RESEARCH ARTICLE **ISSN 2738-9898 (Print) ISSN 2738-9901 (Online)**

Design of an Efficient CNN Architecture with SGD Optimization for Accurate and Robust Object Classification

Bhesh Bahadur Thapa
School of Engineering, Faculty of Science and Technology, Pokhara University, Nepal
** Author to whom correspondence should be addressed; E-Mail:*bhesh.thapa@pu.edu.np

Received: 24 December 2025; Received in revised form: 23 January, 2026; Accepted: 25 January, 2026; Published: 29 January, 2026

Abstract

Deep learning has greatly improved image-based object classification in recent years. Among different models, Convolutional Neural Networks (CNNs) perform better than traditional image processing methods and simple neural networks. Despite their success, achieving a balance between classification accuracy, model complexity, and computational efficiency remains a key research challenge as classification accuracy and training cost strongly depend on network structure and learning strategy. In this study, an optimized deep CNN model for object classification is explored by carefully analyzing important architectural and training hyperparameters. Experiments are conducted using CIFAR-10 standard datasets. The proposed architecture is organized into three convolutional stages with increasing channel capacity. Batch normalization is utilized to stabilize gradient propagation, while max pooling is employed for spatial compression. Dropout-based regularization is strategically integrated to reduce overfitting. Additionally, adaptive average pooling is used before the classifier to maintain expressive feature learning while minimizing the overall parameter count. All experiments are conducted using GPU acceleration and optimized using momentum based minibatch Stochastic Gradient Descent (MSGD) algorithm to ensure stable and efficient learning. Model performance is evaluated in terms of classification accuracy, parameter efficiency, training time and number of training epochs. The experimental results demonstrate that the proposed CNN architecture achieves competitive accuracy on CIFAR-10 while maintaining a relatively low computational cost, highlighting the effectiveness of careful architectural design and optimization for efficient image classification. In addition, comparisons with existing state-of-the-art models show that the proposed method delivers higher classification accuracy and improved computational efficiency.

Keywords

Deep Learning, CNN, Computer Vision, Data Augmentation, Machine Learning, Optimization, Object Classification

1. Introduction

Artificial Neural Networks (ANNs) are machine learning models that learn patterns from data using interconnected processing units inspired by biological neurons. Unlike rule-based programming approaches, ANNs rely on interconnected processing units that adaptively learn patterns and relationships directly from data. Through iterative training, these models have demonstrated the ability to solve complex tasks that were traditionally considered difficult for machines, including image recognition, speech processing, facial analysis, object detection, and visual classification. The success of learning-based systems like neural networks depends not only on having a large amount of high-quality training data, but also on carefully testing and evaluating the model on new, unseen data to ensure that it can generalize well and perform accurately on real-world situations beyond the data it was trained on.

Neural networks are often described in terms of their depth, which refers to the number of layers that exist between the input and output nodes, commonly known as hidden layers. Traditionally, it was believed that a small number of hidden layers, typically two or three, were sufficient for a neural network to learn patterns effectively and perform its intended tasks. However, with the advancement of computational resources and a deeper understanding of network behavior, increasing the number of hidden layers allows the network to capture more complex and abstract representations of the input data. These high-dimensional features provide richer information, enabling the network to model intricate relationships and subtle patterns that simpler architectures might miss. Networks with multiple hidden layers are collectively known as Deep Neural Networks (DNNs), forming the foundation of many state-of-the-art models in machine learning. By stacking layers and increasing network depth thoughtfully, DNNs can achieve superior learning capacity and generalization performance, which is particularly important for complex pattern classification problems like Image recognition, Speech recognition etc. [1]. In computer vision applications, Convolutional Neural Networks (CNNs) are extensively employed due to their ability to capture local spatial patterns and build hierarchical feature representations. The effectiveness of Convolutional Neural Networks (CNNs) is strongly influenced by both their architectural design and training strategies, including choices such as filter dimensions, number of filters, network depth, and the arrangement of hidden layers. This research work aims to develop CNN architecture optimized for object classification and to provide clear, systematic guidance on selecting hyperparameters that achieve an optimal balance between model accuracy and computational cost

1.2 Convolution Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a specialized variant of traditional Multilayer Perceptron (MLP) architectures that integrate one or more convolution layers, optional pooling layers and final fully connected layers. Convolutional layers enforce local connectivity by linking each neuron only to a small receptive field of the previous layer and apply weight sharing across spatial locations to extract hierarchical features from structured inputs such as images [2]. Pooling decreases feature map dimensions by merging values from nearby spatial locations, thereby decreasing computational complexity and improving translation invariance in CNNs [2]· The fully connected layer densely links all neurons from the preceding layer so that the aggregated feature representations can be transformed into the final classification or recognition task output [3].

In this structure, each neuron still has learnable weights and biases and performs a weighted sum followed by a non-linear activation, but compared with ordinary MLPs, CNNs dramatically reduce the number of parameters while capturing spatially invariant patterns and enabling efficient learning of both low-level and high-level representations, which has contributed to their state-of-the-art performance across a range of computer vision and pattern recognition tasks in recent literature [3]· Consider a simple example where the input is an RGB image of size 32×32. In a fully connected neural network, each input pixel is connected to every neuron in the next layer, resulting in 32 × 32 × 3 = 3072 input weights for just one neuron. While this number is relatively manageable, the situation becomes significantly worse as the image size increases. For instance, an RGB image of size 200 × 200 * 3 =120,000 input values, meaning each neuron would require 120,000 weights. Such many parameters lead to high computational cost and increased memory usage. More importantly, the excessive number of weights makes the model highly prone to overfitting, as it can easily memorize the training data rather than generalize well to unseen images. Therefore, fully connected models are inefficient for high-dimensional image data, and this limitation motivates the use of convolutional neural networks, which significantly reduce parameters through local connectivity and weight sharing, thereby improving generalization and reducing overfitting

Convolutional Neural Networks (CNNs or ConvNets) are a specialized form of multilayer perceptions that are widely used for image recognition and classification tasks. Unlike traditional MLPs, where every neuron is fully connected to the next layer, CNNs employ local connectivity, meaning each neuron is connected only to a small, localized region of the previous layer. This region, known as the receptive field, enables the network to capture important local features such as edges and textures while significantly reducing the number of learnable parameters. By stacking multiple convolutional layers, CNNs progressively combine local features into more complex and abstract representations, which explains their effectiveness in visual learning tasks. Neural networks are often characterized by their depth, which refers to the number of layers between the input and output, commonly called hidden layers. Earlier assumptions suggested that networks with only two or three hidden layers were sufficient for most learning tasks. However, subsequent research demonstrated that increasing the number of layers enables neural networks to learn more complex and high-dimensional representations of input data [4]. Networks with many such layers are known as Deep Neural Networks (DNNs). Convolutional Neural Networks (CNNs) are a prominent example of DNNs and have achieved remarkable success in computer vision applications such as image classification and object recognition as shown in Figure 1.

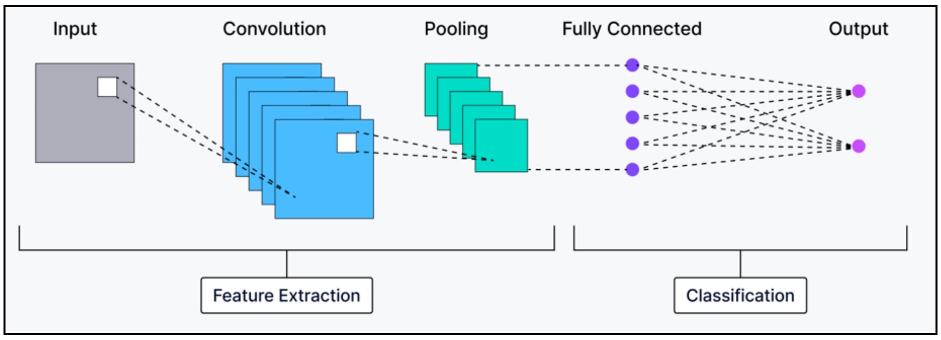


Figure 1 : CNN Architecture [5]

2. Methodology

2.1 Dataset Preparation

Initially, the data are divided into training, validation, and test data sets. The training data are responsible for guiding the learning process by tuning the model's weights and internal parameters. In contrast, the validation set is not utilized for parameter updates, rather, it serves to analyze the model's predictive performance, detect potential overfitting, and evaluate the efficiency of the convolutional neural network (CNN) during training. After training is complete, model performance is examined using the test set, which contains data unseen by the model. In this study, standard CIFAR-10 dataset is utilized and is shown in Figure 2. The CIFAR-10 dataset consists of 60,000 color (RGB) images with a resolution of 32×32 pixels, along with a separate set of 10,000 validation samples. Among these 50000 training datasets, 40,000 images are allocated to the training set for model optimization, while the remaining 10,000 images are reserved for testing to evaluate model performance. The dataset contains 10 distinct object categories with each class represented by 5,000 training images and 1,000 validation images. Further, CIFAR-10 maintains class balance by including an equal number of samples for each category. Its moderate size and diversity make CIFAR-10 an ideal standard for assessing the performance of convolutional neural networks and other machine learning algorithms [6].

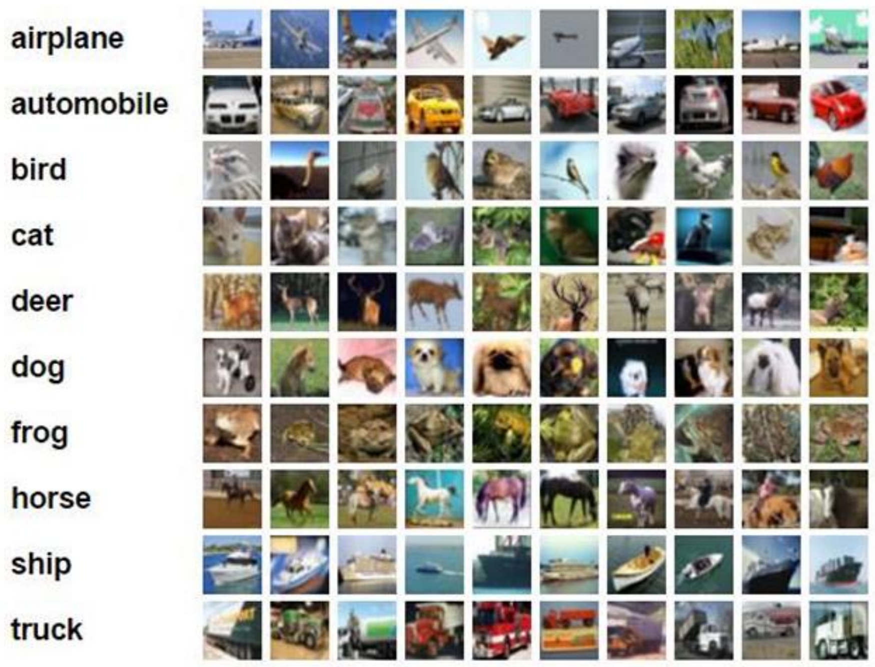


Figure 2: Visualization of CIFER-10 Dataset

Different classes of CIFER-10 dataset are shown in Table 1.

Table 1 : Cifer-10 Dataset Class and Label [6]

S. N	Class	No. of Training Images	No. of Validation Images	No. of testing Images	Label
1	Airplane	4000	1000	1000	0
2	Automobile	4000	1000	1000	1
3	Bird	4000	1000	1000	2
4	Cat	4000	1000	1000	3
5	Deer	4000	1000	1000	4
6	Dog	4000	1000	1000	5
7	Frog	4000	1000	1000	6
8	Horse	4000	1000	1000	7
9	Ship	4000	1000	1000	8

10	Truck	4000	1000	1000	9

2.2 Architecture Design

A Convolutional Neural Network (CNN) consists of an input layer, an output layer, and multiple hidden layers, including convolutional, pooling, and fully connected layers, which can be combined and stacked in various configurations to form a complex architecture capable of handling diverse classification tasks as depicted in Figure 3. The design and performance of CNNs are highly dependent on the target task, requiring careful tuning of hyperparameters such as the number of layers, kernel sizes, number of filters, stride lengths, and dropout rates to balance computational cost and classification accuracy. For instance, shallow networks with fewer layers and small kernels are often sufficient for simpler tasks such as MNIST digit recognition, whereas complex datasets like CIFAR-10 or ImageNet demand deeper and wider networks with larger receptive fields to capture intricate patterns. To address these challenges, the proposed lightweight CNN architecture employs a three-block convolutional design with progressively increasing feature channels, batch normalization after each convolution, and adaptive average pooling before the classifier, enabling effective hierarchical feature extraction while reducing trainable parameters and mitigating overfitting.

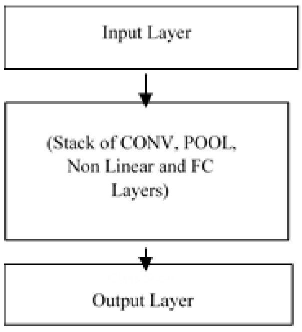


Figure 3: CNN Layers

2.3 Convolution Layer Design

The convolutional layer extracts hierarchical features from the input, with early layers capturing low-level patterns such as edges and corners, and deeper layers learning higher-level representations. Each layer contains a set of learnable filters, applied across local regions called receptive fields, which are shared across neurons to detect specific features. By stacking multiple filters, the depth of the layer increases, enabling the network to learn rich feature representations through linear filter weights applied as convolutions over the previous layer which can be shown in Figure 4.

$$O[m,n] = f[m,n] * g[m,n] = \sum_{u=-\infty}^{\infty} f[u,v]g[m-u,n-v]$$

$$\begin{array}{|c|c|c|c|c|}\hline 22 & 15 & 1 & 3 & 60 \\ \hline 42 & 5 & 38 & 39 & 7 \\ \hline 28 & 9 & 4 & 66 & 79 \\ \hline 0 & 82 & 45 & 12 & 17 \\ \hline 99 & 14 & 72 & 51 & 3 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 0 & 0 & 1 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 1 & 3 & 60 \\ \hline 38 & 39 & 7 \\ \hline 4 & 66 & 79 \\ \hline \end{array}$$

5 * 5 Input Image(f) 3 * 3 Convolution Filter (g) f * g

Figure 4: Convolution Operation

In a convolutional neural network, the convolutional layer processes an input feature volume of size W1 × H1 × D1 and produces a corresponding output feature volume with dimensions W2 × H2 × D2. The operation of this layer is governed by four key hyperparameters: the number of filters (K), the spatial size of each filter (F), the stride length (S), and the amount of zero-padding applied (P). The dimensions of the resulting output volume are determined based on these hyperparameters as follows:

$$\mathbf{W2} = \frac{W1 - F + 2P}{S} + 1$$

$$\mathbf{H2} = \frac{H1 - F + 2P}{S} + 1$$

$$\mathbf{D2} = K$$

2.4 Pooling / Sub Sampling Layer Design

The pooling (subsampling) layer reduces feature representations by decreasing their spatial dimensions, thereby reducing the size of the input data without discarding critical characteristics leading to lower computational cost, decreased memory consumption, and increased robustness to small input variations. This dimensionality reduction is typically achieved using either max pooling or average pooling operations. In both approaches, the input is partitioned into non-overlapping two-dimensional regions, over which the pooling operation is applied. For example, a pooling kernel of size 2 × 2 with a stride of 2 units slides across the input to generate the down sampled output, as illustrated in the Figure 5.

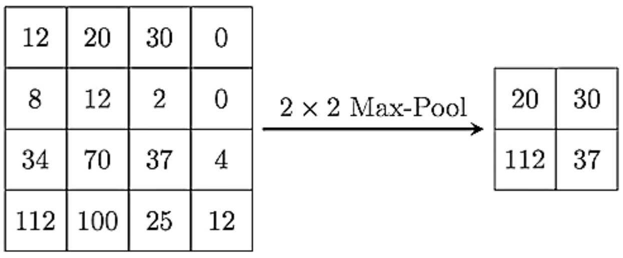


Figure 5: Max Pooling Operation

A pooling layer takes an input volume of size W1 × H1 × D1 and produces an output volume of size W2 × H2 × D2. Its operation is governed by two key hyperparameters: the size of the pooling region (F) and the stride length (S). The resulting output dimensions are determined based on these hyperparameters as follows: While such fixed pooling operations effectively reduce spatial resolution, they require careful manual tuning and may increase architectural rigidity. In the proposed architecture, adaptive average pooling is employed to overcome these limitations. Unlike traditional pooling layers, adaptive average pooling dynamically adjusts its pooling regions to produce a predefined output size, regardless of the input feature map dimensions. By incorporating adaptive average pooling before the final classification stage, the spatial information of each feature map is condensed into a single representative value per channel, producing a compact global feature representation. This approach substantially reduces the number of trainable parameters and removes the dependency on large fully connected layers, thereby lowering computational cost and improving parameter efficiency. Global and adaptive pooling strategies have been used to simplify CNN architectures while preserving discriminative capability, contributing to improved generalization and reduced overfitting. Consequently, adaptive average pooling serves as an effective mechanism for balancing classification performance, architectural complexity, and computational efficiency in convolutional neural networks [7].

2.5 Non-linear layers

CNNs employ non-linear activation functions to enable the network to identify complex features at each hidden layer. The Rectified Linear Unit (ReLU) is commonly used for this purpose, implementing the operation y=max (0, Z), which preserves the input and output dimensions of the layer. By introducing non-linearity, ReLU enhances the representational capacity of the network and the decision function without altering the receptive fields of the convolutional layers. Additionally, ReLU offers the practical benefit of significantly faster training compared to other non-linear activation functions and is shown in Figure 6[8].

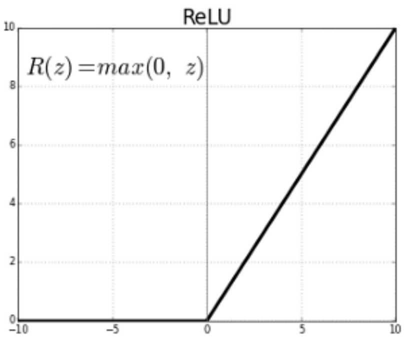


Figure 6: ReLU Activation Function

2.6 Fully connected layers

Fully connected layers typically serve as the final stages of a CNN, integrating features extracted by preceding layers to produce the network's output. Each neuron in a fully connected layer computes a weighted sum of all the activations from the previous layer, effectively combining every feature to determine the contribution toward a specific output. This complete connectivity allows the layer to learn complex, global patterns and relationships across the entire feature set. By connecting all features, fully connected layers enable the network to perform final decision-making, such as Classification, consolidating both low-level and high-level representations captured by earlier convolutional and pooling layers.

2.7 Data Augmentation

In image classification tasks, input images provided to a CNN can vary in multiple ways, including orientation, scale, noise, and stability. Consequently, the model must be capable of handling and accurately interpreting a wide range of input variations. Training the network on every possible variation of each image, however, is impractical. This challenge can be effectively addressed using image data augmentation, a technique that generates transformed versions of existing images according to specified operations. By applying this method, the size of the training dataset is artificially increased, enhancing the diversity of input samples and reducing the risk of overfitting. Figure 7 illustrates the impact of various data augmentation strategies on a subset of CIFAR-10 dataset images.

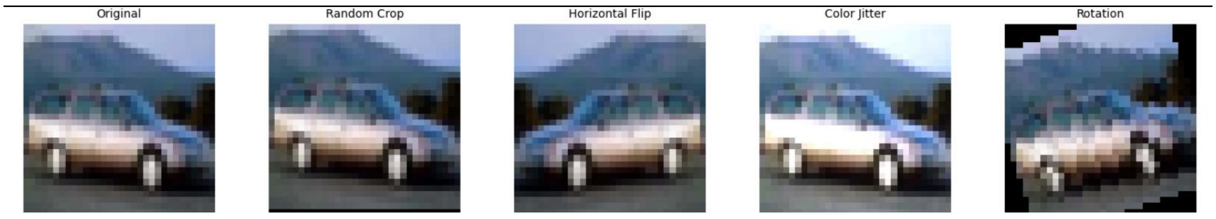


Figure 7: Visualization of Data Augmentation Operations for 'Automobile' class in CIFAR-10 Dataset

2.8 Learning Classifier

Once an efficient convolutional neural network architecture has been structured, the next step is to implement a supervised learning algorithm that enables the model to learn to classify objects from the input data as shown in Figure 8. In supervised learning, the network is trained with labeled examples, allowing it to iteratively adjust its parameters to minimize a defined loss function that measures the difference between predicted outputs and true labels. This process typically employs optimization techniques such as backpropagation combined with stochastic gradient descent, which guides weight updates to improve classification accuracy over successive iterations. By leveraging the large volume of labeled training samples, supervised training enables the CNN to capture complex data patterns and generalize well to new, unseen samples, which is critical for robust classification performance in real-world applications [9].

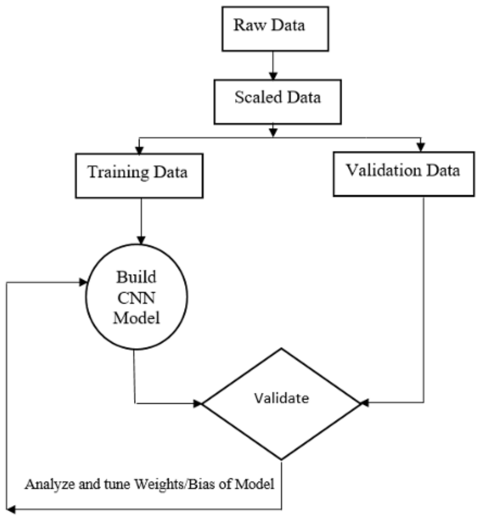


Figure 8: Model for Supervised Learning

This algorithm enables the network to learn optimal weights and biases such that its output closely approximates y(x) for all training inputs x. To achieve this, a cost function, also referred to as an objective function or loss function, is defined to quantify the difference between the predicted outputs and the true labels.
In classification tasks, this is commonly formulated using the negative log-likelihood of the predicted class probabilities.

$$NLL(\theta, \mathcal{D}) = -\sum_{i=0}^{|\mathcal{D}|} \log P(Y = y^{(i)}(x^{(i)}), \theta)$$

Where $\mathcal{D}$ is Dataset, θ is weight parameter, $(x^{(i)}, y^{(i)})$ is ith training data and Y is the target data. The training process is considered successful when it identifies weights and biases that minimize the cost function. To achieve this, optimization is typically performed using the gradient descent algorithm, which iteratively adjusts the parameters in the direction of the steepest reduction of the loss.

2.9 Gradient Descent

After defining a cost function, the training algorithm aims to minimize it by optimizing the network's weights and biases. These parameters are initially assigned small random values to break symmetry and are iteratively refined using gradient-based optimization. Adaptive optimizers like Adam, RMSProp, and Adagrad often converge faster in early training but do not generalize as well as SGD in many deep learning tasks like computer vision and classification [10]. In each iteration, the gradient of the loss with respect to the weights and biases is calculated and propagated backward through the network via the backpropagation algorithm. Variants such as stochastic gradient descent accelerate convergence and enhance stability, allowing the network to effectively capture complex patterns from large-scale datasets [10].

2.10 Stochastic Gradient Descent (SGD)

A cost function is determined by the network's parameters, and the ordinary gradient descent algorithm minimizes this function by iteratively moving in the direction of steepest descent on the error surface. Stochastic Gradient Descent (SGD) follows the same principle but updates parameters more frequently, using gradients computed from a small subset of training examples rather than the entire dataset. A popular variant of SGD is Mini-Batch Gradient Descent (MSGD), in which the training data is partitioned into multiple batches, and gradients are calculated and applied after processing each batch. Much research shows that MSGD and its momentum variants often yield better generalization performance than adaptive optimizers such as Adam in non-convex deep learning settings. SGD with momentum adds a memory of previous gradients to the current update. Study shows that while Adam optimizes faster, SGD frequently achieves superior generalization in practical settings. Thus, MSGD with momentum can be considered the most reliable choice when the primary goal is robust performance on unseen data rather than solely faster convergence [11][12]. The general procedure for MSGD with momentum is outlined in the flowchart in Figure 9.

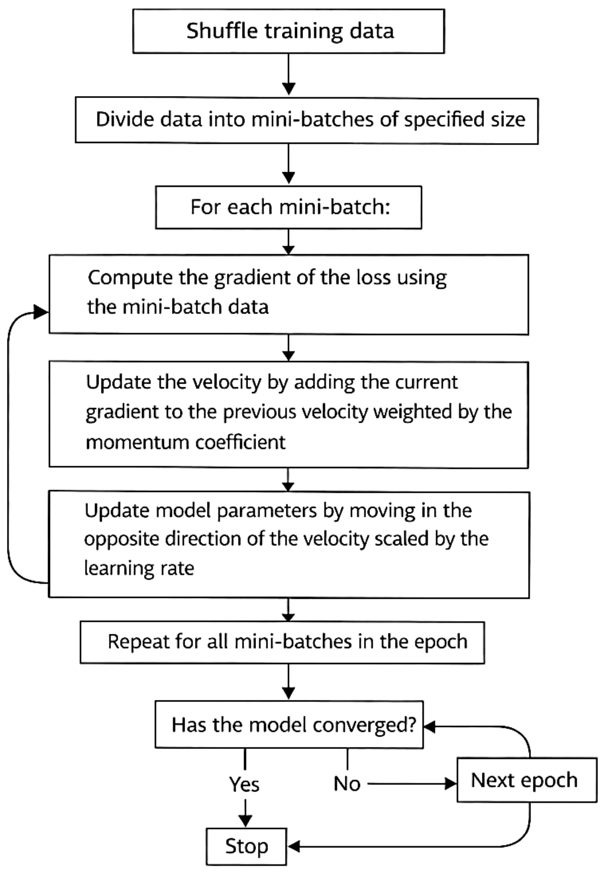

Shuffle training data
Divide data into mini-batches of specified size
For each mini-batch:
Compute the gradient of the loss using the mini-batch data
Update the velocity by adding the current gradient to the previous velocity weighted by the momentum coefficient
Update model parameters by moving in the opposite direction of the velocity scaled by the learning rate
Repeat for all mini-batches in the epoch
Has the model converged?
Yes
No
Next epoch
Stop

Figure 9: Flowchart for MSGD Learning Optimization

The algorithm begins by initializing the model parameters and setting the velocity to zero. For each epoch, the training dataset is first shuffled and then divided into mini batches of a specified size. For each mini-batch, the gradient of the loss function is computed with respect to the model parameters using only the samples in that mini-batch. The velocity is then updated by multiplying the previous velocity by the momentum coefficient and adding the current mini-batch gradient. The model parameters are updated by moving in the direction of the negative velocity scaled by the learning rate. This process is repeated for all mini batches in the epoch and continues for the specified number of epochs.

2.11 Overfitting and Dropout

Throughout the training process, a CNN identifies visual patterns by iteratively updating the parameters associated with neurons in its hidden layers. These weights determine how strongly each neuron responds to specific features, allowing the network to detect edges, textures, and higher-level patterns. When the training dataset is small, the network may perform very well on the training data but fail to generalize to validation or test data, this problem is known as overfitting [13]. Overfitting occurs because the network memorizes specific details of the training images rather than learning features that apply broadly to new images. Techniques such as dropout, where a random subset of neurons is temporarily disabled during training, help prevent overfitting by encouraging the network to develop more robust and generalizable feature representations [14].

Overfitting in a CNN model can be detected by comparing the accuracy of the training and validation datasets over successive iterations or epochs. Typically, a plot of these accuracies reveals overfitting when the training accuracy continues to improve steadily, while the validation accuracy plateaus or fails to increase at the same rate. The figure below illustrates this scenario, showing a divergence between training and validation performance indicative of overfitting which is shown in Figure 10.

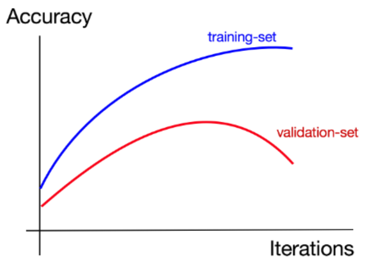


Figure 10: Training vs Validation Accuracy

Dropout is one of the regularization techniques to minimize over-fitting problems. It prevents complex co-adaptation on the training data. Since a fully connected layer occupies most of the parameters, it is prone to overfitting. In this technique, some of the hidden and visible nodes selected randomly with probability P and are dropped. It can also be viewed as model averaging neural networks. Dropout technique not only builds an efficient model by minimizing overfitting problems but also reduce the training time required. The below figure shows how dropout can disintegrate a complex neural structure to smaller networks with less connections thus achieving model averaging principle.

Dropout is a regularization technique used to reduce overfitting and enhance the generalization of neural networks. During training, a random subset of neurons is deactivated based on a probability P, which limits reliance on specific feature patterns. This method is particularly effective in fully connected layers, where the number of parameters is high. By reducing overfitting and promoting more robust feature learning, dropout not only improves model generalization but can also decrease training time. Figure 11 illustrates how dropout decomposes a complex network into smaller subnetworks with fewer connections, thereby implementing the principle of model averaging.

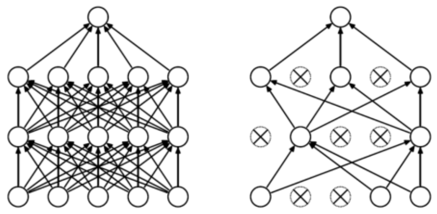


Figure 11: Standard CNN vs CNN After Applying Dropout

The term unit dropout denotes the temporary deactivation of a neuron and the elimination of all connections linked to it during training.

2.12 Testing and Evaluation

The performance of the proposed model is evaluated using the testing dataset from the CIFAR-10 datasets, comprising 10,000 images. Model evaluation is conducted using multiple performance metrics across testing datasets as mentioned below

i. Confusion Matrix
ii. Precision, Recall and F Measure

iii. Validation and Training Accuracy
iv. Number of Parameters, Model Size and Training Time to achieve maximum accuracy

Considering these parameters collectively allows for a comprehensive evaluation of the proposed CNN, capturing both predictive performance and computational efficiency, thereby providing a balanced perspective on its suitability for practical image classification applications.

2.13 Confusion Matrix

The confusion matrix is a performance evaluation tool that organizes prediction results into a matrix format, enabling a direct comparison between true class labels and the classifier's predictions. It organizes the classifier's predictions against the actual class labels, typically with one axis representing actual classes and the other representing predicted classes, allowing easy visualization of correct and incorrect classifications [15]. For binary classification problems, model predictions can be grouped into four distinct outcomes: True Positives (TP), which indicate correct detection of positive samples; False Positives (FP), occurring when negative samples are incorrectly predicted as positive; False Negatives (FN), arising when positive samples are misclassified as negative; and True Negatives (TN), where negative samples are accurately identified. The confusion matrix is a foundation for deriving important evaluation metrics such as precision, recall, and F1-score, which quantify different aspects of classifier performance beyond simple accuracy.

For any class X, the confusion matrix is given as

Actual Class

		X	Not X
	X	TP	FP
Predicted Class	Not X	FN	TN

Figure 12: Confusion Matrix

Correct predictions appear along the main diagonal of the matrix, while off-diagonal entries indicate misclassifications as shown in Figure 12.

2.13 Precision, Recall and F1 Measure

Precision, recall, and F1-score are key metrics for evaluating the performance of a CNN classifier beyond simple accuracy. Precision measures the proportion of correctly predicted positive instances among all instances labeled as positive by the model, while recall quantifies the proportion of actual positive instances that are correctly identified. For example, in a medical imaging application for detecting pneumonia from chest X-rays, high recall ensures that most patients with pneumonia are correctly identified, but if precision is low, many healthy patients may be incorrectly flagged as positive. Conversely, a model with high precision but low recall would correctly identify positive cases while missing a significant number of actual patients with pneumonia. Since optimizing one metric does not guarantee good performance on the other, the F1-score, defined as the harmonic means of precision and recall, provides a balanced measure of model performance. These metrics can be computed using the counts of True Positives (TP), False Positives (FP), and False

Negatives (FN) derived from the confusion matrix, allowing a comprehensive evaluation of the CNN's predictive capability. Precision, recall and F1 score can be calculated as below

- Precision = TP / (TP + FP)
- Recall = TP (TP + FN)
- F1 Score = 2 (Precision * Recall)/ (Precision + Recall)

2.14 Validation and Testing Accuracy

When developing a CNN model, it is essential to utilize two distinct datasets first, a training set and second a testing or validation dataset. The network learns feature representations and optimizes its parameters using the training data, while the validation dataset is used to evaluate its generalization performance. The resulting metrics are referred to as training accuracy and validation accuracy, respectively as shown in Figure 13. A large difference between training and validation accuracies typically signals overfitting, where the model excels on the training set but struggles to generalize to new data. It is therefore essential to analyze and address this discrepancy to achieve stable and reliable model performance. Careful analysis and mitigation of this gap are crucial to ensure robust and reliable model performance.

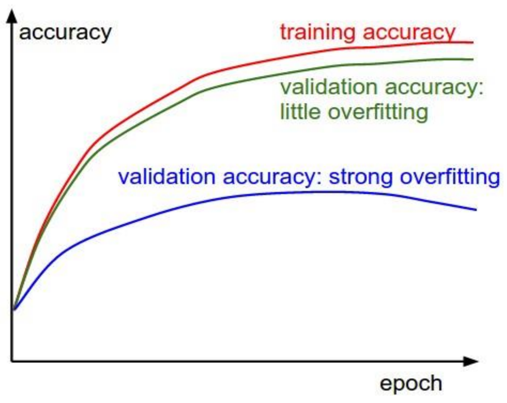


Figure 13: Validation/ Training Accuracy

2.15 Number of Parameters, Model Size and Training Time to achieve maximum accuracy

In addition to accuracy-based metrics, the efficiency of a convolutional neural network (CNN) is evaluated using several computational and model complexity parameters, which are essential for assessing its practical applicability. The number of parameters reflects the total trainable weights within the network, including convolutional kernels, fully connected layer weights, and biases, and provides an indication of the model's complexity and memory footprint. While larger networks can capture more intricate features, they also require more memory and may be prone to overfitting, particularly with limited data.

Model size, typically expressed in megabytes, denotes the storage needed to save the trained network and is influenced by both the number of parameters and data precision, where smaller models improve memory efficiency and ease of deployment.

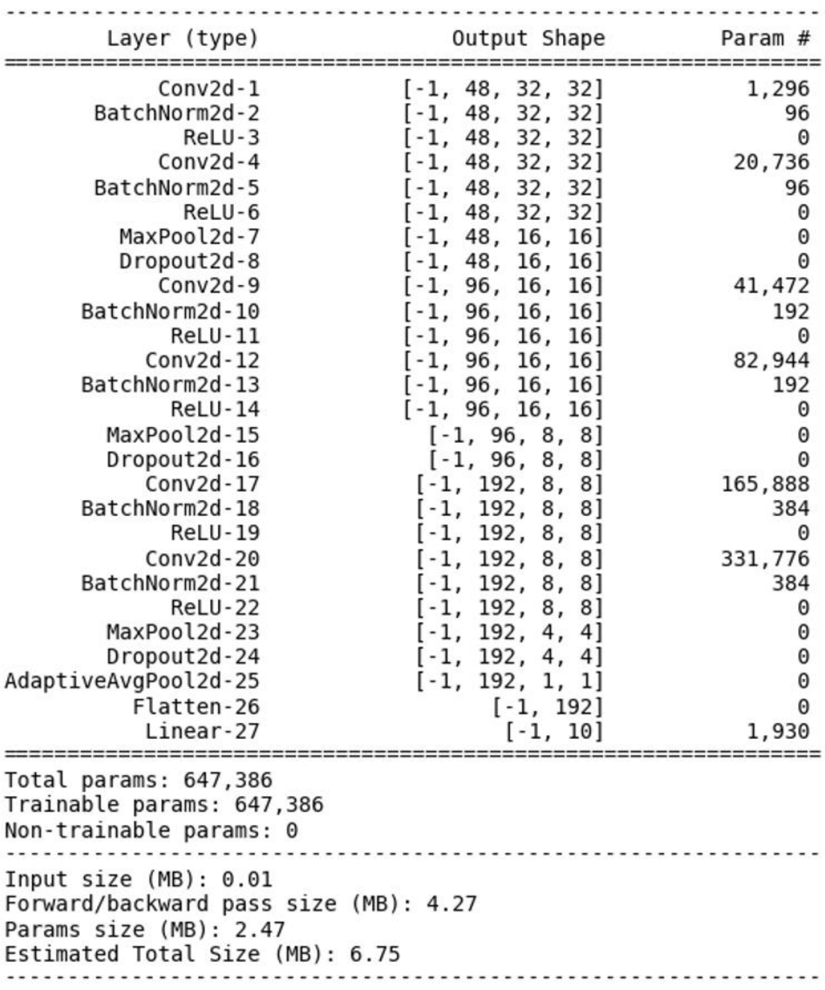
Finally, training time quantifies the duration required for the network to reach its optimal accuracy and is affected by factors such as model depth, parameter count, batch size, learning rate, and the computing hardware used.

2.16 Implementation

The experiments were carried out using the PyTorch deep learning framework, which supports computations on both CPU and GPU, providing an efficient environment for designing, training, and evaluating CNNs. Several CNN architectures were developed and tested on the CIFAR-10 datasets by systematically changing key hyperparameters, such as the number of layers, filter sizes, number of filters, and by applying techniques like data augmentation, global and adaptive pooling, and batch normalization. From these experiments, the best-performing architecture for the CIFAR-10 dataset is presented below and is shown in Figure 14, showing how network design and hyperparameter choices influence both classification accuracy and computational efficiency.

Architecture Specification for Efficient CNN Architecture for CIFAR-10 Dataset

i. INPUT Layer: Accepts input images of size 32 × 32 with 3 channels (RGB).
ii. First CONV Layer: Convolution with 48 filters of size 3 × 3, followed by Batch Normalization and ReLU activation.
iii. Second CONV Layer: Convolution with 48 filters of size 3 × 3, followed by Batch Normalization and ReLU activation.
iv. MAXPOOL Layer: Max pooling of size 2 × 2, followed by 10% Dropout.
v. Third CONV Layer: Convolution with 96 filters of size 3 × 3, followed by Batch Normalization and ReLU activation.
vi. Fourth CONV Layer: Convolution with 96 filters of size 3 × 3, followed by Batch Normalization and ReLU activation.
vii. MAXPOOL Layer: Max pooling of size 2 × 2, followed by 20% Dropout.
viii. Fifth CONV Layer: Convolution with 192 filters of size 3 × 3, followed by Batch Normalization and ReLU activation.
ix. Sixth CONV Layer: Convolution with 192 filters of size 3 × 3, followed by Batch Normalization and ReLU activation.
x. MAXPOOL Layer: Max pooling of size 2 × 2, followed by 30% Dropout.
xi. Global Adaptive Average Pooling: Reduces the spatial dimension of each feature map to 1 × 1.
xii. Flatten Layer: Converts the pooled output into a 192-dimensional vector.
xiii. Fully Connected / OUTPUT Layer: Linear layer mapping 192 units directly to 10 output classes.



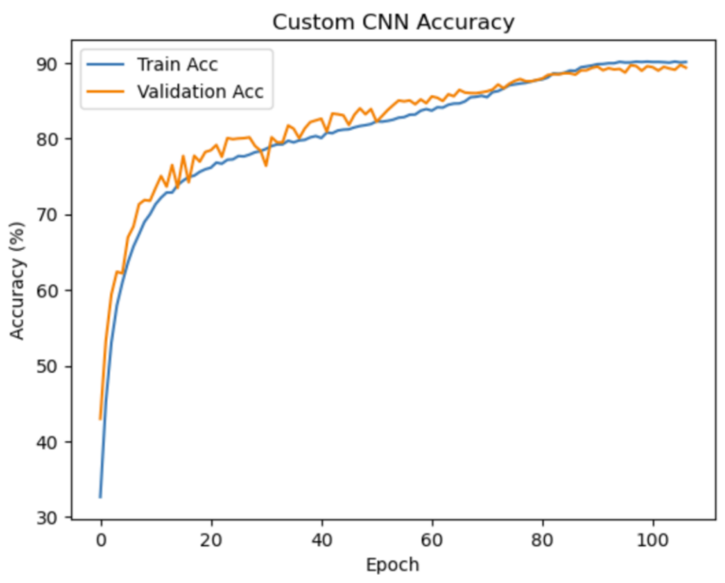
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 48, 32, 32]           1,296
       BatchNorm2d-2           [-1, 48, 32, 32]              96
              ReLU-3           [-1, 48, 32, 32]               0
            Conv2d-4           [-1, 48, 32, 32]          20,736
       BatchNorm2d-5           [-1, 48, 32, 32]              96
              ReLU-6           [-1, 48, 32, 32]               0
         MaxPool2d-7           [-1, 48, 16, 16]               0
         Dropout2d-8           [-1, 48, 16, 16]               0
            Conv2d-9           [-1, 96, 16, 16]          41,472
      BatchNorm2d-10           [-1, 96, 16, 16]             192
             ReLU-11           [-1, 96, 16, 16]               0
           Conv2d-12           [-1, 96, 16, 16]          82,944
      BatchNorm2d-13           [-1, 96, 16, 16]             192
             ReLU-14           [-1, 96, 16, 16]               0
        MaxPool2d-15             [-1, 96, 8, 8]               0
        Dropout2d-16             [-1, 96, 8, 8]               0
           Conv2d-17            [-1, 192, 8, 8]         165,888
      BatchNorm2d-18            [-1, 192, 8, 8]             384
             ReLU-19            [-1, 192, 8, 8]               0
           Conv2d-20            [-1, 192, 8, 8]         331,776
      BatchNorm2d-21            [-1, 192, 8, 8]             384
             ReLU-22            [-1, 192, 8, 8]               0
        MaxPool2d-23            [-1, 192, 4, 4]               0
        Dropout2d-24            [-1, 192, 4, 4]               0
AdaptiveAvgPool2d-25            [-1, 192, 1, 1]               0
          Flatten-26                  [-1, 192]               0
           Linear-27                   [-1, 10]           1,930
================================================================
Total params: 647,386
Trainable params: 647,386
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 4.27
Params size (MB): 2.47
Estimated Total Size (MB): 6.75
----------------------------------------------------------------
```

Figure 14: Efficient CNN Architecture for CIFAR-10 Dataset

3. Results and Discussion

The primary objective of this research is to develop an efficient convolutional neural network (CNN) architecture for accurate object classification on the CIFAR-10 dataset through systematic fine-tuning of hyperparameters and the application of data augmentation techniques. In addition, this study aims to demonstrate an effective methodology for selecting suitable hyperparameter configurations when designing compact and efficient CNN architectures. The performance of the proposed CNN model was evaluated using standard classification metrics, including training accuracy, testing accuracy, and confusion matrix analysis. Figure 15 illustrates the training and testing accuracy curves, showing stable convergence and consistent improvement across epochs. The final model achieved a classification accuracy of 91.11% on the CIFAR-10 test dataset, indicating strong generalization performance. Figure 16 presents the confusion matrix, which demonstrates that the proposed model performs well across all ten CIFAR-10 classes, with only minor misclassifications among visually similar categories. This reflects the model's ability to learn discriminative and class-specific features effectively. Table 2 summarizes the quantitative performance metrics, further confirming the robustness of the proposed architecture.

3.1 Performance evaluation of CNN architecture for CIFAR-10 Dataset



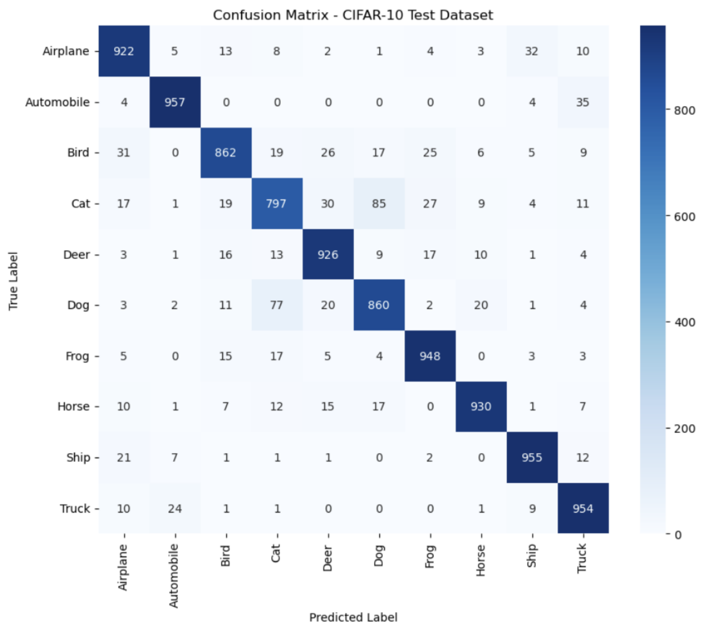


Figure 15: Train and Test Accuracy

Figure 16: Confusion Matrix

Table 2: Performance Metric

```
              precision    recall  f1-score   support

           0      0.899     0.922     0.910      1000
           1      0.959     0.957     0.958      1000
           2      0.912     0.862     0.886      1000
           3      0.843     0.797     0.820      1000
           4      0.903     0.926     0.915      1000
           5      0.866     0.860     0.863      1000
           6      0.925     0.948     0.936      1000
           7      0.950     0.930     0.940      1000
           8      0.941     0.955     0.948      1000
           9      0.909     0.954     0.931      1000

    accuracy                          0.911     10000
   macro avg      0.911     0.911     0.911     10000
weighted avg      0.911     0.911     0.911     10000
```

The optimized CNN was trained for 97 epochs, with an average training time of approximately 30 seconds per epoch, highlighting its computational efficiency. Compared with lightweight CNN architecture MobileNetV*x* trained under standard settings, which contain approximately 1.5 million parameters, and deeper and more complex architectures such as VGG-16 and ResNet-18 trained under standard settings, which contain approximately 138 million and 11.7 million parameters respectively, the proposed model achieves competitive classification accuracy with a significantly reduced parameter count [16],[17],[18],[19],[20],[21]. The final efficient architecture contains only 0.6 million learnable parameters providing a classification accuracy of 91.11%, making it significantly more compact while maintaining high accuracy. The optimized lightweight CNN architecture consists of three convolutional blocks with progressively increasing channel dimensions (48, 96, and 192). The use of 3×3 convolutional filters enable effective feature extraction while limiting parameter growth. Batch normalization improves training stability, while max-pooling layers reduce spatial dimensionality and enhances receptive field efficiency. Stage-wise dropout (0.1, 0.2, and 0.3) mitigates overfitting in deeper layers, and adaptive average pooling significantly reduces the number of parameters in the classification stage. Optimization using minibatch stochastic gradient descent (MSGD), combined with data augmentation, further contributes to improved generalization performance. Several architectural insights are supported by the experimental results:

- The use of 3×3 convolutional filters enable fine-grained feature extraction with fewer parameters.
- Progressive increase in filter depth enhances the network's ability to capture high-level semantic features.
- Batch normalization improves training stability and allows the use of higher learning rates.
- Stage-wise dropout rates (0.1, 0.2, and 0.3) effectively reduce overfitting in deeper layers.
- Max pooling between convolutional blocks efficiently reduces spatial resolution while preserving important features.
- Global adaptive average pooling significantly reduces parameter count and improves generalization.
- Stochastic Gradient Descent (SGD) provides stable and efficient optimization.
- Data augmentation techniques further improve generalization and contribute to higher classification accuracy.

Overall, the results demonstrate that careful architectural design combined with appropriate hyperparameter selection and regularization strategies can yield an efficient light weight CNN model with strong classification performance on CIFAR10 dataset.

4. Conclusions

This study investigates the design of an efficient convolutional neural network (CNN) architecture for image classification on the CIFAR-10 dataset. Through systematic exploration of architectural depth, filter dimensions, regularization strategies, and optimization parameters, an optimized CNN model was developed that achieves 91.11% classification accuracy with only 0.6 million parameters. Overall, this study highlights key design considerations for building efficient CNN architectures and provides empirical evidence supporting the use of small convolutional filters, progressive depth-wise channel expansion, dropout regularization, and data augmentation. Future work will focus on evaluating the model on more complex and high-resolution datasets, such as ImageNet, Tiny ImageNet, or CIFAR-100, including those with class imbalance or greater intra-class variation, to test generalization. Additionally, the impact of alternative optimization algorithms such as AdaDelta, Adam, and AdaGrad can be assessed, and the proposed architecture can be benchmarked against current state-of-the-art models in challenging image classification scenarios.

Conflicts of Interest Statement

The Authors declare that they have no financial interests or personal relationships that could have influenced the research presented in this paper.

Data Availability Statement

The data supporting the findings of this study are presented in the manuscript. Additional data will be provided upon request.

References

1. Shiri, F. M., T. Perumal, N. Mustapha, and R. Mohamed. "A Comprehensive Overview and Comparative Analysis on Deep Learning Models." *Journal of Artificial Intelligence*, vol. 6, no. 1, 2024, pp. 301–360.
2. Krichen, Mohamed. "Convolutional Neural Networks: A Survey." *Computers*, vol. 12, no. 8, 2023, p. 151, doi:10.3390/computers12080151.
3. Krichen, Mohamed. "Fully Connected Layer." *Convolutional Neural Networks: A Survey*, *Computers*, vol. 12, no. 8, 2023.
4. Cagnetta, Francesco, et al. "How Deep Neural Networks Learn Compositional Data: The Random Hierarchy Model." *Physical Review Research*, vol. 14, no. 3, 2023, p. 031001.
5. Zilliz. "What Is a Convolutional Neural Network? An Engineer's Guide." *Zilliz Glossary*, 2025, zilliz.com/glossary/convolutional-neural-network. Accessed 3 June 2025.
6. emangope."CIFAR-10 Dataset." *Kaggle*, 2025, www.kaggle.com/datasets/emangope/cifar-10. Accessed 8 June 2025.
7. Habib, Ghulam, et al. "GAPCNN with HyPar: Global Average Pooling CNN for Image Classification." *IEEE Access*, vol. 10, 2022, pp. 125462–125474.
8. Ramachandran, S., P. K. S. Kumar, and B. H. K. "Activation Functions in Deep Learning." *arXiv*, arXiv:2109.14545, 2022.
9. Latif, Ghulam, et al. "Deep Convolutional Neural Network Model Optimization Techniques—A Review for Medical Imaging." *AIMS Mathematics*, vol. 9, no. 8, 2024, pp. 20539–20571, doi:10.3934/math.2024998.
10. Tian, Y., Y. Zhang, and H. Zhang. "Recent Advances in Stochastic Gradient Descent in Deep Learning." *Mathematics*, vol. 11, no. 3, 2023, p. 682, doi:10.3390/math11030682.
11. Keskar, Nitish Shirish, and Richard Socher. "Improving Generalization Performance by Switching from Adam to SGD." *arXiv*, arXiv:1712.07628, 2017.
12. Jelassi, Samy, and Yu Li. "Towards Understanding How Momentum Improves Generalization in Deep Learning." *Proceedings of the International Conference on Machine Learning (ICML)*, PMLR, 2022.
13. Dishar, H. K., and L. A. Muhammed. "A Review of the Overfitting Problem in Convolutional Neural Networks and Remedy Approaches." *Journal of Al-Qadisiyah for Computer Science and Mathematics*, vol. 15, no. 2, 2023, doi:10.29304/jqcm.2023.15.2.1240.
14. Harris, Oliver, and Mark Andrews. "Effects of Dropout Layers in Reducing Overfitting in Convolutional Neural Networks." *World Journal of Advanced Engineering Technology and Sciences*, vol. 13, no. 2, 2024, pp. 836–853, doi:10.30574/wjaets.2024.13.2.0584.
15. "Confusion Matrix." *Wikipedia*, 20 July 2025, en.wikipedia.org/wiki/Confusion_matrix. Accessed 20 July 2025.
16. PyTorch. "ResNet-18." *Torchvision Models Documentation*, 2025, docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html. Accessed 21 July 2025.

17. PyTorch. "VGG-16." *Torchvision Models Documentation*, 2025, docs.pytorch.org/vision/main/models/generated/torchvision.models.vgg16.html. Accessed 21 July 2025.
18. Liu, X., and K. M. Goh. "ResNet: Enabling Deep Convolutional Neural Networks through Residual Learning." *arXiv*, 2025. Accessed 21 July 2025.
19. Shin, S., et al. "Implementation and Comparison of CNN Models on CIFAR-10 Dataset." *Proceedings of the IEOM World Congress*, Detroit, USA, 2024, pp. 1–8, doi:10.46254/wc01.20240168.
20. Li, Tao. "A Multi-Scale Feature Extraction and Hierarchical Discriminant Analysis Approach for Image Recognition." *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 49, no. 22, 2025.
21. Zhao, L., et al. "A Lightweight Deep Neural Network with Higher Accuracy." *PLoS ONE*, vol. 17, no. 8, 2022, p. e0271225, doi:10.1371/journal.pone.0271225.
22. Xian, Z., et al. "CNN-Based Image Classification Using Different Color Spaces." *Tsinghua Science and Technology*, 2025.
23. Poojary, R., and A. Pai. "Comparative Study of Model Optimization Techniques in Fine-Tuned CNN Models." *Proceedings of the International Conference on Electrical, Computing Technologies and Applications (ICECTA)*, 2019, pp. 1–4.
24. Abdullah, A., and M. S. Hasan. "An Application of Pre-Trained CNN for Image Classification." *Proceedings of the 20th International Conference on Computer and Information Technology (ICCIT)*, 2017, pp. 1–6.
25. Katpally, H., and A. Bansal. "Ensemble Learning on Deep Neural Networks for Image Caption Generation." *Proceedings of the IEEE 14th International Conference on Semantic Computing (ICSC)*, 2020, pp. 61–68.
26. Tang, S., and Y. Yuan. *Object Detection Based on Convolutional Neural Networks*. Stanford University, Technical Report.
27. Girshick, R., et al. *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*. University of California, Berkeley, Technical Report.
28. Yang, X., S. Yu, and W. Xu. "Enhanced Convolutional Neural Networks for Improved Image Classification." *arXiv*, 2025.
29. Li, B., et al. "On the Optimization and Generalization of Two-Class Deep Models." *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.
30. Simonyan, K., and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.
31. HojjatK. "MNIST Dataset." *Kaggle*, 2025, www.kaggle.com/datasets/hojjatk/mnist-dataset. Accessed 8 June 2025.