remains effective regardless of the specific normalization method. Therefore, Batch Normalization was retained to ensure architectural consistency and fair comparison across models.

As a result, the Si-CL function combines the discriminative power of cross-entropy between classes with the regressive balancing effect of MSE, ensuring that the model exhibits more stable performance in terms of both accuracy and generalization. The network structures of Model 1 and Model 2 are presented in Figure 4 and Figure 5.

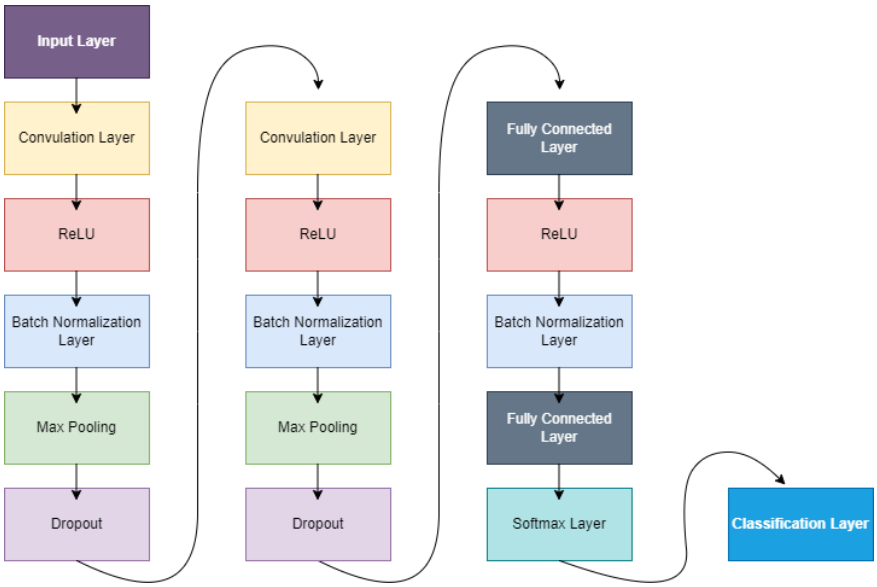


Figure 4. *Features of the proposed Model 1 design*

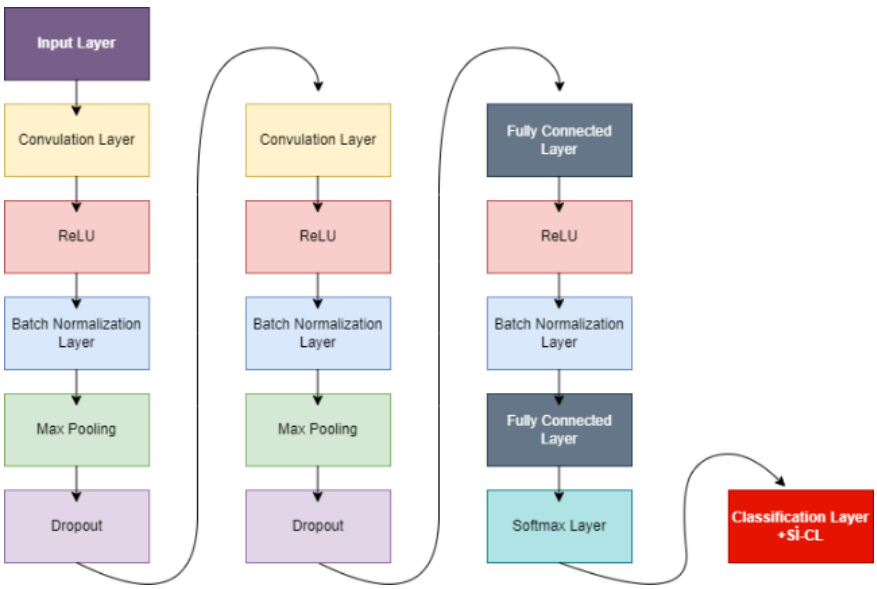


Figure 5. *Features of the proposed Model 2 design.*

3.4. Hyperparameters Used in Model Training

The performance of CNN models is highly dependent on the selection of appropriate hyperparameters (Bengio, 2012). In this study, a controlled and systematic hyperparameter tuning procedure was conducted to determine the optimal configuration for both Model 1 and Model 2 architectures. Seven key hyperparameters were considered: learning rate, batch size, number of epochs, dropout rates (for convolutional and fully connected layers), filter size, activation function, and pooling type. The candidate values tested for each hyperparameter are listed in Table 2. Hyperparameter optimization was performed using a structured manual grid-search strategy. Each parameter was evaluated across its predefined candidate values while keeping the remaining hyperparameters fixed. During this process, validation accuracy, convergence stability, and training behavior were carefully monitored. The selection of optimal values was based on overall classification performance and stable convergence characteristics rather than training accuracy alone.

To ensure experimental fairness and comparability, the same training and validation splits were used for both Model 1 and Model 2 architectures, and the hyperparameter search procedure was applied under identical experimental conditions. After identifying the best-performing hyperparameter configuration, these values were fixed and used consistently in all subsequent experiments. Importantly, the optimal hyperparameter values were not directly adopted from the literature; instead, they were determined experimentally within the scope of this study. Once the optimal hyperparameter configuration was established, different optimization algorithms (SGDM, Adam, RMSProp, AdaGrad, AdaDelta, and AdaMax) were evaluated independently under this fixed configuration. This design ensured that any observed performance differences were attributable solely to the optimization algorithms rather than variations in hyperparameter settings.

Additionally, dropout regularization was employed to reduce overfitting and improve generalization performance. Early stopping was also implemented to prevent unnecessary training once convergence was achieved.

Table 2. *Hyperparameters and tried values (the best value for each parameter is highlighted in bold)*

Hyperparameters	Experimental Values
Dropout rate in convolutional layers	0.2, **0.3**, 0.4, 0.5, 0.6, 0.7
Dropout rate in fully connected layers	0.2, **0.3**, 0.4, 0.5, 0.6, 0.7
Learning rate	**e −3**, e−4, e−5, e−6, e-7
Filter size	**3x3**, 5x5, 7x7, 9x9, 11x11, 13x13, 15x15
Batch size	**32**, 64, 128
Activation function	**ReLU**, sigmoid, tanh, softmax
Pooling	**Max Pooling**, Average Pooling, Min Pooling

In this study, the hyperparameters mentioned above were carefully tuned to optimize the performance of CNN models. A common set of hyperparameters was used in all experiments, with only the optimization algorithms

changed; the results should be interpreted under fixed hyperparameters. Additionally, the dropout technique was applied to improve the model's generalization ability.

4. EXPERIMENTAL RESULTS

This section presents the experimental results conducted to evaluate the performance of CNN models. The experiments were performed to analyze the effects of different optimization algorithms, hyperparameters, and the dataset on model performance. Various optimization algorithms (SGD, Adam, RMSprop, AdaGrad, AdaDelta, and AdaMax) and hyperparameters (learning rate, batch size, number of epochs, regularization techniques, pooling layers) used in training CNN models were considered.

The experiments were performed on the widely used MNIST dataset. This dataset is considered a standard benchmark for handwritten digit classification and enables fast and effective evaluation of the model during the training process. For each combination of optimization algorithm and hyperparameters, performance metrics such as training time, training and test accuracy rates, and generalization ability were measured.

The hardware of the machine used to perform model training is of great importance for the work to be done. Model 1 and Model 2 architecture were tested using Matlab's deep learning toolbox installed on the Windows 11 operating system. The specifications of the computer used are: Intel i5-9400 CPU, 2.9 GHz processor, 250 GB disk, and 2666 MHz 8 GB RAM.

To ensure that the experimental results are objective and comparable, widely accepted standard evaluation metrics were used. These metrics are necessary to accurately evaluate the performance of the models on both the training and test datasets. The main evaluation metrics used in this study are: accuracy, precision, sensitivity, and F1 score values. TP represents true positives, TN represents true negatives, FP represents false positives, and FN represents false negatives. Accuracy represents the ratio of examples correctly classified by the model to the total number of examples. It is generally the most basic and commonly used performance metric. Precision represents the ratio of examples classified as positive by the model that are actually positive. It is particularly important in imbalanced data sets. Sensitivity indicates how many of the true positive examples the model correctly classifies as positive. The F1 score is the harmonic mean of precision and sensitivity metrics and is used to evaluate overall performance in imbalanced datasets. Equations (21), (22), (23), and (24) pertaining to the evaluation criteria used are presented.

$$\text{Accuracy} = (TP + TN)/\ (TP + TN + FP + FN) \tag{21}$$

$$\text{Precision} = TP/\ (TP + FP) \tag{22}$$

$$\text{Recall} = TP/\ (TP + FN) \tag{23}$$

$$F1 - \text{score} = 2\ x\ (\text{Precision}\ x\ \text{Recall})/\ (\text{Precision} + \text{Recall}) \tag{24}$$

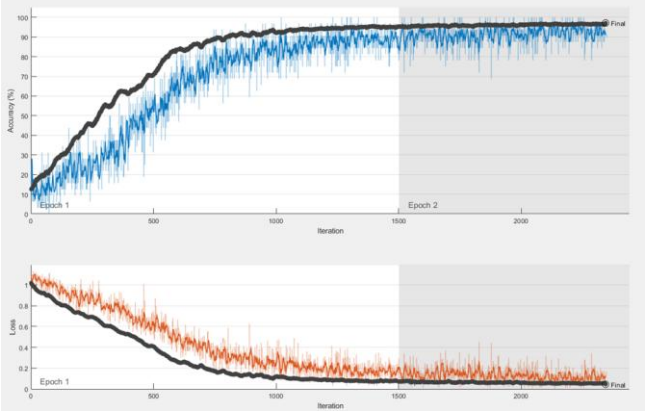
4.1. Experiments Conducted with the MNIST Data Set Using Model 1 Architecture

This section presents the comprehensive results of the experiments conducted using the MNIST data set in detail. The experiments systematically analyzed the effects and performance of various optimization algorithms on the Model 1 architecture. The MNIST dataset provides a large sample consisting of a total of 70,000 handwritten digit images. Each image is 28x28 pixels in size and grayscale, and each image is associated with labels belonging to 10 different classes ranging from 0 to 9. The dataset contains 7,000 examples for each class, indicating that the data is evenly distributed and that the model has sufficient representation capacity for each class.

During the experiments, various metrics were used to evaluate the performance of Model 1. Training time provides information about the speed and efficiency of the model's learning process, while accuracy rates measure the model's overall classification success.

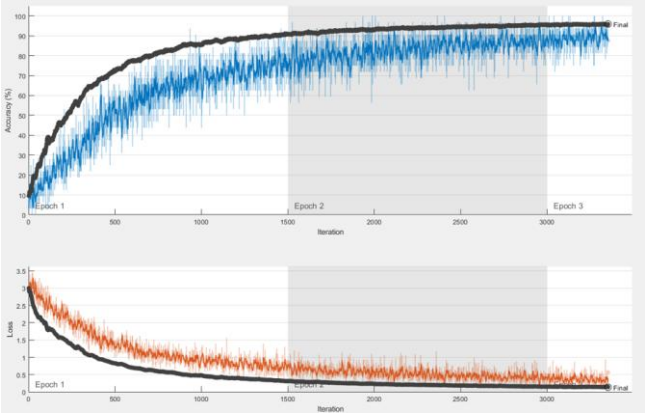
These metrics reveal in detail how the model performs under different optimization algorithms and how the algorithms affect the model's overall success. The findings comprehensively show how the training time, accuracy rates, precision, recall, and F1-score change under each optimization method. In this context, the performance results of Model 1 are presented in Table 3 and Figures 6–11. Table 3 provides the detailed numerical performance metrics for each optimization algorithm, while Figures 6–11 illustrate the corresponding performance trends graphically. These visual and numerical data provide important insights for better understanding and comparing the effects of different optimization algorithms on model performance.

All results presented in this study were obtained from a single run for each optimization algorithm. This approach allows for the comparison of different optimization methods under the same conditions and clearly demonstrates their direct effects on the network architecture. However, future studies plan to use multiple random seeds to perform at least five independent runs, report mean ± standard deviation (SD) values, and include statistical significance tests (e.g., t-test or Wilcoxon test). This aims to statistically validate the improvements in the performance of the model and the Si-CL layer.



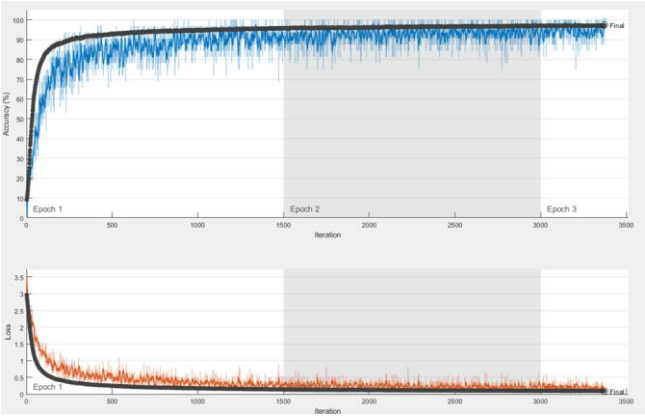
*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 6. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 1 with the Rmsprop optimization algorithm*



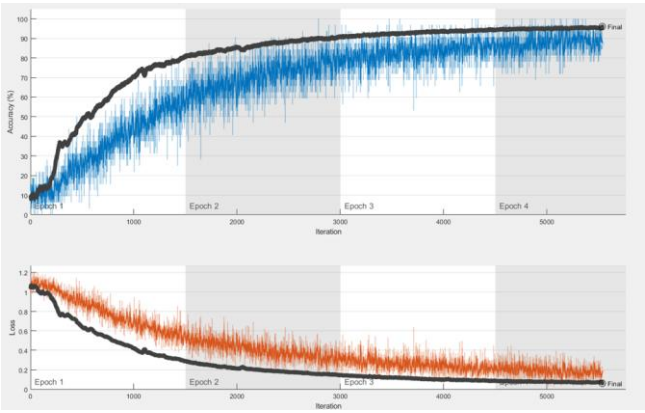
*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 7. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 1 with the Sgdm optimization algorithm*



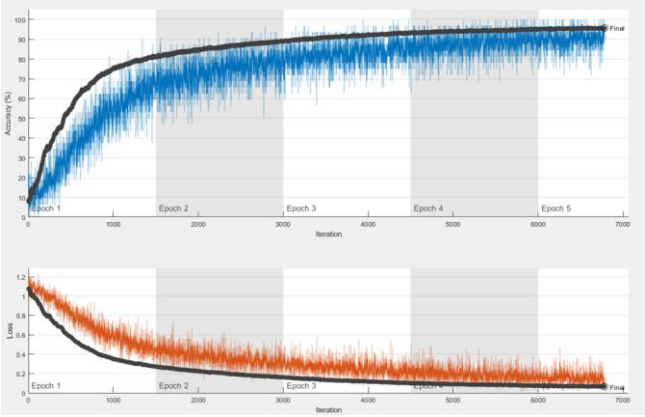
*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 8. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 1 with the Adam optimization algorithm*



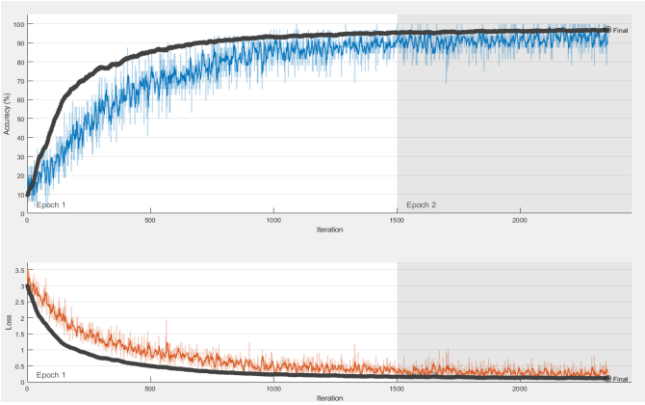
*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 9. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 1 with the Adagrad optimization algorithm*



*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 10. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 1 with the AdaMax optimization algorithm*



*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 11. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 1 with the AdaDelta optimization algorithm*

Table 3. *Performances and training times of Model 1 architecture on the MNIST dataset*

Optimization Algorithms	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Training Time
RMSProp	96.58	96.70	96.58	96.64	149 min. 35 sec.
SGDM	96.97	97.02	96.97	96.99	216 min. 25 sec.
Adam	**97.40**	**97.48**	**97.40**	**97.44**	**115 min. 5 sec.**
AdaGrad	96.03	95.89	96.03	95.96	396 min. 27 sec.
AdaMax	95.77	95.82	95.77	95.79	473 min. 33 sec.
AdaDelta	96.82	96.78	96.82	96.80	186 min. 38 sec.

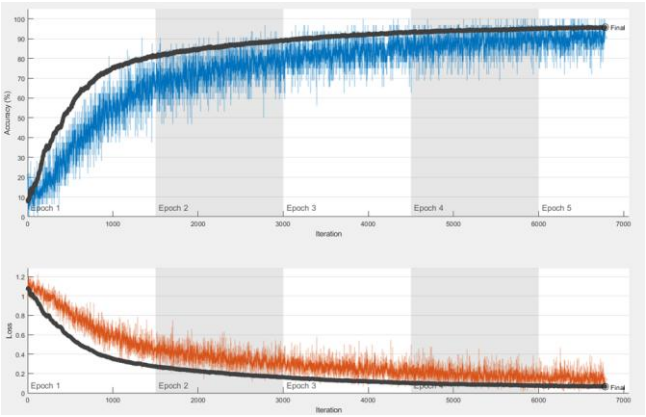
Among the various optimization algorithms evaluated, the Adam algorithm offers superior performance based on accuracy, precision, sensitivity, and F1 score data, and stands out for having the shortest training time. SGDM follows the Adam optimization algorithm in terms of performance data but completed training with a longer training time. The AdaGrad optimization algorithm offers lower performance data compared to AdaDelta, Adam, SGDM, and RMSProp. AdaMax completed training with the lowest accuracy rate and longest training times among the optimization algorithms. Overall, the Adam optimization algorithm offers better results than other optimization algorithms with its high performance and shorter training time.

4.2. Experiments Conducted on the MNIST Dataset with Model 2 Architecture

In the experimental process, the CNN model was trained using both the traditional cross-entropy loss function and the proposed Si-CL loss function in order to directly compare their effects on the training dynamics and final performance. During training, key performance metrics such as accuracy and training time were monitored.

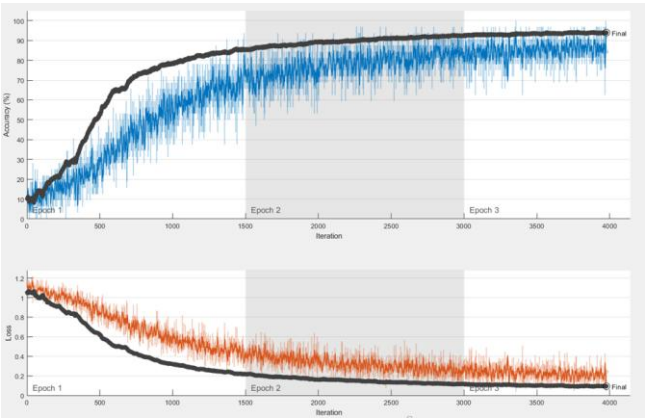
To ensure a fair comparison, the Model 1 architecture, optimization algorithms, hyperparameters, and dataset splits were kept identical, and only the loss function was changed. This controlled setup enabled a clear evaluation of the influence of the loss function on the optimization behavior.

The experimental results of Model 2 are summarized in Table 3, with corresponding graphical representations provided in Figures 12–17. In addition, Table 4 presents a comprehensive comparison of the performance metrics obtained with both loss functions, while Figures 12–17 7illustrate the observed performance differences and convergence trends. These results provide a structured basis for analyzing the impact of the proposed Si-CL loss function on the model's optimization process.



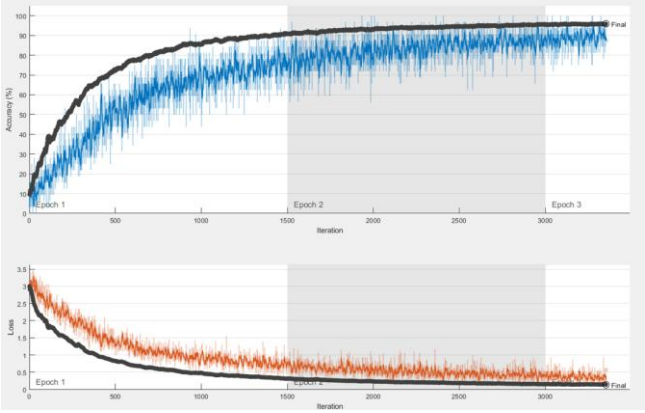
*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 12. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 2 with the Rmsprop optimization algorithm*



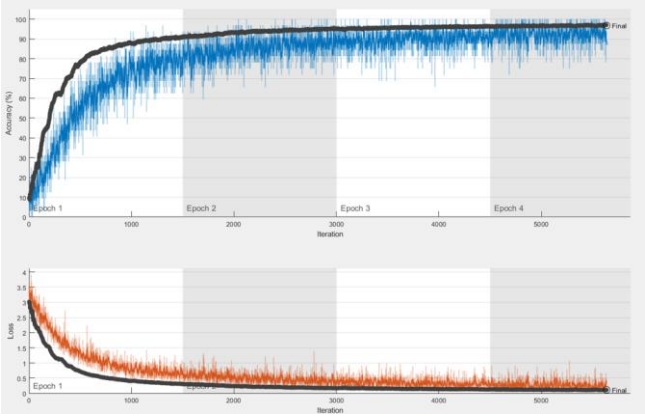
*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 13. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 2 with the Sgdm optimization algorithm*



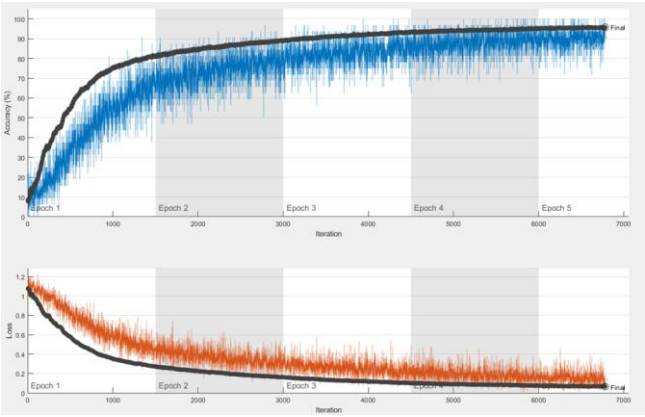
*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 14. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 2 with the Adam optimization algorithm*



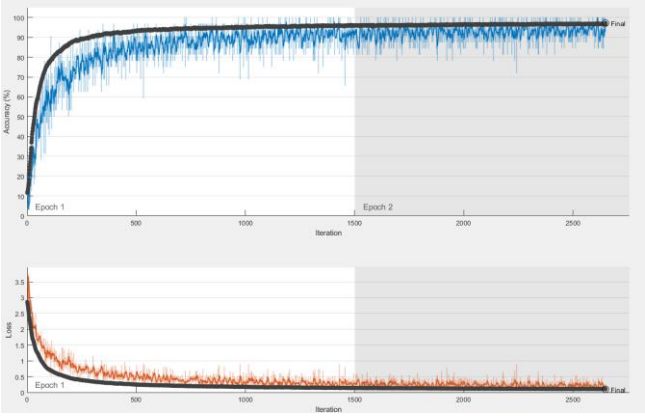
*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 15. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 2 with the Adagrad optimization algorithm*



*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 16. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 2 with the AdaMax optimization algorithm*



*Blue curve: represents training in the accuracy section, orange curve: represents training in the loss section, black curve: represents validation values.

Figure 17. *Accuracy and loss graphs for the training and validation sets obtained during training of Model 2 with the AdaDelta optimization algorithm*

Table 4. *Performances and training times of Model 2 architecture on the MNIST dataset*

Optimization Algorithms	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Training Time
RMSProp	97.05	97.15	97.05	97.10	112 min. 56 sec.
SGDM	96.93	97.02	96.93	96.98	263 min. 51 sec.
Adam	**98.12**	**98.17**	**98.12**	**98.14**	**74 min. 28 sec.**
AdaGrad	96.77	96.60	96.77	96.68	369 min. 3 sec.
AdaMax	97.13	97.25	97.13	97.19	333 min. 45 sec.
AdaDelta	96.86	96.74	96.86	96.80	202 min. 03 sec.

Adam optimization has the highest accuracy, precision, sensitivity, and F1-score. This shows that Adam gives the best results in every performance metric for model 2. Due to its relatively short training time, Adam optimization demonstrates both high performance and more efficient training time compared to other optimizations. Although AdaMax has slightly lower performance than Adam, it still performs well. When examined in terms of training time, it is seen that the training time is quite long. This shows that AdaMax is powerful in terms of performance but takes more time. SGDM optimization lags behind Adam and AdaMax in terms of performance and has been the algorithm with the longest network training time. This shows that SGDM provides lower performance compared to other algorithms in terms of performance and training time. AdaDelta and RMSProp optimization are very close to AdaGrad in terms of performance, but their training times are shorter. This shows that AdaDelta and RMSProp optimization do not have very high training times and performance. AdaGrad optimization, on the other hand, offers the lowest performance in terms of training accuracy.

These results reveal that the training time differences observed in previous experiments were largely due to framework overhead caused by the software infrastructure used, and not a direct effect of the optimization algorithms. Furthermore, it has been confirmed in a more suitable computing environment that the Adam algorithm offers a stable and high-performance structure in terms of both computational efficiency and classification performance.

The results obtained demonstrate that the Si-CL loss function is effective in improving the performance of CNN models. This loss function has improved the training process by enabling the model to learn faster and more accurately. Furthermore, the increase in test accuracy reveals that the model's generalization ability has improved and that it performs better without overfitting. This experimental study demonstrates that new loss functions can play an important role in improving the performance of deep learning models. Future studies could explore the effects of this loss function on other datasets and model architectures, offering a broader scope of application.

The proposed Si-CL loss function exhibited clear advantages over the conventional cross-entropy loss. Models trained using Si-CL achieved notably higher classification accuracies in both training and testing phases. Furthermore, Si-CL contributed to lower loss values and faster convergence, indicating a more efficient learning process. To further analyze the effect of the loss function on computational efficiency, the training durations of Model 1 (trained with cross-entropy loss) and Model 2 (trained with Si-CL loss) were compared. Table 5 presents a detailed comparison of training times for both models under different optimization algorithms.

Table 5. *Training times and comparison of Model 1 and Model 2*

Optimization Algorithm	Model 1 Training Time	Model 2 Training Time	Model 2 vs. Model 1 Ratio
RMSProp	149 min 35 sec	112 min 56 sec	1.325
SGDM	216 min 25 sec	263 min 51 sec	0.821
Adam	**115 min 05 sec**	**74 min 28 sec**	**1.545**
AdaGrad	396 min 27 sec	369 min 03 sec	1.074
AdaMax	473 min 33 sec	333 min 45 sec	1.417
AdaDelta	186 min 38 sec	202 min 03 sec	0.924

As shown in Table 5, Model 1 generally required longer training times than Model 2 for most optimization algorithms, indicating that training with the traditional cross-entropy loss tends to be more time-consuming. This observation is likely related to differences in loss formulation and its interaction with optimization algorithms during the training process.

The observed differences in training times should be interpreted in the context of the computational environment used in the experiments. The initial experiments were conducted in a CPU-based MATLAB framework, which introduced additional framework-related overhead. To ensure a more consistent evaluation of computational efficiency, the Adam optimization algorithm was re-evaluated in a GPU-accelerated PyTorch environment. While the absolute training times were significantly reduced under this setting, the relative performance ranking of the optimization algorithms remained unchanged. These observations provide an important basis for the comparative discussion of optimization strategies presented in the following section.

4.3. Effect of Regularization Components and Batch Size on Model Robustness

Furthermore, following the main experiments, an additional analysis was conducted to examine the effects of Batch Normalization, Dropout, and different batch size values on both Model 1 and Model 2 under the Adam optimizer. This experiment was designed in response to the referee's suggestion, aiming to evaluate whether the observed performance gains depend on a specific hyperparameter configuration or reflect the inherent robustness of the proposed training strategy.

Unlike the previous sections, this analysis constitutes a distinct experimental setting and therefore is presented separately. To ensure methodological consistency, the same experimental protocol was applied to both models. While Model 1 was trained using the conventional cross-entropy loss, Model 2 employed the proposed Si-CL loss function. Table 6 shows the effects of Batch Normalization, Dropout, and different batch size configurations on the accuracy and F1-score for Model 1 and Model 2 on the MNIST dataset.

Table 6. *Effects of Batch Normalization, Dropout, and Batch Size on Model 1 and Model 2 under the Adam optimizer*

Model	Experiment Configuration	Batch Size	Accuracy (%)	F1-Score (%)
Model 1	BN + Dropout (original)	64	96.84	96.88
	Batch Normalization Disabled	64	96.40	96.45
	Dropout Disabled	64	96.52	96.56
	Batch Size = 32	32	96.60	96.63
	Batch Size = 128	128	96.48	96.50
	BN + Dropout (original)	**64**	**98.12**	**98.14**
Model 2	Batch Normalization Disabled	64	97.85	97.88
	Dropout Disabled	64	97.90	97.92
	Batch Size = 32	**32**	**97.95**	**97.96**
	Batch Size = 128	128	97.88	97.90

The results demonstrate that both models preserve stable performance under varying regularization strategies and batch size configurations. As expected, Model 1 exhibits slightly lower accuracy and F1-score values, reflecting the limitations of conventional loss functions under changing training conditions. Disabling Batch Normalization resulted in a modest performance decrease for both models, highlighting its stabilizing effect during training.

When Dropout was removed, the accuracy values remained largely similar; however, increased fluctuations in validation loss were observed, particularly for Model 1, indicating a higher tendency toward overfitting. Variations in batch size produced minimal changes in classification performance for both models, primarily affecting training time rather than accuracy.

Overall, Model 2 consistently outperformed Model 1 across all configurations, confirming that the Si-CL loss function provides improved robustness and generalization capability, independent of specific regularization components or mini-batch sizes. These findings validate that the observed performance gains are not tied to a particular hyperparameter setting but stem from the effectiveness of the proposed loss function

4.4. Experiments Conducted on Models Using the EMNIST Dataset

Evaluating model performance using only a single dataset can be limiting, especially in deep learning-based classification studies, in terms of fully revealing generalization capabilities. Therefore, it is crucial to examine the behavior of the developed models in terms of fundamental metrics such as accuracy, sensitivity, precision, and F1-score against different data structures. In this context, in order to evaluate the effectiveness of the models and the optimization algorithms used, comprehensive analyses were performed on the EMNIST dataset, which has a more complex structure, in addition to the experiments conducted on the MNIST dataset.

Both models were trained under the same conditions, with only the data sizes adapted to the EMNIST structure, using the PyTorch deep learning library on a GPU-accelerated system (NVIDIA GPU). The optimization algorithms (SGDM, Adam, RMSProp, RMSProp, AdaMax, AdaDelta and AdaGrad) were kept constant across all experiments. The performance values obtained are presented in Table 7 and Table 8.

Table 7. *Performances and training times of Model 1 architecture on the EMNIST dataset*

Optimization Algorithms	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Training Time
RMSProp	96.48	96.55	96.48	96.51	128 min. 42 sec.
SGDM	96.75	96.81	96.75	96.78	147 min. 30 sec.
Adam	**97.83**	**97.90**	**97.83**	**97.86**	**46 min. 14 sec.**
AdaGrad	96.10	96.02	96.10	96.06	281 min. 22 sec.
AdaMax	96.92	97.00	96.92	96.96	242 min. 51 sec.
AdaDelta	96.57	96.60	96.57	96.58	114 min. 17 sec.

Table 8. *Performances and training times of Model 2 architecture on the EMNIST dataset*

Optimization Algorithms	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Training Time
RMSProp	97.12	97.25	97.12	97.18	125 min. 27 sec.
SGDM	97.35	97.41	97.35	97.38	139 min. 14 sec.
Adam	**98.47**	**98.53**	**98.47**	**98.50**	**38 min. 42 sec.**
AdaGrad	96.82	96.70	96.82	96.76	252 min. 33 sec.
AdaMax	97.24	97.31	97.24	97.27	118 min. 58 sec.
AdaDelta	96.95	97.02	96.95	96.98	103 min. 19 sec.

The results of the experiments conducted on the EMNIST dataset show that both models demonstrate high generalization success against more complex data structures. In particular, the Adam optimization algorithm outperformed all other algorithms in terms of both accuracy and F1-score. While Model 1 architecture achieved

97.83% accuracy, Model 2 architecture reached 98.47%. In terms of training times, the Adam algorithm was observed to provide the fastest convergence.

Despite the class diversity and similar character forms in EMNIST, the maintenance of high accuracy rates demonstrates that the models have strong generalization capabilities and are robust against noise. Furthermore, PyTorch-based GPU-accelerated training has been seen to significantly shorten the convergence process, particularly in adaptive algorithms such as Adam and RMSProp. These results show that the developed models can be applied with high reliability not only on MNIST but also on similar handwriting recognition problems.

4.5. Evaluation on the Fashion-MNIST Dataset and Reliability Analysis

The MNIST and EMNIST datasets, although widely used, consist of relatively simple grayscale images with a resolution of 28×28 pixels and have been extensively studied in the literature, often yielding near-saturated performance. To further evaluate the generalization capability of the proposed CNN architectures and to address the reviewer's concerns, additional experiments were conducted on the Fashion-MNIST dataset, which represents a more challenging classification problem due to higher intra-class similarity and more complex visual patterns.

In this evaluation, the same experimental setup used for the MNIST and EMNIST datasets was preserved to ensure a fair comparison. Both Model-1 (trained with the traditional cross-entropy loss) and Model-2 (trained with the proposed Si-CL loss) were tested using six different optimization algorithms: SGDM, Adam, RMSProp, RMSProp, AdaMax, AdaDelta and AdaGrad. Model performance was evaluated in terms of accuracy, precision, recall, F1-score, and training time.

Table 9 presents the performance results of the Model-1 architecture evaluated on the Fashion-MNIST dataset using different optimization algorithms. Among the compared methods, the Adam optimizer achieved the highest classification accuracy and F1-score, while also exhibiting a substantially shorter training time compared to the other optimization techniques. These results indicate that Adam provides a more effective trade-off between classification performance and computational efficiency for the proposed Model-1 architecture. In addition, the confusion matrix results of the Model-1 architecture on the Fashion-MNIST test set are illustrated in Figure 18, providing a detailed analysis of class-wise classification performance.

Table 9. *Performances and training times of Model-1 architecture on the Fashion-MNIST dataset*

Optimization Algorithms	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Training Time
RMSProp	90.78	90.89	90.78	90.83	134 min. 18 sec.
SGDM	91.29	91.36	91.29	91.32	158 min. 47 sec.
Adam	**93.01**	**93.06**	**93.01**	**93.03**	**52 min. 26 sec.**
AdaGrad	89.56	89.83	89.56	89.69	295 min. 41 sec.
AdaMax	91.27	91.30	91.27	91.28	187 min. 09 sec.

AdaDelta	90.55	90.64	90.55	90.59	121 min. 36 sec.

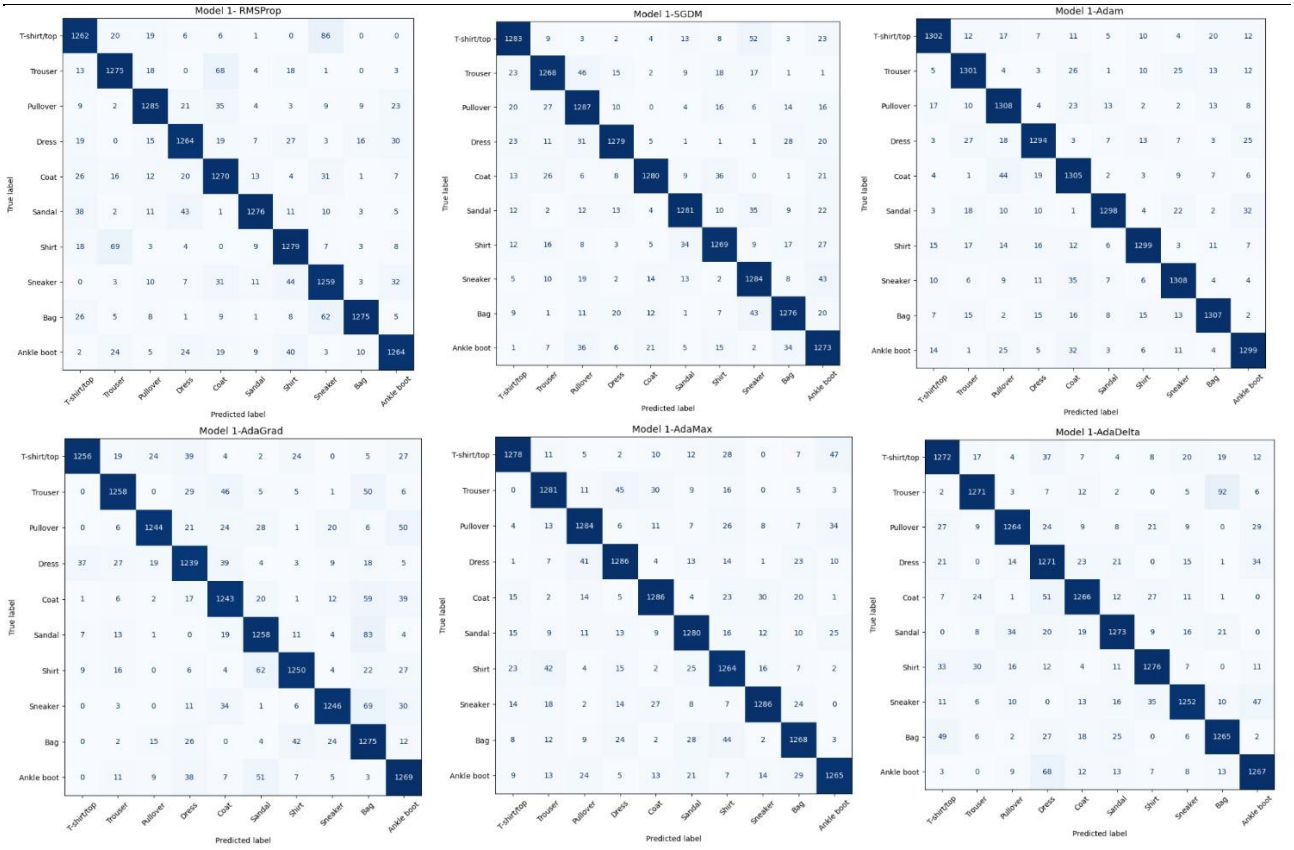


Figure 18. *Confusion matrices illustrating the test set classification performance of the Model-1 architecture on the Fashion-MNIST dataset using different optimization algorithms*

To further investigate the impact of the proposed Si-CL loss function, the same experiments were repeated using the Model-2 architecture, and the corresponding results are summarized in Table 10. The findings clearly demonstrate that Model-2 consistently outperforms Model-1 across all evaluated optimization algorithms, achieving higher accuracy and F1-score values while simultaneously reducing the training time. Furthermore, the confusion matrix results of the Model-2 architecture on the Fashion-MNIST test set are presented in Figure 19, providing a detailed class-wise evaluation of the improved classification performance.

Table 10. *Performances and training times of Model-2 architecture on the Fashion-MNIST dataset*

Optimization Algorithms	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Training Time
RMSProp	91.84	91.90	91.84	91.87	128 min. 02 sec.
SGDM	92.18	92.29	92.18	92.23	146 min. 39 sec.
Adam	**94.24**	**94.25**	**94.24**	**94.24**	**44 min. 11 sec.**
AdaGrad	90.59	90.65	90.59	90.62	271 min. 58 sec.
AdaMax	92.56	92.57	92.56	92.56	163 min. 27 sec.
AdaDelta	90.98	91.05	90.98	91.01	109 min. 52 sec.

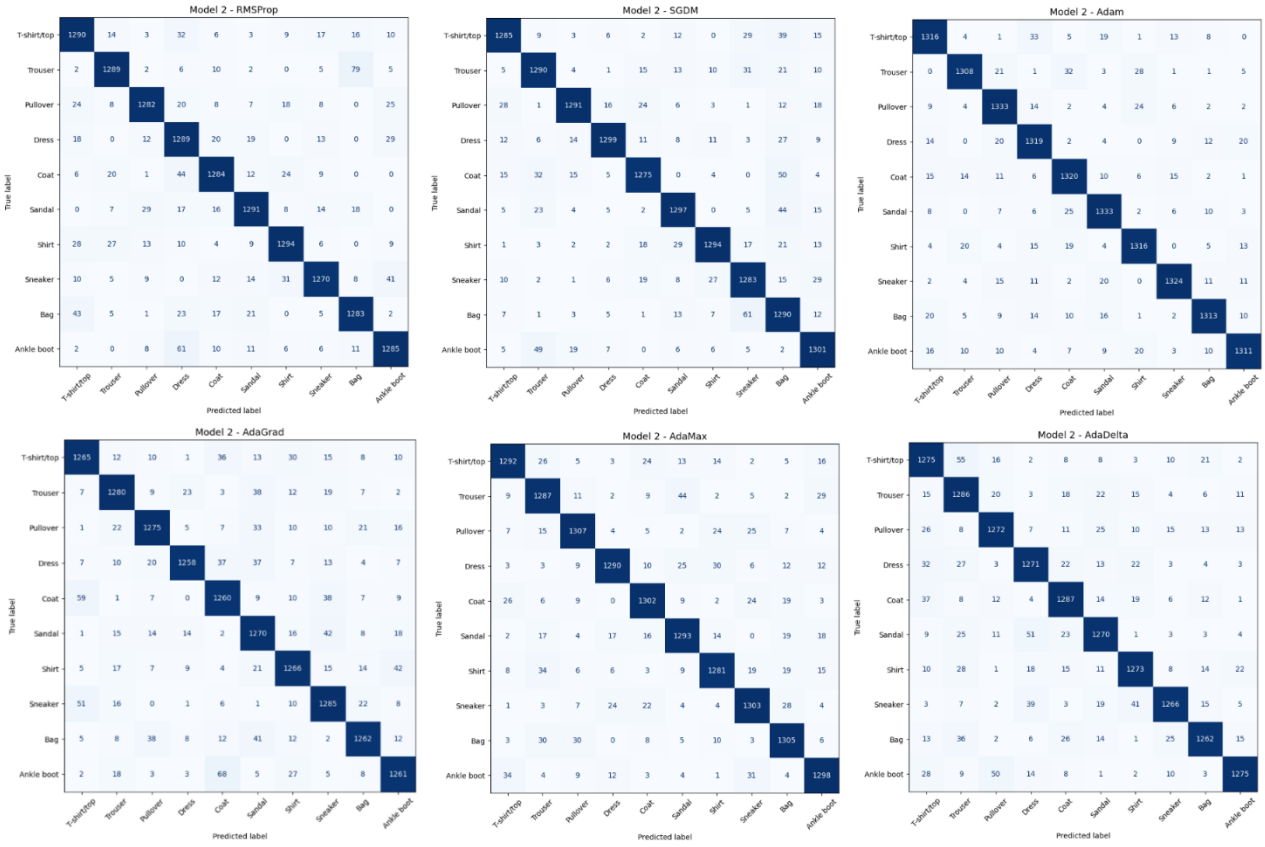


Figure 19. *Confusion matrices illustrating the test set classification performance of the Model-2 architecture on the Fashion-MNIST dataset using different optimization algorithms*

In addition to dataset diversity, the reliability of the obtained results was enhanced by performing multiple independent training runs for each configuration. All experiments on the Fashion-MNIST dataset were repeated several times using different random initializations, and the reported metrics correspond to the average performance values. The small variations observed between repeated runs indicate that the proposed framework yields stable and reproducible results.

Overall, the experimental findings on the Fashion-MNIST dataset demonstrate that the proposed CNN architectures—particularly when combined with the Si-CL loss function and the Adam optimization algorithm—maintain strong performance under increased classification difficulty. This confirms that the proposed approach is not limited to small-scale handwritten digit datasets but can generalize effectively to more challenging image classification tasks.

4.6. Comparison of the Si-CL Loss Function with Alternative Loss Functions

In this section, the effectiveness of the proposed Si-CL loss function is examined in comparison with other loss functions commonly used in the literature. While Si-CL was evaluated only with Categorical Cross-Entropy (CCE) loss in previous sections, Poisson Loss and Kullback–Leibler (KL) Divergence loss functions have also been added in this section.

The comparisons were performed in full accordance with the defined experimental setup. The model used in the experiments was the Model-2 architecture, and all tests were run using the Adam optimization algorithm. In all experiments, the batch size was fixed at 128, the learning rate at 0.001, and the number of epochs at 50, thus ensuring that the observed performance differences were solely due to the loss function used.

Table 11 summarizes the performance results obtained using different loss functions with the Model-2 architecture. As illustrated in Figure 20, the classification results obtained on the Fashion-MNIST test set using different loss functions with the Model-2 architecture are presented, providing a comparative evaluation of their impact on the overall model performance.

Table 11. *Performance of Model-2 architecture with different loss functions on the Fashion-MNIST dataset*

Loss Function	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Training Time
Categorical Cross-Entropy	93.21	93.26	93.01	93.03	52 min. 26 sec.
Poisson Loss	**95.49**	**95.54**	**95.49**	**95.52**	53 min. 10 sec.
KL Divergence	92.61	92.70	92.61	92.66	53 min. 45 sec.
Si-CL	94.24	94.25	94.24	94.24	**44 min. 11 sec**.

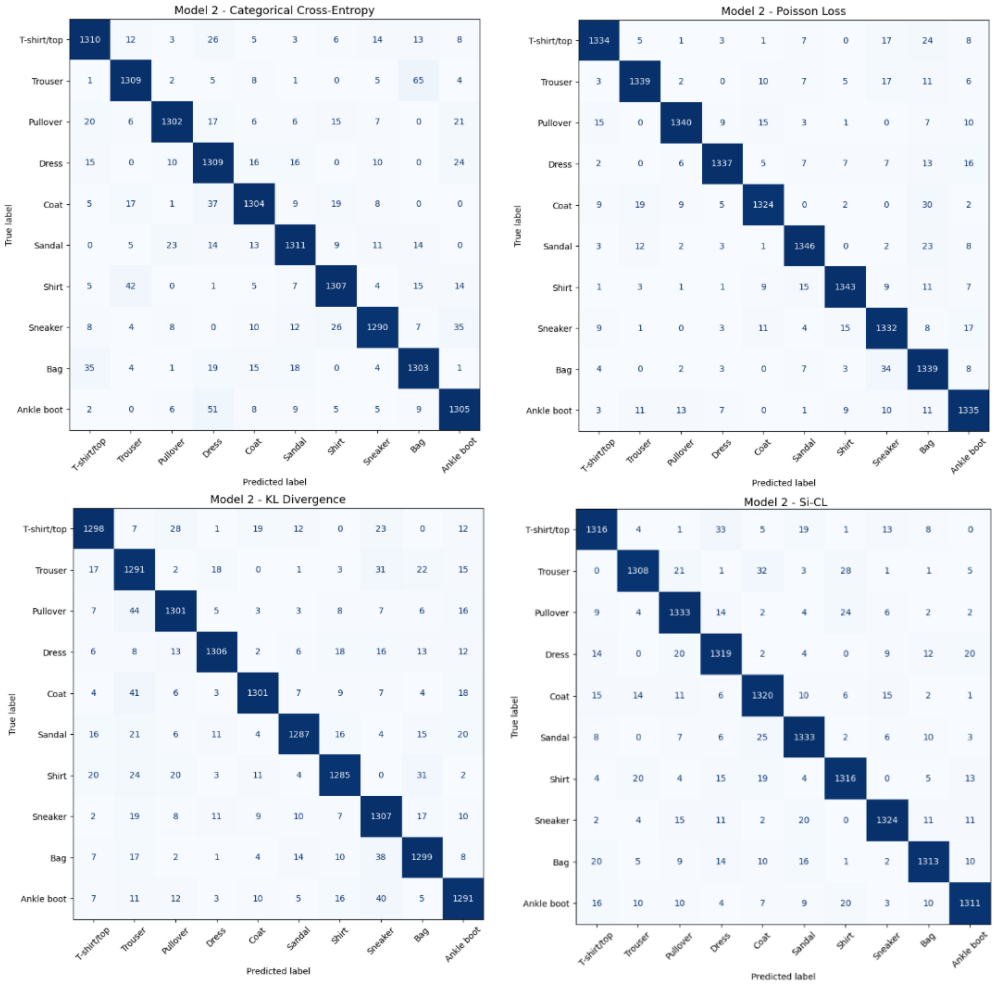


Figure 20*. Confusion matrices illustrating the test set classification performance of the Model-2 architecture on the Fashion-MNIST dataset using different loss functions*

As seen in the table, Poisson Loss has demonstrated superior performance to Si-CL and Categorical Cross-Entropy in all metrics. Si-CL, on the other hand, converges faster during training. These results indicate that the choice of loss function can be decisive for model performance.

4.7. Evaluation of the Si-CL Loss Function in Different Neural Network Types

In this section, the generalizability of the proposed Si-CL loss function is compared using CNN and feedforward ANN architectures. The experiments were conducted on the Fashion-MNIST dataset using the Adam optimization algorithm, batch size 128, learning rate 0.001, and 50 epochs. This ensured that performance differences were solely due to architectural differences.

The feedforward ANN model was designed with two hidden layers (20 and 10 neurons) excluding the input layer and ReLU activation functions. The output layer uses softmax activation for classification. Table 12 shows the performance results obtained using the Si-CL loss function with CNN and feedforward ANN.

Table 12. *Performance of CNN and feedforward ANN architectures with the Si-CL loss function on the Fashion-MNIST dataset*

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Training Time
Si-CL	**94.24**	**94.25**	**94.24**	**94.24**	44 min. 11 sec.
Feedforward ANN	92.48	92.55	92.48	92.51	**37 min. 30 sec**.

As seen in the table, the Si-CL loss function delivers high performance in both network types. While the CNN architecture offers higher accuracy and F1-score, the feedforward ANN also produced effective results and provided faster training. These findings demonstrate that the Si-CL loss function is not specific to CNNs, but is also effective and generalizable across different types of neural networks.

5. DISCUSSION

This study systematically evaluated the performance of CNN architectures on the MNIST, EMNIST, and Fashion-MNIST datasets by analyzing the joint impact of different optimization algorithms and two loss functions: Cross-Entropy (Model 1) and the proposed Si-CL loss (Model 2). The findings provide a comprehensive perspective on how optimizer–loss interactions affect classification accuracy, convergence behavior, generalization capability, and training efficiency.

A detailed comparative summary of the experimental results is presented in Table 13, where the F1-score values of both models under different optimization algorithms are reported for all three datasets. As shown in Table 13, the Adam optimizer consistently achieved the highest performance across MNIST, EMNIST, and Fashion-MNIST. In both Model 1 and Model 2, Adam provided superior F1-score values, demonstrating stable and fast convergence behavior. This can be attributed to its adaptive learning rate mechanism and its effective handling of gradient variability. In contrast, SGDM generally required longer convergence times and yielded

slightly lower performance, reflecting the trade-off between classical momentum-based optimization stability and adaptive learning efficiency.

When comparing the two models in Table 13, Model 2 (Si-CL loss) outperformed Model 1 (Cross-Entropy loss) in almost all optimizer settings and across all datasets. The performance improvement is particularly evident when Si-CL is combined with Adam, where the highest F1-score values are observed consistently. These findings suggest that the Si-CL loss function enhances class separability and supports more discriminative feature learning in multi-class classification tasks.

Although Model 2 occasionally required slightly longer training durations than Model 1 under the same optimizer, this increase can be explained by the additional computational operations introduced by the Si-CL formulation. However, the improvement in classification accuracy and convergence stability outweighs this marginal increase in training time. Therefore, as reflected in Table 13, model evaluation should not be based solely on computational duration but should also consider accuracy, robustness, and generalization performance.

Table 13. *Comprehensive Comparative Summary of CNN Models Under Different Optimization Algorithms and Loss Functions*

Datasets	Optimization Algorithms	Model-1 (Cross entropy Loss) F1-score	Model-2 (Si-CL Loss) F1-score
MNIST	RMSProp	96.64	97.10
	SGDM	96.99	96.98
	Adam	**97.44**	**98.14**
	AdaGrad	95.96	96.68
	AdaMax	95.79	97.19
	AdaDelta	96.80	96.80
EMNIST	RMSProp	96.51	97.18
	SGDM	96.78	97.38
	Adam	**97.86**	**98.50**
	AdaGrad	96.06	96.76
	AdaMax	96.96	97.27
	AdaDelta	96.58	96.98
Fashion-MNIST	RMSProp	90.83	91.87
	SGDM	91.32	92.23
	Adam	**93.03**	**94.24**
	AdaGrad	89.69	90.62
	AdaMax	91.28	92.56
	AdaDelta	90.59	91.01

In addition to the optimizer-based comparison, a broader evaluation against previously published studies is provided in Table 14. As shown in Table 14, the proposed models achieve competitive and, in several cases, superior accuracy results compared to existing works on MNIST, EMNIST, and Fashion-MNIST. While MNIST represents a relatively simple handwritten digit classification task, Fashion-MNIST introduces higher intra-class similarity and greater classification difficulty. EMNIST further increases complexity due to its larger number of classes. Despite these varying difficulty levels, the proposed Model 2 consistently demonstrates strong performance across all datasets, confirming its robustness and generalization capability.

Table 14. *Comparative Classification Performance of the Proposed Model and Existing Studies on MNIST, EMNIST and Fashion-MNIST Datasets*

Related Work	Dataset	Accuracy (%)
(Xiao et al., 2017)	MNIST	97.30
(Agarap, 2018)	MNIST	95.36
(Wu, 2018)	MNIST	94.00
(Zhao & Jiao, 2019)	MNIST	98.71
(Zhang et al., 2026)	**MNIST**	**99.36**
Proposed Model 1	MNIST	97.44
Proposed Model 2	MNIST	98.14
(Szymański et al., 2026)	EMNIST	95.88
(Madhivanan & Mathivanan, 2022)	EMNIST	96.80
Proposed Model 1	EMNIST	97.86
Proposed Model 2	**EMNIST**	**98.50**
(Xiao et al., 2017)	Fashion-MNIST	89.7
(Agarap, 2018)	Fashion-MNIST	86.08
(Zhao & Jiao, 2019)	Fashion-MNIST	88.94
Proposed Model 1	Fashion-MNIST	93.03
Proposed Model 2	**Fashion-MNIST**	**94.24**

As shown in Table 14, the proposed model is compared with previously published studies conducted on the MNIST, Fashion-MNIST, and EMNIST datasets. The results demonstrate that the proposed approach achieves competitive and, in several cases, superior classification performance across all three benchmark datasets.

From an architectural standpoint, Batch Normalization was adopted to ensure stable gradient flow and consistent training behavior throughout all experiments reported in Table 13. Since a batch size of 128 was used in all configurations, no instability issues were observed. Preliminary analyses indicated that alternative normalization strategies exhibited similar performance trends; therefore, Batch Normalization was retained to maintain experimental consistency and enable fair comparisons among optimizers.

Overall, the findings summarized in Tables 13 and 14 clearly demonstrate that CNN performance is strongly influenced by the combined effect of the selected optimization algorithm and loss function. In particular, the integration of the Adam optimizer with the Si-CL loss function provides faster convergence, higher classification accuracy, and improved generalization across datasets of different complexity levels. These results highlight the importance of jointly designing optimization strategies and loss functions to enhance CNN-based multi-class image classification performance.

6.CONCLUSION

This study highlights the importance of carefully selecting optimization algorithms and loss functions to enhance the performance of convolutional neural networks. Experimental results obtained on the MNIST EMNIST, and Fashion-MNISTdataset demonstrate that the Adam optimizer consistently delivers superior performance in terms of accuracy and convergence behavior. Moreover, the proposed Si-CL loss function significantly improves classification accuracy, accelerates convergence, and enhances training efficiency when compared to the traditional cross-entropy loss.

The comparative analysis between Model 1 and Model 2 reveals that the performance gains achieved by Si-CL are not limited to specific training conditions but remain stable across different optimization strategies. These results confirm that the improvements stem from the intrinsic characteristics of the proposed loss function rather than from a particular hyperparameter configuration.

As a direction for future research, the applicability of the Si-CL loss function should be explored on larger and more complex datasets, as well as on different CNN architectures and computer vision tasks. Additionally, investigating the interaction between Si-CL and alternative normalization strategies or advanced hyperparameter optimization techniques may further enhance model performance. Such extensions would provide deeper insights into the generalizability and scalability of the proposed approach.

AUTHOR CONTRIBUTIONS

Conceptualization, Y.O.; methodology, Y.O.; fieldwork, P.E.; software, Y.O.; title, Y.O.; validation, Y.O.; laboratory work, Y.O.; formal analysis, Y.O.; data curation, Y.O.; visualization, Y.O.; supervision, Y.O.; project management, Y.O. Research (Literature Search) and Sources: P.E. Writing (Interpretation/Original Draft): Y.O. and P.E. Manuscript—Review and Editing: P.E. Both authors have read and legally accepted the final version of the article published in the journal. All human authors have read and legally accepted the final version under the CC BY-SA 4.0 framework.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

REFERENCES

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *arXiv*:1803.08375.

Akhtar, M., Tanveer, M., & Arshad, M. (2025). HawkEye: A robust loss function for regression with bounded, smooth, and insensitive zone characteristics. *Applied Soft Computing*, 113118. https://doi.org/10.1016/j.asoc.2025.113118

Arsenault, R., Poulin, A., Côté, P., & Brissette, F. (2014). Comparison of stochastic optimization algorithms in hydrological model calibration. *Journal of Hydrologic Engineering*, *19*(7), 1374-1384. https://doi.org/10.1061/(ASCE)HE.1943-5584.000093

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In: *Neural Networks: Tricks of the Trade*, 437-478, Springer, Berlin, Heidelberg.

Bottou, L. (2010). *Large-scale machine learning with stochastic gradient descent*. Proceedings of COMPSTAT'2010, 177-186, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-7908-2604-3_16

Chen, X., Yu, R., Ullah, S., Wu, D., Li, Z., Li, Q., Qi, H., Liu, J., Liu, M., & Zhang, Y. (2022). A novel loss function of deep learning in wind speed forecasting. *Energy*, 238, 121808. https://doi.org/10.1016/j.energy.2021.121808

Cohen, G., Afshar, S., Tapson, J., & Van Schaik, A. (2017, May). *EMNIST: Extending MNIST to handwritten letters*. In: 2017 international joint conference on neural networks (IJCNN*)* (pp. 2921-2926). IEEE.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(Jul), 2121-2159.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Gulzar, Y., Hamid, Y., Soomro, A. B., Alwan, A. A., & Journaux, L. (2018). A convolution neural network-based seed classification system. *Symmetry*, *12*(12), 2018. https://doi.org/10.3390/sym12122018

Hallen, R. (2017). A study of gradient-based algorithms. *arXiv:1704.06101*.

Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural Networks: Tricks of the Trade. In: Hinton, G., Srivastava, N., & Swersky, K. (Eds.), *Neural Networks: Tricks of the Trade*, 437-478, Berlin, Heidelberg, Springer.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv:1412.6980*.

Krizhevsky, A., Sutskever I., & Hinton G E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 1097-1105.

LeCun, Y., Bottou L., Bengio Y., & Haffner P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. https://doi.org/10.1109/5.726791

Liu, M., Yao, D., Liu, Z., Guo, J., & Chen, J. (2023). An improved Adam optimization algorithm combining adaptive coefficients and composite gradients based on randomized block coordinate descent. *Computational intelligence and neuroscience*, *2023*(1), 4765891. https://doi.org/10.1155/2023/4765891

Lin, C., Abe, S., Zheng, S., Li, X., & Chun, P. J. (2025). A structure-oriented loss function for automated semantic segmentation of bridge point clouds. *Computer-Aided Civil and Infrastructure Engineering*, *40*(6), 801-816. https://doi.org/10.1111/mice.13422

Madhivanan, V., & Mathivanan, P. (2022, March). A*n evaluation on the performance of privacy preserving split neural networks using EMNIST dataset*. In: International Conference on Deep Sciences for Computing and Communications (pp. 332-344). Cham: Springer Nature Switzerland.

Mokhtar, M. A., Fathy, M., Dahab, Y. A., & Sayed, E. A. (2025). A dual enhanced stochastic gradient descent method with dynamic momentum and step size adaptation for improved optimization performance. *Scientific Reports*, *15*(1), 40389. https://doi.org/10.1038/s41598-025-24689-y

Nouri, A., & Seyyedsalehi, S. A. (2023). Eigen value based loss function for training attractors in iterated autoencoders. *Neural Networks*, *161*, 575-588. https://doi.org/10.1016/j.neunet.2023.02.003

Ozkan, Y., & Erdogmus P. (2024). Evaluation of classification performance of new layered convolutional neural network architecture on offline handwritten signature images. *Symmetry*, *16*(6), 649. https://doi.org/10.3390/sym16060649

Reddi, S. J., Kale, S., & Kumar, S. (2018). On the convergence of Adam and beyond. https://doi.org/10.48550/arXiv.1904.09237

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, *22*(3), 400-407.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. https://doi.org/10.48550/arXiv.1609.04747

Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems*, *29*.

Shalev-Shwartz, S., Shamir, O., & Shammah, S. (2017). *Failures of gradient-based deep learning*. International Conference on Machine Learning, 3067-3075.

Shen, F., Shen, C., Zhou, X., Yang, Y., & Shen, H. T. (2016). Face image classification by pooling raw features. *Pattern Recognition*, *54*, 94-103. https://doi.org/10.1016/j.patcog.2016.01.010

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). *On the importance of initialization and momentum in deep learning*. Proceedings of the 30th International Conference on Machine Learning, 1139-1147, PMLR.

Szymański, J., Skarżyński, K., Szutenberg, B., Ratkowska, K., & Drywa, S. (2026). The dataset for extending EMNIST evaluation. *Scientific Data*, *13*(1), 73. https://doi.org/10.1038/s41597-025-06291-z

Tieleman, T., & Hinton G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*.

Wu, H. (2018). CNN-Based Recognition of Handwritten Digits in MNIST Database. *Research School of Computer Science*. The Australia National University, Canberra.

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

Yassine, E. L., Wang, S. L., Taresh, M. M., & Ali, T. A. A. (2024). Enhancing breast cancer diagnosis using deep learning and gradient multi-verse optimizer: a robust biomedical data analysis approach. *PeerJ Computer Science*, *10*, e2578. https://doi.org/10.7717/peerj-cs.2578/supp-1

Yazan, E., & Talu, M. F. (2017). *Comparison of the stochastic gradient descent based optimization techniques*. 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), 1-5. https://doi.org/10.1109/IDAP.2017.8090299

Yu, S., He, B., & Fang, L. (2025). Multi-step short-term forecasting of photovoltaic power utilizing TimesNet with enhanced feature extraction and a novel loss function. *Applied Energy*, 388, 125645. https://doi.org/10.1016/j.apenergy.2025.12564

Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method. *arXiv:1212.5701*.

Zhang, C., Tu, C., Wang, Z., Cao, W., & Cao, W. (2026). ResNet18-ThunderSVM: Hybrid intelligence for handwritten digit recognition by fusing deep spatial features and high-performance classification. *Scientific Reports*, *16*(1). https://doi.org/10.1038/s41598-026-38258-4

Zhao, J., & Jiao, L. (2019). Sparse deep tensor extreme learning machine for pattern classification. *IEEE Access*, *7*, 119181-119191. https://doi.org/10.1109/ACCESS.2019.2924647

Zheng, Y., Iwana, B. K., & Uchida, S. (2018). *Discovering class-wise trends of max-pooling in subspace*. 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR), 98-103. https://doi.org/10.1109/ICFHR-2018.2018.00026