RBE 549 Computer Vision HW 0 Alohomora

Divam Trivedi
Robotics Engineering
Worcester Polytechnic Institute
Worcester, USA
dtrivedi.wpi.edu

***Abstract*—This paper presents a two-phase approach to enhance boundary detection and neural network-based classification for image processing tasks. In the first phase, we propose an improved boundary detection methodology leveraging Derivative of Gaussians (DoG), Leung-Malik (LM), and Gabor filters combined with half-disk images. These components generate texton maps and gradients, brightness maps and gradients, and color maps and gradients, which are merged with Canny and Sobel baselines to achieve a simplified and refined Pblite boundary. The second phase introduces a comparative study of deep learning architectures on CIFAR-10 dataset, focusing on ResNet, ResNeXt, and DenseNet, alongside the development of a custom neural network to evaluate performance in terms of efficiency and accuracy.**

***Index Terms*—boundary detection, filters, image processing, neural networks, deep learning**

I. Introduction

Boundary detection and neural network classification are key in computer vision, aiding object recognition and scene understanding. Traditional methods balance simplicity with precision, while deep learning offers promising results but requires resources and data.

This paper proposes a two-phase approach integrating traditional image processing with deep learning. The first phase uses DoG, LM, Gabor, and half-disk images to generate texton maps, brightness maps, and color maps, merging them with Canny and Sobel baselines for effective boundary detection.

The second phase employs a custom neural network to evaluate its suitability for boundary detection. This approach bridges the gap between traditional methods and advanced neural networks, offering a comprehensive solution leveraging both paradigms. The methodology, experimental setup, and results demonstrate the efficacy of the approach to improve boundary detection and classification.

II. Phase 1 - Shake My Boundary

In this section, we will develop a simplified version of PbLite using the texture, brightness and color information. To do this, we need to implement different filter banks.

A. Filter bank implementation

The first step of the PbLite boundary detection pipeline is to filter the image with a set of filter banks. We will create three different sets of filter banks for this purpose namely Derivative of Gaussians(DoG), Leung-Malik (LM) and Gabor filters. Once we filter the image with these filters, we'll generate a texton map which depicts the texture in the image by clustering the filter responses.

1) *Oriented Derivative of Gaussian (DoG):* The DoG filter bank captures fine edge details by highlighting regions of rapid intensity changes, providing multi-scale edge detection capabilities essential for robust boundary detection. Shown in fig. 1 is the filter bank produced by sigma values 1 and 2 each having 16 orientations evenly spaced from 0 to 360°.

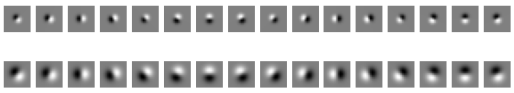

Fig. 1. Oriented Derivative of Gaussian (DoG) Filter Bank

2) *Leung-Malik (LM) Filters:* The LM filters are a set of multi-scale, multi-orientation filter banks with 48 filters. It consists of first and second-order derivatives of Gaussians at 6 orientations and 3 scales, making a total of 36; 8 Laplacian of Gaussian (LoG) filters; and 4 Gaussians. We consider two versions of the LM filter bank. In LM Small (LMS), the filters occur at basic scales $\sigma = [1, \sqrt{2}, 2, 2\sqrt{2}]$ as seen in fig. 2. The first and second derivative filters occur at the first three scales with an elongation factor of 3, i.e., $\sigma_x = \sigma$. The Gaussians occur at the four basic scales, while the 8 LoG filters occur at σ and 3σ. For LM Large (LML), the filters occur at the basic scales $\sigma = [\sqrt{2}, 2, 2\sqrt{2}, 4]$ as shown in fig. 3.

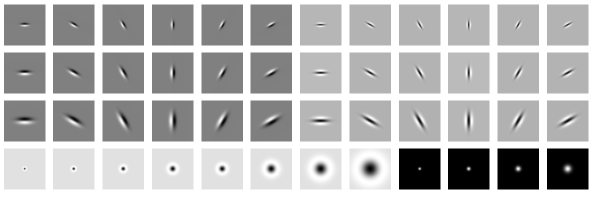

Fig. 2. Leung-Malik Small (LMS) Filter Bank

3) *Gabor Filters:* Gabor filters, characterized by orientation and frequency tuning, efficiently capture spatial frequency and directional information, aiding in the detection of oriented textures and edges within images.

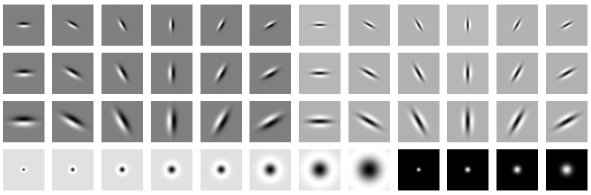

Fig. 3. Leung-Malik Large (LML) Filter Bank

This bank has been implemented using parameters $\lambda = [10, 15, 20]$, $\sigma = [10, 15]$, $\psi = [0.5, 1]$, $\phi = [0, \pi/2]$ as seen in Fig. 4.

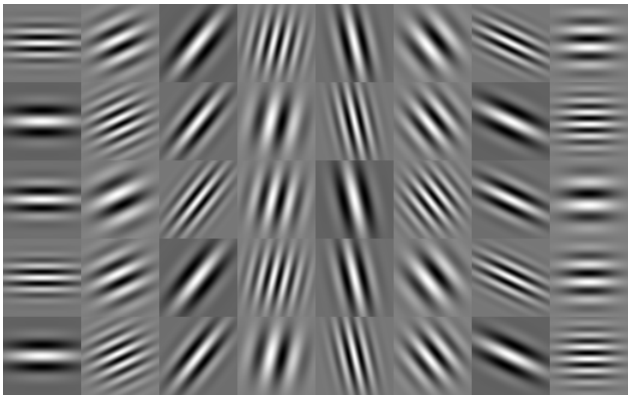

Fig. 4. Gabor Filters

B. Half-Disc Masks

To obtain gradients, we need to compute differences of values across different shapes and sizes. This can be achieved very efficiently by the use of half-disc masks. The half-disc masks are pairs of binary images of half-discs. This will allow us to compute the χ^2 (chi-square) distances using a filtering operation, which is much faster than looping over each pixel neighborhood and aggregating counts for histograms. The set of masks (16 orientations, 3 scales) is shown in Fig. 5.

C. Texton, Brightness and Color Maps

In this section, we develop various image maps based on texture, brightness and color intensities of each pixel across the images. The texton maps are generated by convolving the DoG filter bank with the images and map them into 128 clusters. The concept of the brightness map is as simple as capturing the brightness changes in the image and clustering it into 16 groups. Also, The concept of the color map is to capture the color changes then cluster it into 16 clusters. Illustration of the generated map are shown in figs. 6 to15.

D. Texton, Brightness and Color Gradients

Using the half-disc masks along with the Chi-square distance, we can generate the gradient map for all texture, brightness, and color information of the images. the generated gradients are shown in Fig 15-24. We observe that the brightness gradient is grayscale since we are concerened only with

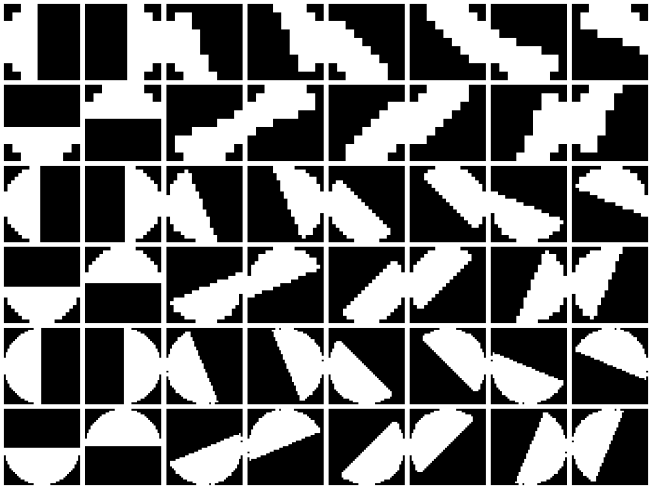

Fig. 5. Half-Disc Masks

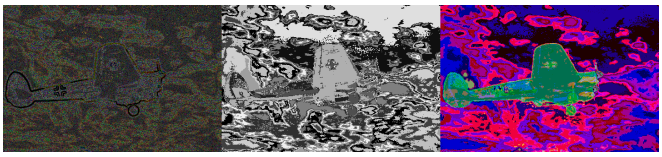

Fig. 6. Texton, Brightness and Color Map for image 1.png

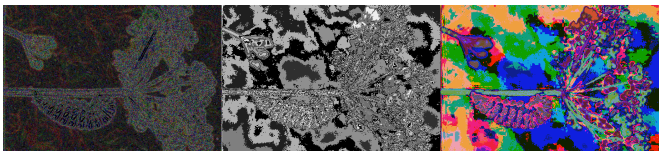

Fig. 7. Texton, Brightness and Color Map for image 2.png

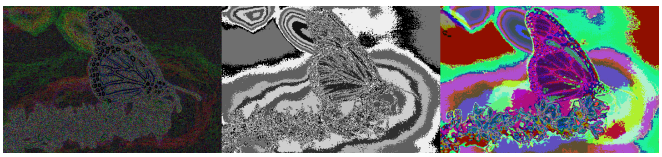

Fig. 8. Texton, Brightness and Color Map for image 3.png

Fig. 9. Texton, Brightness and Color Map for image 4.png

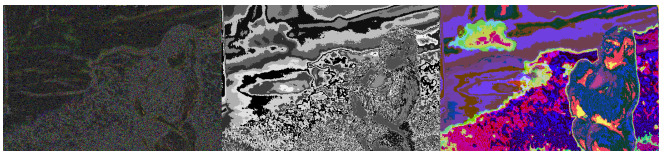

Fig. 10. Texton, Brightness and Color Map for image 5.png

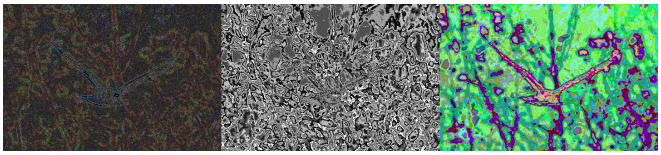

Fig. 11. Texton, Brightness and Color Map for image 6.png

Fig. 12. Texton, Brightness and Color Map for image 7.png

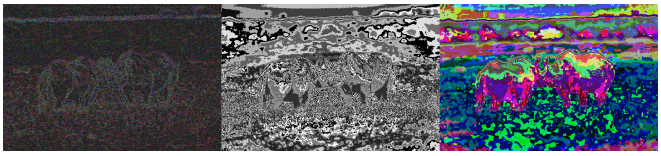

Fig. 13. Texton, Brightness and Color Map for image 8.png

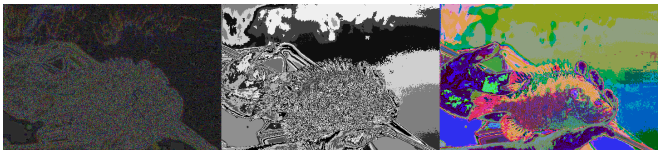

Fig. 14. Texton, Brightness and Color Map for image 9.png

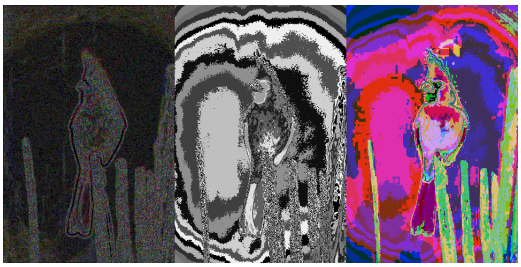

Fig. 15. Texton, Brightness and Color Map for image 10.png

the intensity chnages of pixels. However, in color gradient, we observe an intereting map of colors due to using the "Lab" color space (other color spaces such as "RGB", "YCbCr", or "HSV" could also be used).

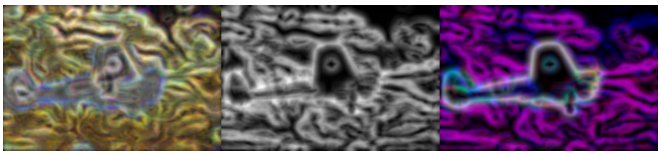

Fig. 16. Texton, Brightness and Color Gradients for image 1.png

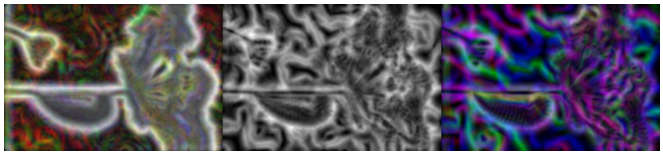

Fig. 17. Texton, Brightness and Color Gradients for image 2.png

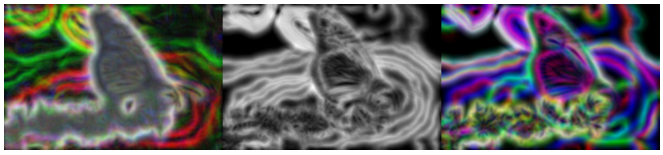

Fig. 18. Texton, Brightness and Color Gradients for image 3.png

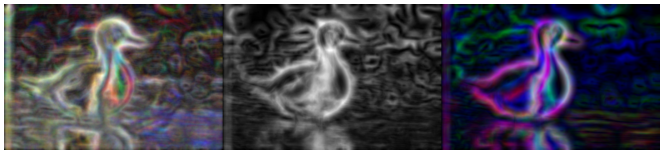

Fig. 19. Texton, Brightness and Color Gradients for image 4.png

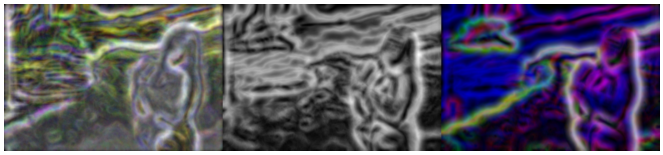

Fig. 20. Texton, Brightness and Color Gradients for image 5.png

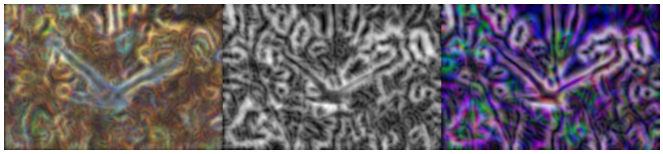

Fig. 21. Texton, Brightness and Color Gradients for image 6.png

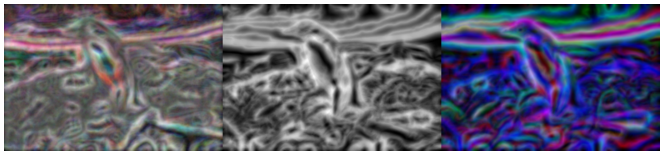

Fig. 22. Texton, Brightness and Color Gradients for image 7.png

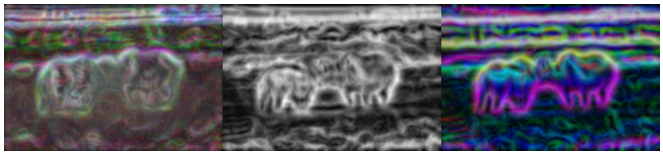

Fig. 23. Texton, Brightness and Color Gradients for image 8.png

E. Boundary Detection

The final step is to combine information from the features with a baseline method (based on weighted average of Sobel

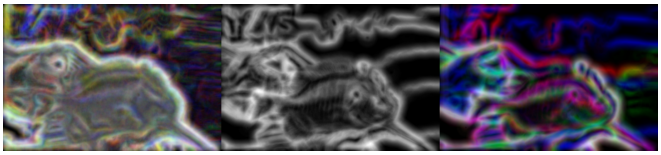

Fig. 24. Texton, Brightness and Color Gradients for image 9.png

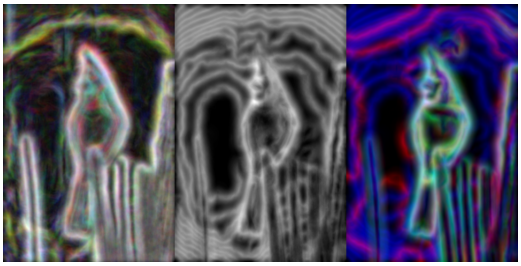

Fig. 25. Texton, Brightness and Color Gradients for image 10.png

and Canny edge detection) using a simple equation as shwown below.

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

A simple choice for w1 and w2 would be 0.5. The results are shown in fig 25-34.

Fig. 26. Sobel, Canny and PbLite Detection Boundaries for image 1.png

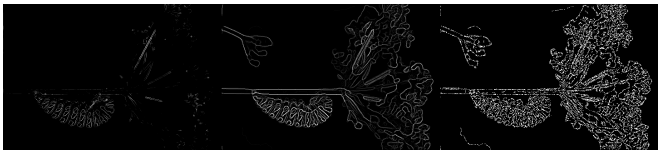

Fig. 27. Sobel, Canny and PbLite Detection Boundaries for image 2.png

Fig. 28. Sobel, Canny and PbLite Detection Boundaries for image 3.png

F. Analysis

The result is not as clear as the Canny baseline, however, it's better than the Sobel base line. In some degree, the PbLite is better than Canny since it does not have false positive results. We can see a lot of incorrect edges in the Canny which are

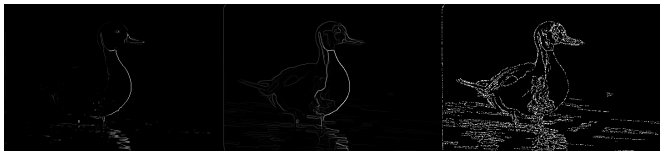

Fig. 29. Sobel, Canny and PbLite Detection Boundaries for image 4.png

Fig. 30. Sobel, Canny and PbLite Detection Boundaries for image 5.png

Fig. 31. Sobel, Canny and PbLite Detection Boundaries for image 6.png

Fig. 32. Sobel, Canny and PbLite Detection Boundaries for image 7.png

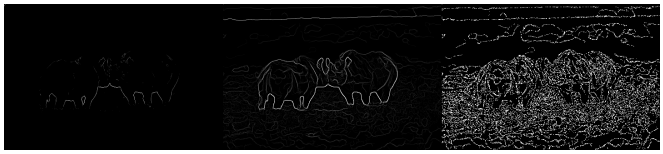

Fig. 33. Sobel, Canny and PbLite Detection Boundaries for image 8.png

Fig. 34. Sobel, Canny and PbLite Detection Boundaries for image 9.png

Fig. 35. Sobel, Canny and PbLite Detection Boundaries for image 10.png

not presented in the PbLite. We also consider the fact that the Texton Maps were generated using only the DoG filters, had it been other filter banks or a combination of them, result might vary. The comparison for Sobel, Canny and PbLite edge detections are as shown in figs. 26 to 35.

III. Phase 2: Deep Dive on Deep Learning

In this section we implement different neural network architectures and try to train them for the image classification. For the image classification a custom shuffled CIFAR-10 dataset with 50000 training images and 10000 test images is used.

A. My first Neural Network

The CIFAR10Model, a Convolutional Neural Network (CNN) [1], is designed for image classification tasks, particularly with datasets like CIFAR-10. It consists of three convolutional layers followed by fully connected layers. The first convolutional layer uses a 3×3 kernel with 32 output channels, padding of 1, and ReLU activation. A 2×2 max pooling layer reduces the spatial dimensions to 16×16×32. The second convolutional layer increases output channels to 64, followed by another 2×2 max pooling layer to reduce dimensions to 8×8×64. The third convolutional layer expands channels to 128, followed by a final 2×2 max pooling layer, resulting in a 4×4×128 tensor. This is flattened into a 2048-dimensional vector and passed through two fully connected layers: a hidden layer with 512 neurons activated by ReLU, and an output layer with 10 neurons pertaining to each class of CIFAR-10.

```
==========================================================
Layer (type:depth-idx)                   Param #
==========================================================
MyModel                                  --
├─Sequential: 1-1                        --
│    └─Conv2d: 2-1                       896
│    └─ReLU: 2-2                         --
│    └─MaxPool2d: 2-3                    --
│    └─Conv2d: 2-4                       18,496
│    └─ReLU: 2-5                         --
│    └─MaxPool2d: 2-6                    --
│    └─Conv2d: 2-7                       73,856
│    └─ReLU: 2-8                         --
│    └─MaxPool2d: 2-9                    --
│    └─Flatten: 2-10                     --
│    └─Linear: 2-11                      1,049,088
│    └─ReLU: 2-12                        --
│    └─Linear: 2-13                      5,130
==========================================================
```

Fig. 36. Simple CNN Architecture

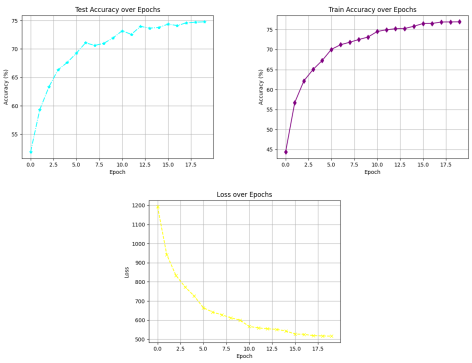

Fig. 37. Simple CNN - (a) Testing Accuracy, (b) Training Accuracy, (c) Cumulative Loss over an epoch

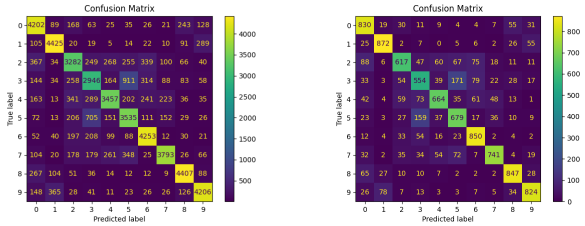

Fig. 38. Simple CNN - (a) Train Confusion Matrix, (b) Test Confusion Matrix

B. Improving the Neural Network

The improved CIFAR10Model2 incorporates enhancements over the baseline model to improve training stability, generalization, and robustness. It includes Batch Normalization after each convolutional layer and the second fully connected layer to normalize activations, accelerate convergence, and stabilize gradients. A dropout layer with a probability of 0.5 is added before the fully connected layers to reduce overfitting by deactivating random neurons during training. These additions address limitations of the baseline model, which lacked normalization and regularization layers, making it less robust to overfitting and internal covariate shifts. Despite a slight increase in parameters due to Batch Normalization, these improvements enable the network to generalize better to unseen data, making it significantly more effective for CIFAR-10 image classification.

```
==========================================================
Layer (type:depth-idx)                   Param #
==========================================================
MyModel                                  --
├─Conv2d: 1-1                            280
├─BatchNorm2d: 1-2                       20
├─MaxPool2d: 1-3                         --
├─Conv2d: 1-4                            1,820
├─BatchNorm2d: 1-5                       40
├─Linear: 1-6                            163,968
├─Linear: 1-7                            8,256
├─BatchNorm1d: 1-8                       128
├─Linear: 1-9                            650
├─Dropout: 1-10                          --
==========================================================
```

Fig. 39. Improved CNN Architecture

C. ResNet

Residual Networks [2] introduced the groundbreaking concept of residual learning to address the vanishing gradient problem in deep neural networks. It employs skip connections, allowing the model to bypass certain layers, which significantly enhances gradient flow during backpropagation. The architecture used here ResNet-18 is composed of 18 convolutional layers arranged in a hierarchical structure, utilizing batch normalization and ReLU activations to ensure efficient training. ResNet-18 strikes a balance between depth and computational efficiency, making it particularly effective for tasks involving moderately complex datasets. Its ability

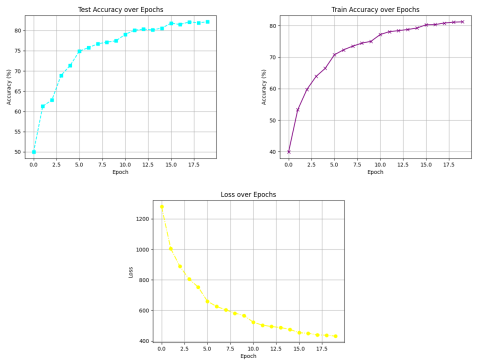

Fig. 40. Improved CNN - (a) Testing Accuracy, (b) Training Accuracy, (c) Cumulative Loss over an epoch

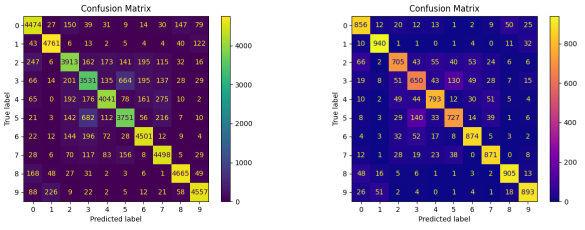

Fig. 41. Improved CNN - (a) Train Confusion Matrix, (b) Test Confusion Matrix

to learn residual mappings ensures superior feature extraction capabilities, reducing training errors and enhancing overall accuracy.

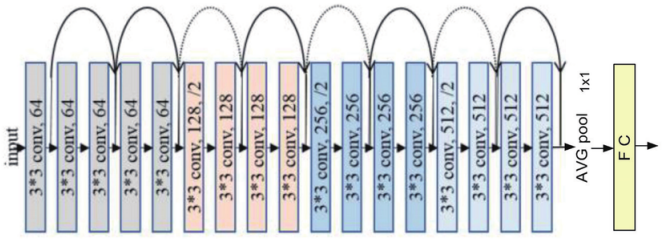

Fig. 42. ResNet Architecture

D. ResNeXt

ResNeXt builds upon the foundation of ResNet by introducing a cardinality dimension as described in [**?**], which refers to the number of parallel pathways within each residual block. This modification enhances the network's representational capacity without significantly increasing computational costs. By leveraging grouped convolutions, ResNeXt achieves greater flexibility in feature extraction, enabling it to capture diverse patterns in the input data. Unlike ResNet, ResNeXt scales effectively by increasing cardinality rather than depth or width, which helps mitigate overfitting while maintaining a high level of model performance. Its modular architecture, characterized by homogeneous blocks, simplifies implementation and allows seamless scaling across different computational environments.

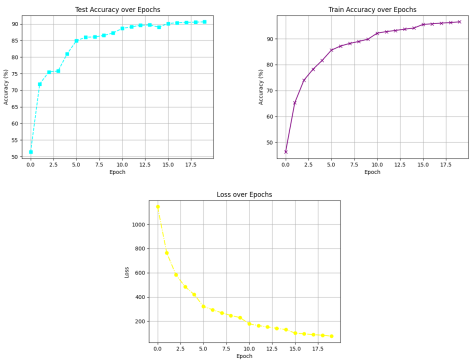

Fig. 43. ResNet CNN - (a) Testing Accuracy, (b) Training Accuracy, (c) Cumulative Loss over an epoch

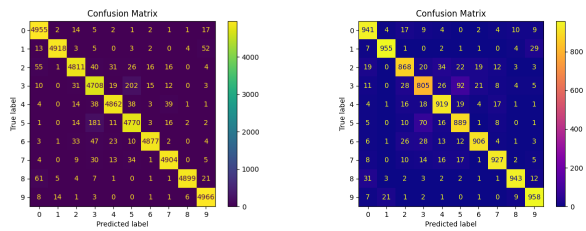

Fig. 44. ResNet CNN - (a) Train Confusion Matrix, (b) Test Confusion Matrix

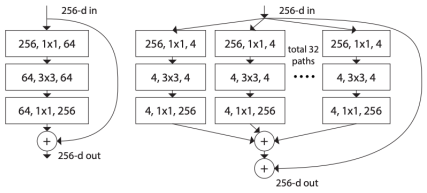

Fig. 45. ResNeXt Architecture

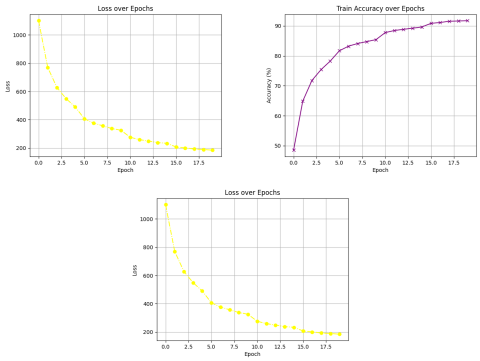

Fig. 46. ResNeXt CNN - (a) Testing Accuracy, (b) Training Accuracy, (c) Cumulative Loss over an epoch

E. DenseNet

DenseNet (Dense Convolutional Network) introduces dense connectivity as given in [4], where each layer is connected to every other layer in a feed-forward manner. This design promotes feature reuse, as each layer has access to the outputs

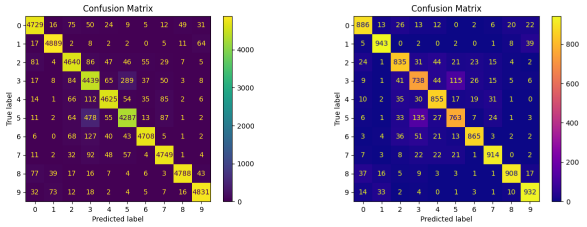

Fig. 47. ResNeXt CNN - (a) Train Confusion Matrix, (b) Test Confusion Matrix

of all preceding layers, leading to more efficient feature propagation and gradient flow. DenseNet reduces the number of parameters by eliminating redundant feature maps, making it computationally efficient despite its dense connections. The architecture is particularly effective at learning fine-grained features, which makes it highly suitable for tasks involving intricate patterns. DenseNet's compactness and efficient use of parameters contribute to improved performance on small to medium-sized datasets.

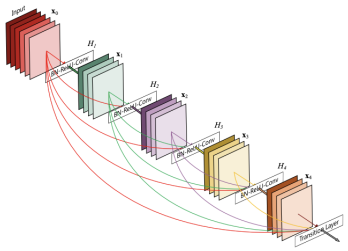

Fig. 48. DenseNet Architecture

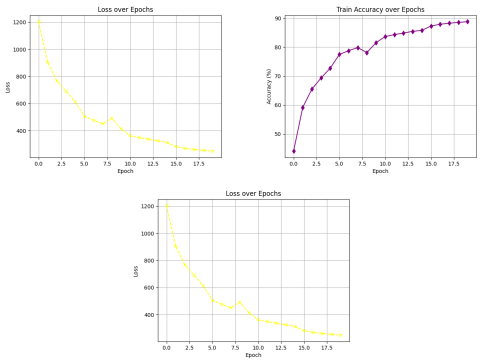

Fig. 49. DenseNet CNN - (a) Testing Accuracy, (b) Training Accuracy, (c) Cumulative Loss over an epoch

F. Analysis

The models were trained on a macOS running computer with M2 chip. In the table 1 we can see the total number of parameters and time taken to train each model for 25 epochs and a minibatchsize of 64 with a learning rate of $6.25 * 10^{-5}$ using the Adam Optimizer.

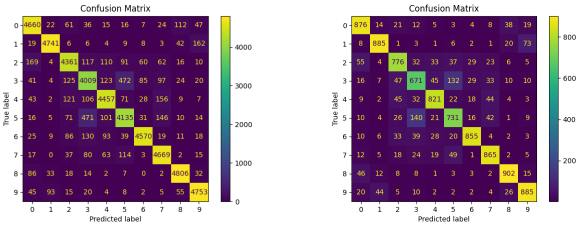

Fig. 50. DenseNet CNN - (a) Train Confusion Matrix, (b) Test Confusion Matrix

TABLE I
NETWORK COMPARISON

Network	Parameters	Time to Train	Accuracy
Simple CNN	1147466	22 minutes	49.96%
Improved CNN	1380062	40 minutes	76.23%
ResNet	11178762	1 hour	91.3%
ResNeXt	25738186	1.5 hours	85.48%
DenseNet	7978856	3 hours	83.72%

G. Conclusion

In this paper, we developed and analyzed five different convolutional neural network (CNN) architectures to classify CIFAR-10 images. The first part involved creating a simple CNN, providing a baseline for comparison. In the second part, we enhanced the original CNN by incorporating batch normalization, dropout, and additional convolutional layers, which significantly improved performance. The third part extended the study by implementing state-of-the-art architectures, including ResNet, ResNeXt, and DenseNet, which introduced deeper residual connections, aggregated transformations, and densely connected layers, respectively. Each architecture was evaluated based on training and testing accuracies, loss values, and confusion matrices. Our comparison showed a clear trend: as the network complexity increased, so did the performance, with ResNet achieving the highest accuracy as shown in I. However, the training time also grew, reflecting the trade-off between model depth and computational cost. This study highlights the importance of selecting the right architecture based on the desired trade-off between accuracy and computational efficiency for image classification tasks.

REFERENCES

[1] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," *arXiv preprint arXiv:1511.08458*, 2015. [Online]. Available: https://arxiv.org/abs/1511.08458.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385.

[3] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[4] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely Connected Convolutional Networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993.