Alohomora!
Computer Vision (RBE549) Homework 0

Prasham Soni
Masters of Science
Robotics Engineering
Worcester Polytechnic Institute
Worcester , Massachusetts 01609
Email: psoni@wpi.edu

***Abstract*—This report presents a two-phase approach to solving boundary detection and image classification problems using classical computer vision and deep learning techniques. In Phase 1, we develop a simplified probability of boundary (Pb-lite) algorithm that leverages texture, brightness, and color gradients derived from custom filter banks, including Oriented Derivative of Gaussian (DoG), Leung-Malik (LM), and Gabor filters. This approach significantly improves edge detection compared to classical Sobel and Canny baselines.**

In Phase 2, we implement and evaluate neural network architectures for image classification using the CIFAR-10 dataset. The phase involves training a baseline convolutional neural network (CNN), applying performance-enhancing techniques such as data standardization, learning rate decay, batch normalization, and data augmentation, and exploring advanced architectures like ResNet, ResNeXt, and DenseNet. Metrics including training and testing accuracy, model parameters, loss curves, and confusion matrices are analyzed to compare the architectures.

The study combines classical computer vision techniques with modern deep learning methodologies to provide insights into feature extraction, optimization, and architectural performance, offering a comprehensive solution to edge detection and image classification challenges.

I. Introduction

Boundary detection and image classification are two essential tasks in computer vision, with applications in areas like object recognition, scene analysis, and autonomous navigation. Classical edge detection methods, such as Sobel and Canny, work well for identifying sharp changes in intensity, but they often struggle with identifying meaningful boundaries in regions with complex textures or colors. To address these challenges, modern techniques like the probability of boundary (Pb) algorithm incorporate additional information, such as texture, brightness, and color, to better distinguish boundaries. In the first phase of this project, we implement a simplified version of the Pb algorithm, known as Pb-lite, using customized filter banks, clustering methods, and gradient calculations to detect boundaries more effectively.

In the second phase, we shift focus to image classification, leveraging the power of deep learning. Using the CIFAR-10 dataset, we train and evaluate convolutional neural networks (CNNs), starting with a baseline model and then exploring more advanced architectures such as ResNet, ResNeXt, and DenseNet. We also experiment with performance-enhancing techniques like data augmentation, batch normalization, and learning rate decay to improve accuracy and efficiency. By combining classical computer vision methods with deep learning, this study provides insights into the strengths and limitations of different approaches across two critical computer vision tasks.

II. Phase 1: Shake My Boundary

In Phase 1 we dive deep into the classical computer vision techniques for boundary detection wherein we do a comprehensive analysis of the probability of Boundary (Pb) Algorithm by developing a simplified version called " Pb-Lite" This approach enhances boundary detection by leveraging the texture (T), brightness (B) and color (C) information, achieving improved results over Sobel and Canny baseline. This approach is divided into four main stages: (*Fig. 1*)

1). Filter Banks Generation
2). Generating Texton, Brightness and Color maps
3). Calculating Texton, Brightness and Color Gradients
4). Detecting Boundaries

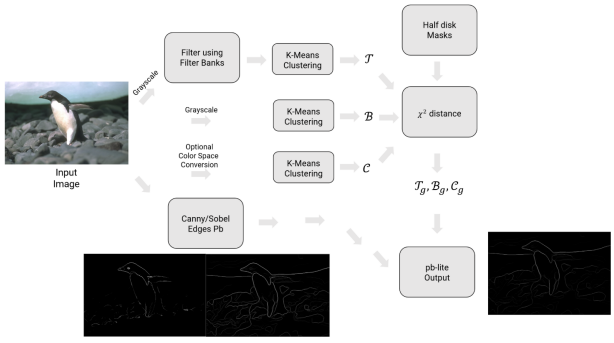

Fig. 1. Overview of Pb-Lite Pipeline

A. Generating Filter Banks

The first step of generating a Probability of Boundary lite (Pb-Lite) filter is to generate filter banks, which are necessary to extract low-level image features that are necessary for boundary detection. Filters are designed to respond to patterns, textures, or intensity variations in an image. In this implementation, we use three types of filter:
1. Oriented Derivative of Gaussian (DoG) Filters
2. Leung-Malik Filters
3. Gabor Filters

1) Oriented DoG (Derivative of Gaussian) Filter : A Simple DoG Filter Bank (Fig. 2.) with DoG Filters is created with the following parameters:
Kernel size = 15
sigmas =[1, 2]
number of orientations = 8

We begin by creating Gaussian Kernels for Smoothing, followed by applying Sobel filters to compute intensity gradients in the x and y directions. These gradients are then rotated to capture edge information at various angles. The process is repeated across different scales, defined by the parameters given above, thus resulting in a filter bank. This approach enables robust feature extraction by capturing texture, brightness, and boundary variations in an image, making it particularly useful for algorithms like Pb-lite in classical computer vision.

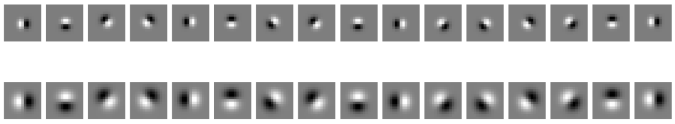

Fig. 2. Oriented Derivative of Gaussian (DoG) Filter Bank

2) Leung-Malik Filters: The LM Filter bank is created with following parameters:
kernel size = 49
number of orientations = 6
Versions:

- LM Small (LMS): $\sigma = \{1, \sqrt{2}, 2, 2\sqrt{2}\}$
- LM Large (LML): $\sigma = \{\sqrt{2}, 2, 2\sqrt{2}, 4\}$

We begin by generating Gaussian kernels, which form the foundation for all filters in the Leung-Malik filter bank. First and second derivatives of the Gaussian are calculated to capture edges, gradients, and curvature in the image. These filters are then rotated systematically to detect directional features at various angles. Additionally, Laplacian of Gaussian (LoG) filters are created to identify blob-like structures, while Gaussian filters are included for baseline smoothing. This process is performed at multiple scales, depending on the LMS (Small Version) or LML (Large Version) version, resulting in a comprehensive filter bank. Both versions are depicted in the figures below (Fig.3.) ,(Fig.4.)This approach allows for robust texture and boundary detection, making it highly effective for tasks like texture segmentation and feature extraction in classical computer vision.

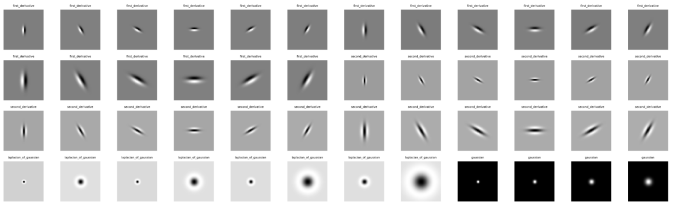

Fig. 3. LM Small (LMS) Filter Bank

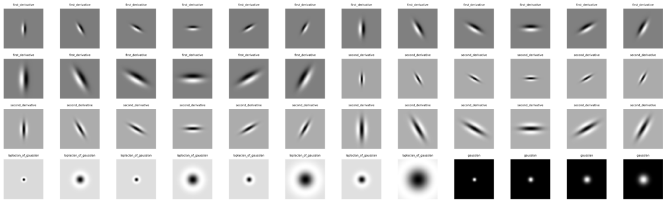

Fig. 4. LM Large (LML) Filter Bank

3) Gabor Filters: These filters are designed based on the operation of human eye. As mentioned in the problem statement. ". A gabor filter is a gaussian kernel function modulated by a sinusoidal plane wave " Gabor Filter Bank is created with the following parameters:

- Kernel Size: 33
- Sigma (σ): 8
- Gamma (γ): 1.4
- Orientations (θ): 0, $\pi/8$, $\pi/4$, $3\pi/8$, $\pi/2$, $5\pi/8$, $3\pi/4$, $7\pi/8$
- Wavelengths (λ): 5, 10, 15, 20
- Phase (ϕ): 0

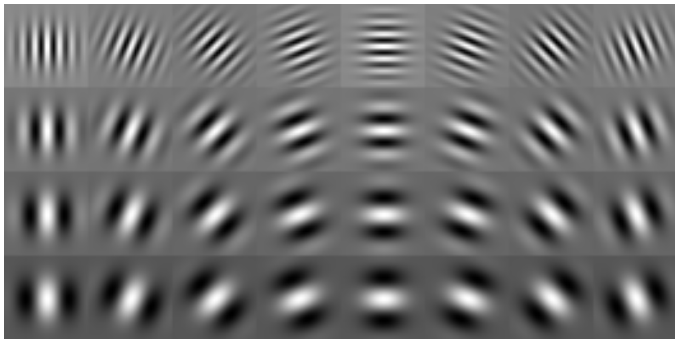

Fig. 5. Gabor Filter Bank

B. Generating Texton, Brightness and Color Maps

1) Texton Map (T): To create the Texton map the input image is processed using a combined filter bank (Dog , Gabor and LM filter , total 112 filters) producing texture specific responses for each pixel, the filter responses are stacked into a high dimensional vector after which k means clustering is

applied to these vectors which results in a single channel map where each pixel represents a ” texton” (Texture cluster)

2) Brightness Map (B): To create Brightness map the image is converted to gray scale to extract intensity information. after which Pixel intensity values are flattened and clustered using K-Means. thus giving us a A single-channel map where each pixel value represents a brightness cluster.

3) Color Map (C): To create the color map, the image is transformed into a perceptually uniform color space, pixel color values are flattened and clustered using K-Means thus giving us a A single-channel map where each pixel value represents a color cluster The generated Texton Maps T , Brightness Maps B and Color Maps C for all the 10 provided images is shown in Fig. 7.

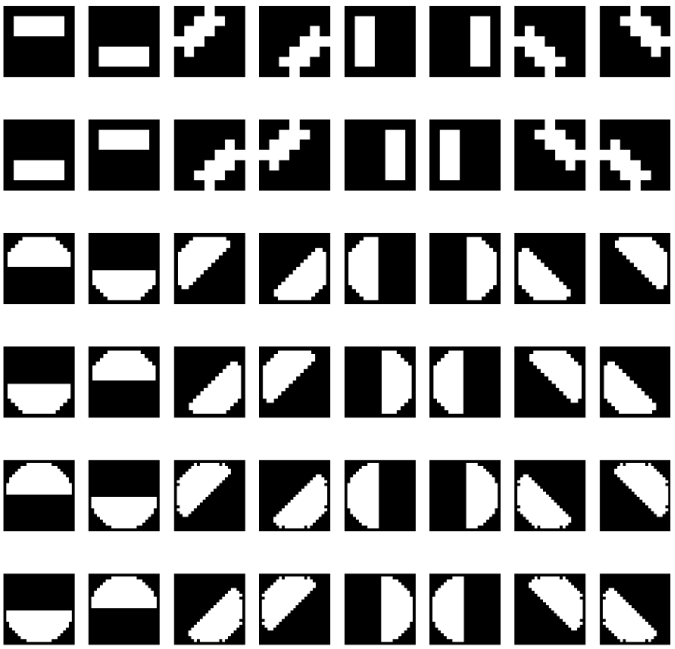

Fig. 6. Half-Disk Mask FilterBank

C. Calculating Texton, Brightness and Color Gradients

To compute the Texton, Brightness, and Color Gradients from their respective maps, the first step is to create half-disc masks (Fig. 6). These masks, which are pairs of binary images representing half-disc shapes, are used to calculate the χ^2 distances via a simple filtering operation. This approach significantly simplifies the process, eliminating the need to aggregate histogram counts by looping over the entire pixel neighborhood.

The gradients T_g, B_g, and C_g encode the rate of change in texture, brightness, and color distributions at each pixel. These gradients are computed by comparing the distributions in left/right half-disc pairs (opposing directions of filters at the same scale). As shown in Fig. 6, these pairs are easily created by controlling the angle of the masks. If the distributions in the half-disc regions are similar, the gradient value will be small. Conversely, if the distributions are dissimilar,

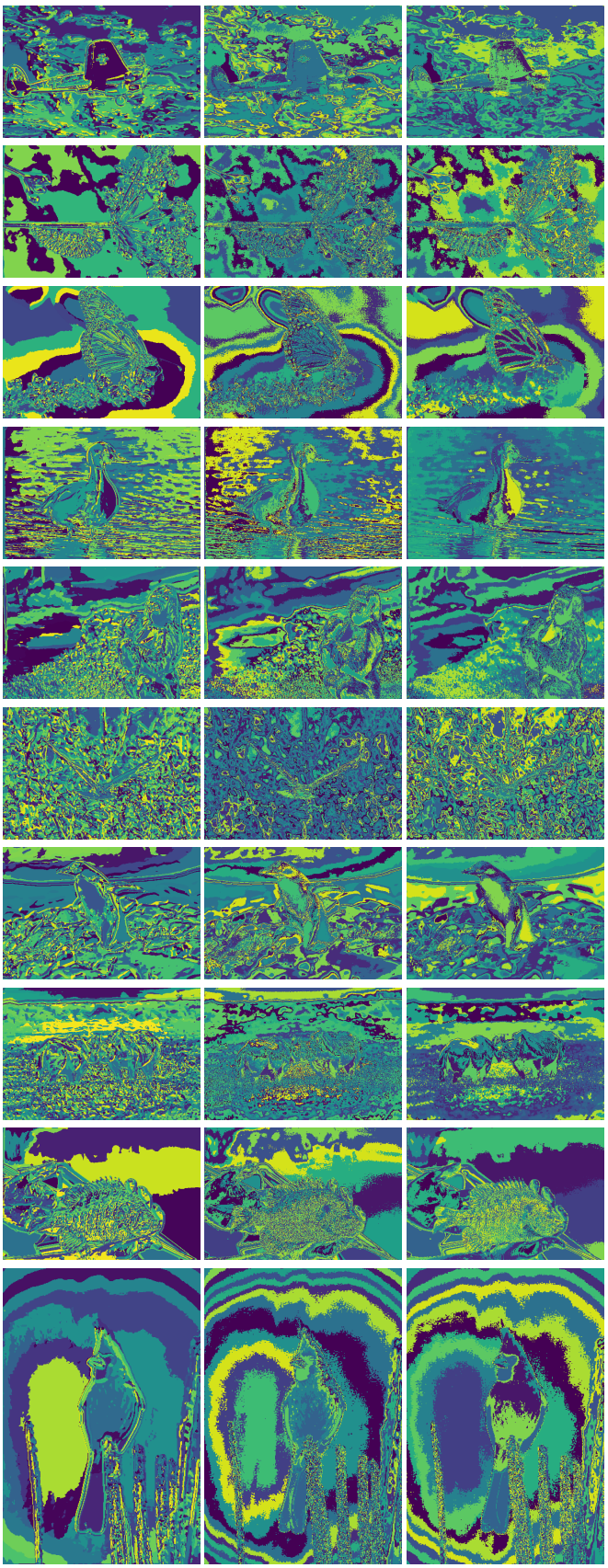

Fig. 7. Texton Maps, Brightness Maps, and Color maps.

the gradient value will be large. Applying half-disc masks at multiple scales and orientations produces local gradient measurements, capturing changes in texture, brightness, and color distributions across scales and directions.

The χ^2 distance quantifies the difference between histograms in opposing regions, making it a robust metric for identifying gradients. It is a widely adopted method for comparing two histograms and is defined as:

$$\chi^2(g,h) = \frac{1}{2}\sum_{i=1}^{K}\frac{(g_i - h_i)^2}{g_i + h_i}$$

Here, K represents the number of bins in the histograms.

Using this method, we compute the Texton Gradient Map (T_g), Brightness Gradient Map (B_g), and Color Gradient Map (C_g), which capture the rate of change in texture, brightness, and color distributions across the image. These gradient maps (T_g, B_g, and C_g) highlight areas of significant variation, making them crucial for boundary detection. The gradient maps for the input images are shown in Fig.8 .

D. Detecting Boundaries

The final step in the boundary detection pipeline involves combining the Texton, Brightness, and Color Gradients (T_g, B_g, and C_g) with the Sobel and Canny baseline edge detections to produce the final boundary map. This combination enhances the accuracy of boundary detection by leveraging both the gradient features and the baseline edge detection outputs. The process effectively highlights significant boundaries while suppressing noise and false positives. The combination is performed using the following equation as defined in the problem statment:

$$PbEdges = \frac{(T_g + B_g + C_g)}{3} \odot (w_1 \cdot \text{cannyPb} + w_2 \cdot \text{sobelPb})$$

where $\odot$ represents the Hadamard product operator. This approach integrates feature-based gradients with baseline edge detectors, resulting in enhanced boundary detection.

The generated Pb-Lite boundaries and the corresponding canny and Sobel baseline for all 10 input images are shown in Fig. 9

E. Analysis

Upon comparing the Pb-Lite output (Fig .9.) with the Canny and Sobel baselines, it is evident that Pb-Lite successfully detects many boundaries overlooked by the Sobel baseline while avoiding the false boundaries often detected by the Canny baseline.

The Pb-Lite output presented in this work was generated using the DoG filter, LM (Small) filter, and Gabor filter, all of which were implemented and described in the earlier sections of this paper. These filters play a crucial role in enhancing the boundary detection capabilities of Pb-Lite. Therefore, it can be concluded that Pb-Lite provides superior boundary detection compared to both Sobel baseline and Canny baselines.

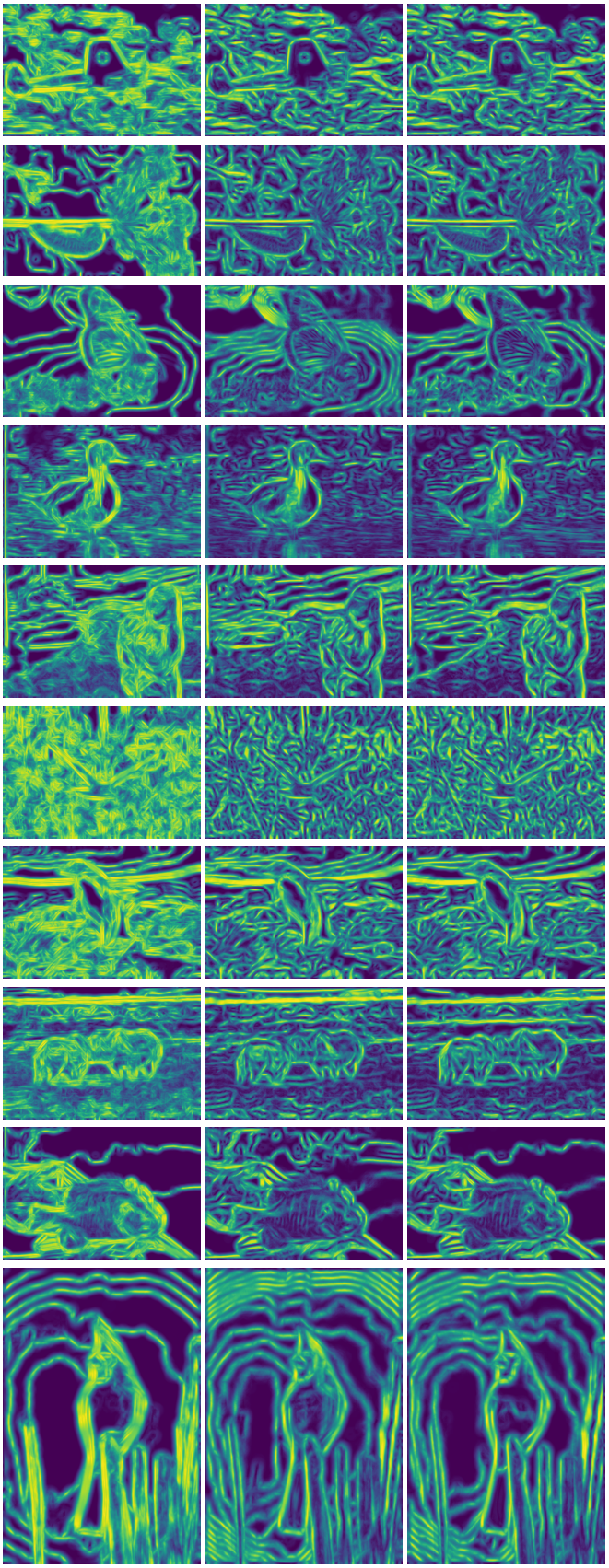

Fig. 8. Texton Gradient Maps, Brightness Gradient Maps, and Color Gradient Maps.

Fig. 9. Canny Baselines, Sobel Baselines, and Pb-Lite Outputs for the Input Images.

III. Phase II: Deep Dive on Deep Learning

A. Overview

In this phase, we implemented several neural networks to perform classification on the CIFAR-10 image dataset. The CIFAR-10 dataset comprises 60,000 images, split into 50,000 for training and 10,000 for testing. Each image is 32×32 pixels and belongs to one of the following 10 classes:

['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'].

We trained a total of five neural networks:

1) LeNet
2) CIFAR10Model
3) ResNet
4) DenseNetCIFAR10
5) ResNeXT

LeNet Model Architecture

As a starting point, we implemented a simple LeNet model for image classification. LeNet is a lightweight neural network consisting of 11 layers, including three convolutional layers, two average pooling layers, and two fully connected (linear) layers. Tanh activations are used throughout to introduce non-linearity.

The workflow of the LeNet model is as follows:

- The input is processed through a convolutional layer with six filters of size 5×5, followed by Tanh activation and an average pooling layer to reduce spatial dimensions.
- This is followed by a second convolutional layer with 16 filters and a third with 120 filters, each reducing the spatial dimensions and learning more complex features.
- The feature maps are flattened and passed through a fully connected layer with 48 neurons activated by Tanh.
- Finally, the output layer generates the class predictions.

This model progressively reduces image dimensions while extracting increasingly complex features, making it a straightforward and effective architecture for classification tasks.

Training Details

The model was trained with the following parameters:

- **Total Number of Parameters:** 57,290
- **Number of Epochs:** 25
- **Mini-Batch Size:** 128

Results and Analysis

The performance of the LeNet model is illustrated in the following figures:

- **Figure 1:** Confusion Matrix
- **Figure 2:** Train and Test Loss vs. Epochs
- **Figure 3:** Train and Test Accuracy vs. Epochs

From the confusion matrix (Fig. 10), we observe that the model achieved an accuracy of **59.99%**, which is relatively low. The train and test loss curves (Fig. 11) and accuracy curves (Fig. 12) indicate that the model faced limitations in learning complex features from the CIFAR-10 dataset.

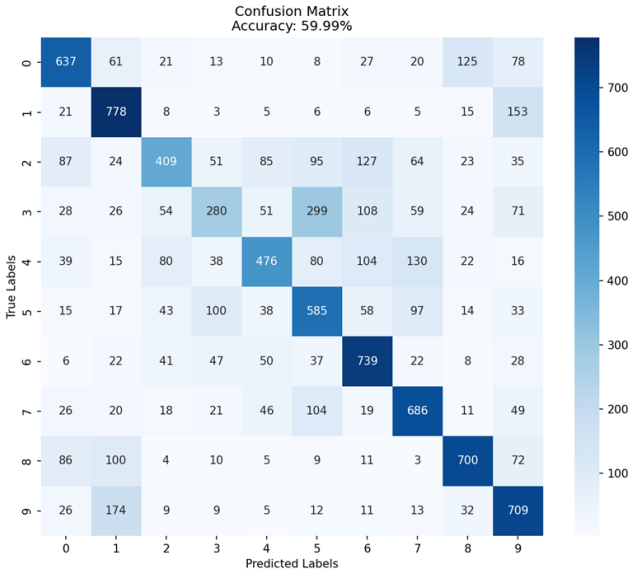

Fig. 10. LeNet: Confusion Matrix

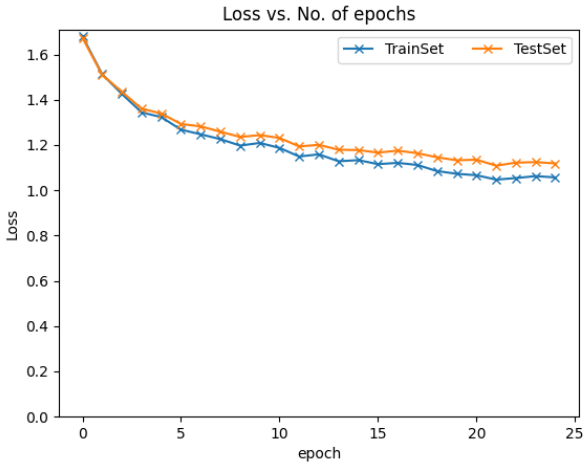

Fig. 11. LeNet: Train and Test Loss over Epochs

Due to this performance, we decided to experiment with deeper and more advanced models to achieve better classification results.

CIFAR10 Model Development and Results

After implementing the basic **LeNet** model, we aimed to achieve higher accuracy by increasing the model's depth and complexity. This led to the development of the **CIFAR10Model**, a custom convolutional neural network specifically designed for the CIFAR-10 dataset.

CIFAR10Model Architecture

The **CIFAR10Model** consists of 18 layers, including five convolutional layers, three max-pooling layers, and three fully connected layers. It also incorporates batch normalization and

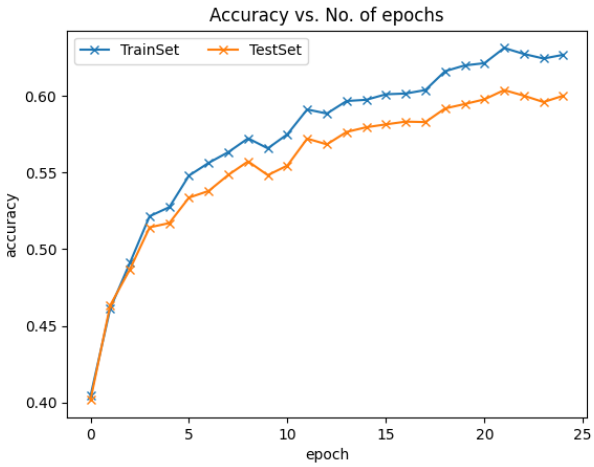

Fig. 12. LeNet: Train and Test Accuracy over Epochs

ReLU activations to enhance training efficiency and overall performance.

The architecture follows a structured progression:

- The first convolutional layer applies 16 filters of size 3×3, followed by batch normalization, ReLU activation, and another convolutional layer with 32 filters. A max-pooling layer then reduces the spatial dimensions.
- This process is repeated with increasing filter sizes: 64 filters in the third convolutional layer, 128 filters in the fourth, and 256 filters in the fifth, each followed by batch normalization and ReLU activations.
- After the final max-pooling layer, the output is flattened and passed through fully connected layers with 512 and 128 neurons, respectively, before reaching the final output layer, which produces class probabilities for the 10 CIFAR-10 categories.

This architecture progressively extracts meaningful features while reducing spatial dimensions, making it well-suited for accurate image classification.

Visual Results and Analysis

The following figures illustrate the performance of the CIFAR10Model:

Key Improvements Over LeNet

The CIFAR10Model introduces several enhancements compared to the LeNet architecture:

- **Deeper Architecture:** A more complex structure enables the model to learn richer and more detailed features from the dataset.
- **Batch Normalization:** Improves training stability and speeds up convergence by normalizing layer inputs.
- **ReLU Activation:** Replacing Tanh with ReLU provides better non-linearity, avoids vanishing gradient issues, and accelerates training.
- **Dropout Layers:** Reduces overfitting by randomly deactivating neurons during training.

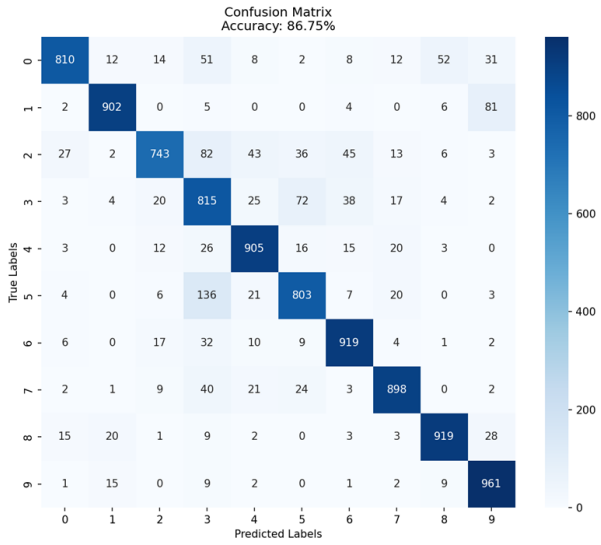

Fig. 13. CIFAR10Model: Confusion Matrix

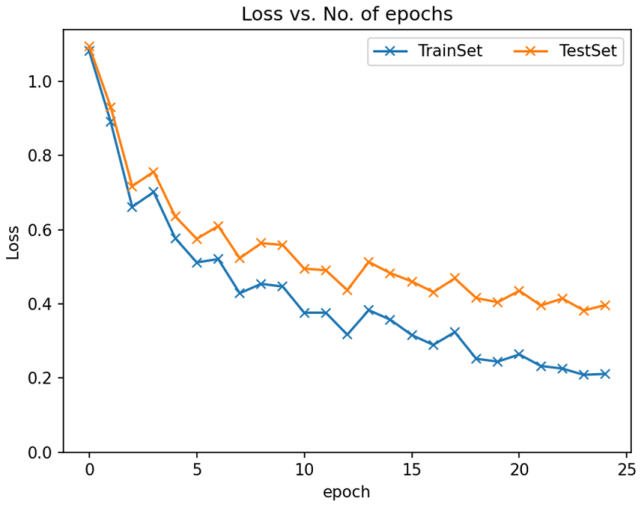

Fig. 14. CIFAR10Model: Training and Testing Loss Over Epochs

- **Data Augmentation:** Techniques such as random cropping, horizontal flipping, and normalization increase the diversity of the training data, improving generalization.

Training Details

The CIFAR10Model was trained using the following parameters:

- **Optimizer:** Adam optimizer with an initial learning rate of 0.001.
- **Loss Function:** Cross-entropy loss for multi-class classification.
- **Number of Epochs:** 50.
- **Mini-Batch Size:** 128.
- **Learning Rate Schedule:** The learning rate was reduced by a factor of 0.1 after 20 epochs to fine-tune the model.

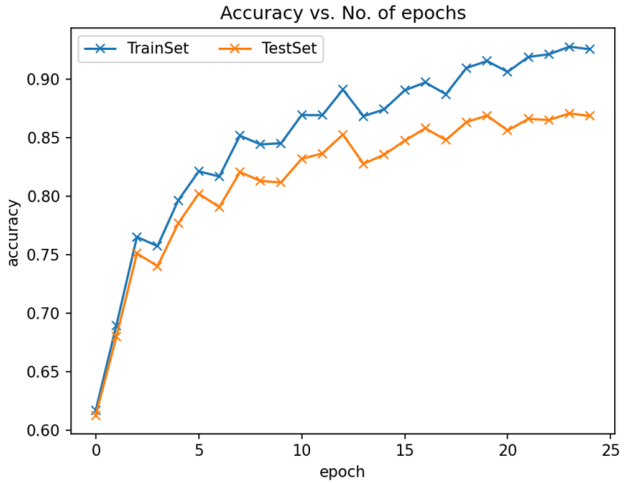

Fig. 15. CIFAR10Model: Training and Testing Accuracy Over Epochs

Results

The final accuracy achieved by the CIFAR10Model was **86.75%**, a significant improvement over the previous **LeNet model's accuracy of 75.3%**. This result demonstrates the effectiveness of using a deeper architecture, advanced optimization techniques, and proper regularization strategies for more complex datasets like CIFAR-10.

ResNet Model Implementation

After successfully implementing the **CIFAR10Model**, we were tasked with implementing a **ResNet** model. ResNet, or Residual Network, is a deep learning architecture designed to address the vanishing gradient problem by introducing skip connections (or residual connections). These connections allow the model to learn identity mappings, making it easier to optimize deeper networks.

Model Architecture

The implemented ResNet model comprises several building blocks:

- **Convolutional Block 1:**
  - A convolutional layer with 64 filters, kernel size 3×3, and padding of 1.
  - Followed by batch normalization and ReLU activation.
  - A second convolutional layer with 128 filters, kernel size 3×3, and padding of 1.
  - Followed by batch normalization, ReLU activation, and max pooling.
- **Residual Block 1:**
  - Two convolutional layers, each with 128 filters and kernel size 3×3.
  - Both layers include batch normalization and the first is followed by ReLU activation.
  - The input is added to the output of this block (skip connection), followed by ReLU activation.

- **Convolutional Block 2:**
  - A convolutional layer with 256 filters, kernel size 3× 3, and padding of 1, followed by batch normalization and ReLU activation.
  - A max pooling layer.
  - A convolutional layer with 512 filters, kernel size 3× 3, and padding of 1, followed by batch normalization, ReLU activation, and another max pooling layer.
- **Residual Block 2:**
  - Two convolutional layers, each with 512 filters and kernel size 3×3.
  - Both layers include batch normalization and the first is followed by ReLU activation.
  - The input is added to the output of this block (skip connection), followed by ReLU activation.
- **Classifier:**
  - Adaptive average pooling to reduce the spatial dimensions to 1×1.
  - A flattening layer.
  - A dropout layer with a dropout rate of 0.2.
  - A fully connected layer with 512 inputs and the number of outputs equal to the number of classes.

Forward Propagation

The forward propagation follows these steps:

- Input is passed through **Convolutional Block 1**.
- The output is added to the result of **Residual Block 1** via a skip connection, followed by ReLU activation.
- The resulting output is passed through **Convolutional Block 2**.
- The output is added to the result of **Residual Block 2** via a skip connection, followed by ReLU activation.
- Finally, the output is passed through the **Classifier** to produce class predictions.

Key Features of ResNet

- **Skip Connections:** Allow the network to bypass some layers, mitigating the vanishing gradient problem and making it easier to train deeper architectures.
- **Batch Normalization:** Helps stabilize the learning process and speed up convergence.
- **Adaptive Average Pooling:** Reduces the spatial dimensions effectively, allowing the classifier to handle inputs of varying sizes.

The final accuracy achieved on the CIFAR-10 dataset was **89.66%**, showcasing the effectiveness of the ResNet architecture. ResNet provides a robust framework for handling deeper networks and more complex datasets.

DenseNet CIFAR-10 Implementation and Results

We implemented a **DenseNetCIFAR10** model, achieving an accuracy of **85.83%** on the CIFAR-10 dataset. This model leverages the strengths of DenseNet, such as feature reuse and efficient gradient flow, to deliver high performance on image classification tasks.

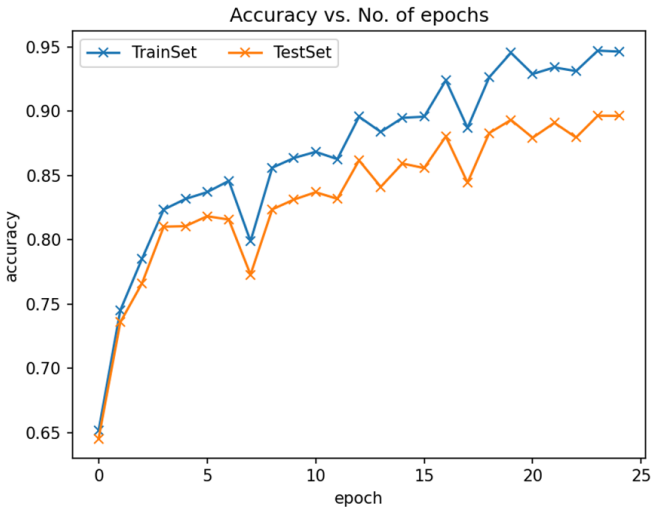

Fig. 16. ResNet: Training and Testing Accuracy Over Epochs

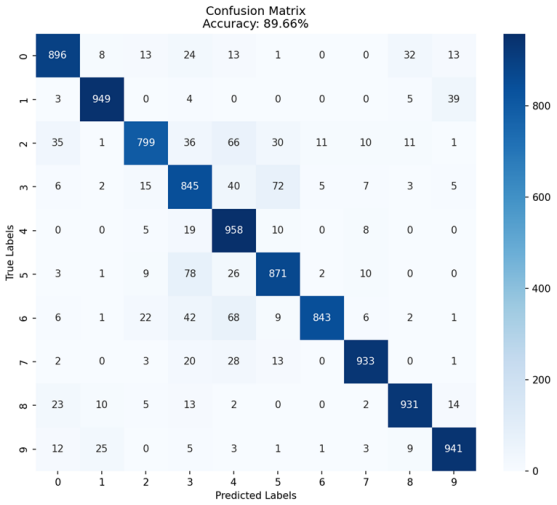

Fig. 17. ResNet: Confusion Matrix

Model Architecture

The DenseNetCIFAR10 model consists of the following key components:

- **Initial Convolution:** A 3×3 convolutional layer is applied to the input, increasing the channel depth to twice the growth rate.
- **Dense Blocks:** The model includes three dense blocks. Each block consists of multiple layers, where each layer outputs feature maps that are concatenated with the input feature maps. This dense connectivity ensures feature reuse and improves parameter efficiency.
- **Transition Layers:** After every dense block (except the last one), a transition layer is used to downsample the feature maps. It includes:
  - Batch normalization.

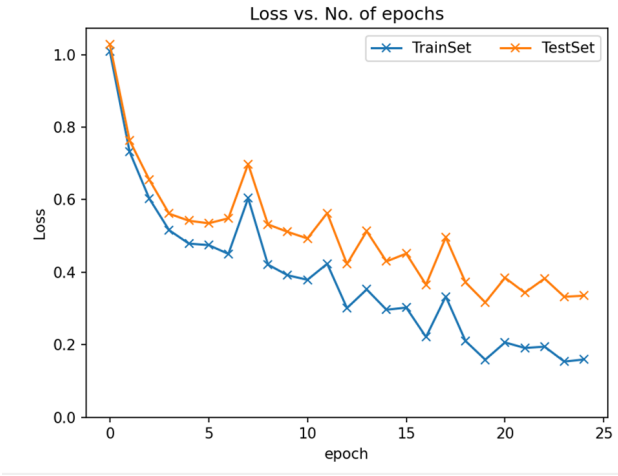

Fig. 18. ResNet: Training and Testing Loss Over Epochs

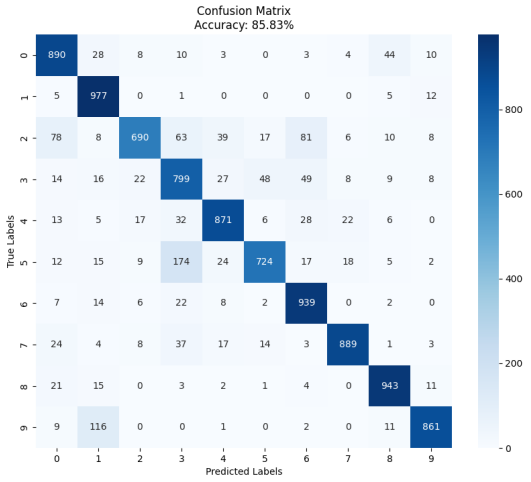

Fig. 19. DenseNetCIFAR10: Confusion Matrix

  - ReLU activation.
  - 1×1 convolution to reduce channel depth.
  - 2×2 average pooling for downsampling.
- **Global Average Pooling and Classifier:** The final feature maps are batch-normalized, followed by global average pooling. The flattened output is passed through a fully connected layer to produce class predictions.

Training Details

The DenseNetCIFAR10 model was trained with the following settings:

- **Growth Rate:** 12
- **Number of Dense Blocks:** 3
- **Layers per Block:** 6
- **Optimizer:** Adam optimizer with a learning rate of 0.001.
- **Loss Function:** Cross-entropy loss.
- **Number of Epochs:** 50.
- **Mini-Batch Size:** 128.
- **Learning Rate Schedule:** Reduced the learning rate by a factor of 0.1 after 20 epochs.
- **Regularization:** Dropout and weight decay (0.0001) were applied to reduce overfitting.

Results

The final accuracy achieved by the DenseNetCIFAR10 model on the CIFAR-10 dataset was **85.83%**. This performance demonstrates the effectiveness of dense connectivity in improving feature reuse and optimizing gradient flow, making the model highly efficient for complex image classification tasks. The model results are depicted in Fig 19. - Fig. 20

ResNeXt Model Implementation for CIFAR-10

We implemented a ResNeXt-based model for the CIFAR-10 dataset and achieved an accuracy of **87.82%**. ResNeXt extends the ResNet architecture by introducing the concept of *cardinality*, which refers to the number of parallel paths in each block.

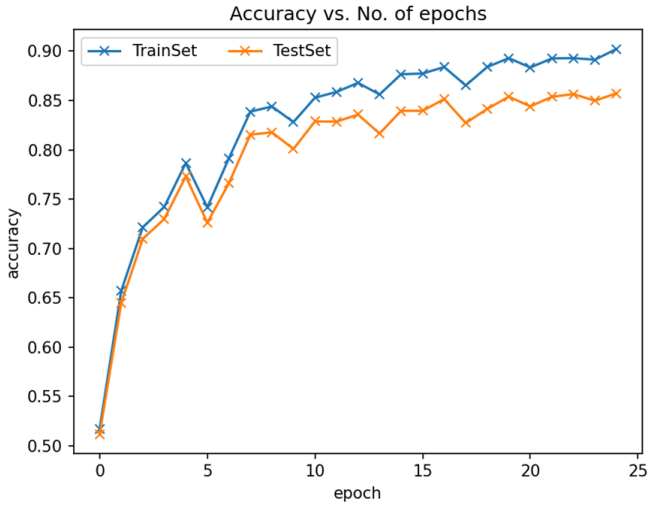

Fig. 20. DenseNetCIFAR10: Training and Testing Accuracy Over Epochs

This approach allows for more efficient feature learning while keeping the model's computational cost manageable.

Model Architecture

The ResNeXt model is composed of the following components:

- **Initial Convolutional Layer:** A convolutional layer with 64 filters, a kernel size of 3×3, and a stride of 1, followed by batch normalization and ReLU activation.
- **ResNeXt Blocks:** The model includes three stages, each comprising multiple ResNeXt blocks. These blocks leverage grouped convolutions to enhance feature extraction capabilities while maintaining efficiency:
  - **Stage 1:** Three ResNeXt blocks, outputting 128 channels while preserving spatial dimensions.
  - **Stage 2:** Three ResNeXt blocks, outputting 256 channels and reducing spatial dimensions by half.

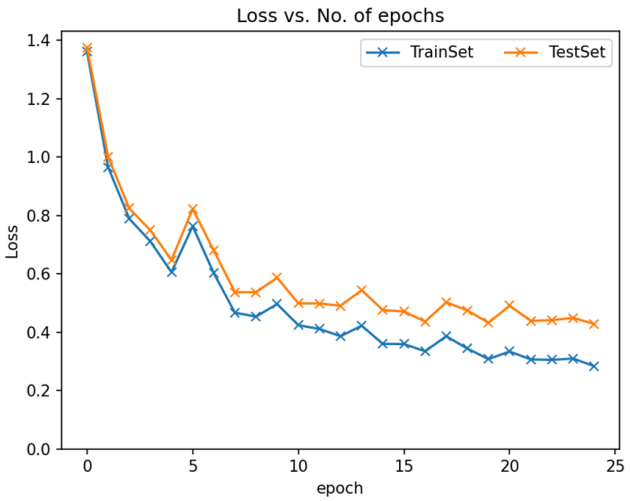

Fig. 21. DenseNetCIFAR10: Training and Testing Loss Over Epochs

  - **Stage 3:** Three ResNeXt blocks, outputting 512 channels and further reducing spatial dimensions by half.
- **Global Average Pooling:** This layer reduces each feature map to a single value by computing the global average.
- **Fully Connected Layer:** A final fully connected layer with 10 outputs corresponds to the 10 classes in the CIFAR-10 dataset.

Details of the ResNeXt Block

Each ResNeXt block consists of:

- A 1×1 convolution for dimensionality reduction.
- A 3×3 grouped convolution, where the number of groups is defined by the cardinality parameter.
- A 1×1 convolution to restore the output dimensionality.
- A shortcut connection for residual learning. If input and output dimensions differ, a 1×1 convolution is applied in the shortcut path.

Training Details

The ResNeXt model was trained using the following configuration:

- **Optimizer:** Adam optimizer with an initial learning rate of 0.001.
- **Loss Function:** Cross-entropy loss for multi-class classification.
- **Number of Epochs:** 50.
- **Mini-Batch Size:** 128.
- **Cardinality:** 8 (number of groups in the grouped convolution).
- **Learning Rate Schedule:** The learning rate was reduced by a factor of 0.1 after 25 epochs.

Results

The ResNeXt model achieved an accuracy of **87.82%** on the CIFAR-10 dataset, demonstrating a significant improvement over simpler architectures such as LeNet. This result highlights the effectiveness of grouped convolutions and residual learning in achieving higher performance for image classification tasks.

Visualization of Results

The following figures depict the performance of the ResNeXt model:

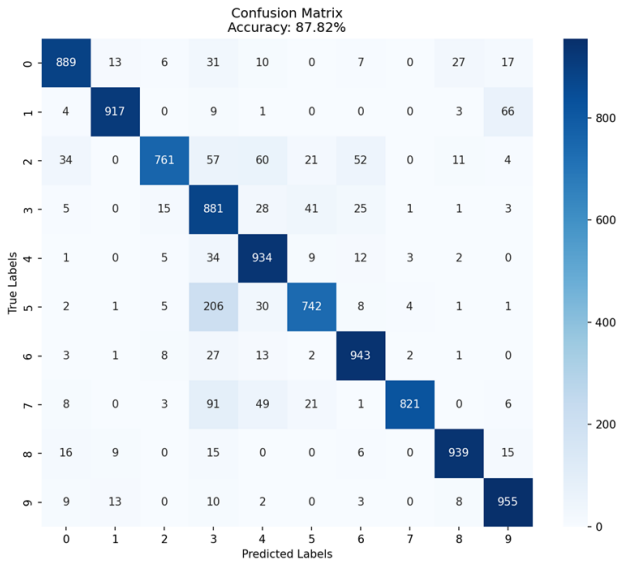

Fig. 22. ResNeXt: Confusion Matrix

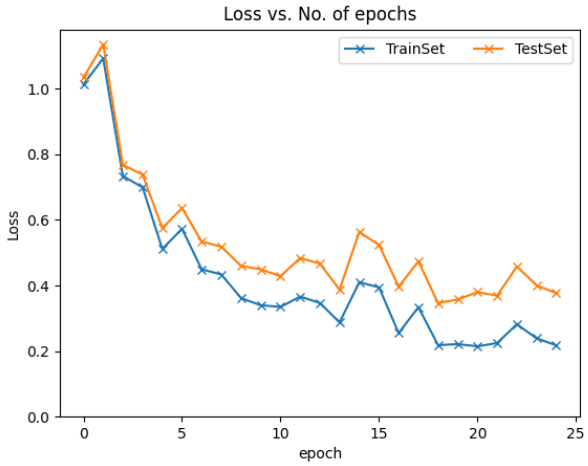

Fig. 23. ResNeXt: Training and Testing Loss Over Epochs

These visualizations provide a clear understanding of the model's performance, including its convergence behavior and classification accuracy.

The table I compares the performance of various models for CIFAR-10 classification based on their accuracy, number of parameters, and training epochs. The LeNet model achieves an accuracy of 59.99% with the smallest number of parameters (57,290), making it the most lightweight model.

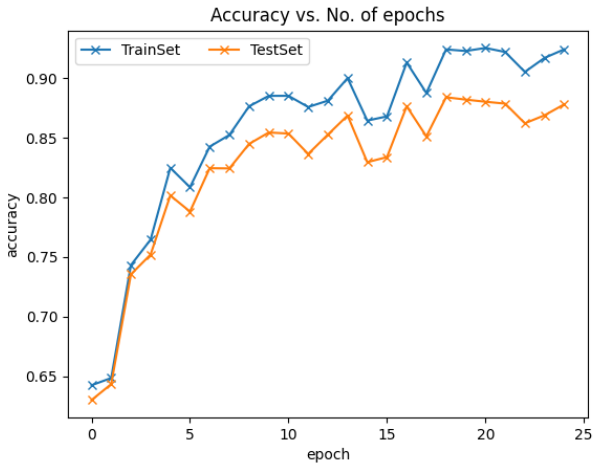

Fig. 24. ResNeXt: Training and Testing Accuracy Over Epochs

TABLE I
COMPARISON OF MODELS FOR CIFAR-10 CLASSIFICATION

Model	**Accuracy (%)**	**Parameters**	**Epochs**
LeNet	59.99	57,290	25
CIFAR10Model	86.75	2,558,218	25
ResNet	89.66	6,575,370	25
DenseNetCIFAR10	85.83	176,122	25
ResNeXt	87.82	3,250,506	25

The custom CIFAR10Model improves significantly, reaching 86.75% accuracy with 2,558,218 parameters. ResNet achieves the highest accuracy of 89.66% but has a larger parameter count of 6,575,370. DenseNetCIFAR10 offers competitive accuracy at 85.83% with only 176,122 parameters, highlighting its efficiency. ResNeXt strikes a balance between accuracy (87.82%) and parameter count (3,250,506). All models were trained for 25 epochs, providing a fair comparison of their performance under the same conditions.

ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to **Dr. Nitin J. Sanket** at **Worcester Polytechnic Institute** for crafting such a fun and challenging learning experience. This project provided an engaging platform to explore and understand fundamental concepts. Additionally, the use of AI tools during this process greatly enhanced my understanding and helped deepen my learning experience, making it both educational and enjoyable.