Homework 0 - Alohomora
RBE/CS549 Computer Vision

Shambhuraj Anil Mane
Robotics Engineering, Worcester Polytechnic Institute
Email: samane@wpi.edu

***Abstract*—This report presents a two-phase assignment focusing on boundary detection and neural network architecture evaluation. In Phase 1, the objective is to develop a pb (probability of boundary) algorithm for efficient boundary detection in images. Classical edge detection algorithms, such as Canny and Sobel, serve as baselines for comparison. The proposed algorithm explores brightness, color, and texture information across multiple scales to generate per-pixel probability of boundaries.**
Moving to Phase 2, the assignment transitions into implementing multiple neural network architectures for analysis on various criteria, including the number of parameters, train and test set accuracies. The evaluation aims to provide a detailed analysis of the effectiveness of each architecture, shedding light on why one outperforms another. The dataset chosen for this phase is CIFAR-10, comprising 60,000 32×32 color images distributed across 10 classes.
The combined efforts in both phases contribute to advancing the understanding of image processing techniques and neural network architectures in computer vision.

Index Terms — Gaussian filter, Leung-Malik Filters, Gabor Filters, Sobel and Canny edge detection, BSDS500, CIFAR10, ResNet.

I. PHASE 1: SHAKE MY BOUNDARY

Edge detection is a key task in computer vision, involving the identification of abrupt changes in intensity within images. Traditionally, methods like the Sobel operator and the Canny edge detector have been employed for this purpose. However, the more recent Pb (Probability of Boundary) edge detection algorithm has surpassed these conventional techniques by integrating information on texture, brightness and color discontinuities in addition to intensity variations.

In this project, we developed a simplified iteration of the Pb algorithm known as Pb-Lite. Despite its streamlined design, Pb-Lite exhibits moderate performance compared to the Canny and Sobel edge detectors. This enhancement in edge detection capabilities is crucial for more effective pattern recognition and image analysis.

A. Filter Banks

The first step in the Pb-Lite Contour Detection is to create a set of filters with which the image of interest is filtered to produce useful responses. A filter bank is defined as a collection of filters. There are three types of filter banks used for this process:

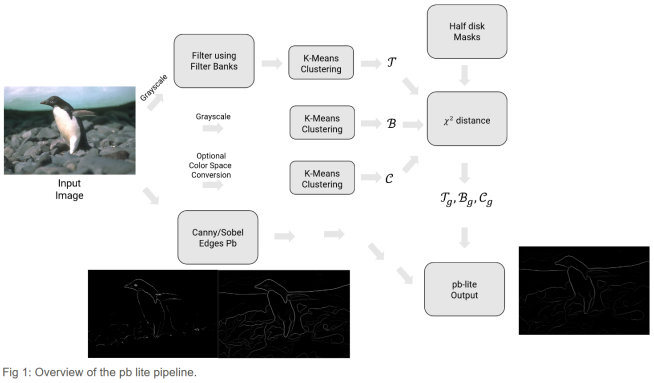

Fig. 1. Pb-Lite overview

1) Oriented DoG Filters: These filters are crafted by convolving a basic Sobel filter with a Gaussian kernel and subsequently rotating the resultant filter. To cater to a specific requirement for "o" orientations (ranging from 0 to 360 degrees) and "s" scales, the implementation produces a total of "s × o" filters. A practical example is a filter bank sized 2 × 16, featuring 2 scales and 16 orientations.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

$$SobelFilter_{3\times3} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} /8 \tag{2}$$

$$DoGFilter = SobelFilter * G(x,y) \tag{3}$$

$$RotatedDoGFilter = RotatedSobelFilter * G(x,y) \tag{4}$$

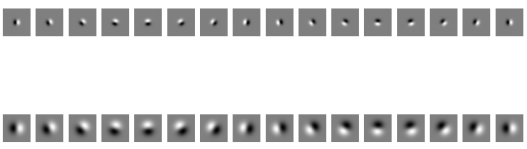

Fig. 2. Oriented DoG filter bank

2) Leung-Malik Filters: In my implementation, I have integrated the Leung-Malik (LM) filter bank, a comprehensive set of 48 filters with multi-scale and multi-orientation capabilities. This filter bank consists of 36 filters derived from first and second-order derivatives of Gaussians, distributed across 6 orientations and 3 scales. Additionally, it includes 8 Laplacian of Gaussian (LOG) filters and 4 Gaussian filters, resulting in a total of 48 filters.

For the LM Small (LMS) version, the filters are positioned at basic scales ($\sigma = \{1, 2-\sqrt{2}, 2, 2+\sqrt{2}\}$). The first and second derivative filters are situated at the first three scales with an elongation factor of 3 ($\sigma_x = \sigma$ and $\sigma_y = 3\sigma_x$), while the Gaussians are located at the four basic scales. The 8 LOG filters are positioned at σ and 3σ.

In the case of LM Large (LML), the filters are placed at the basic scales ($\sigma = \{2-\sqrt{2}, 2, 2+\sqrt{2}, 4\}$). This implementation allows for a versatile and comprehensive utilization of the Leung-Malik filter bank, contributing to the enhancement of image processing tasks.

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y}\exp\left(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2}\right) \tag{5}$$

$$laplacianfilter_{3\times3} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{6}$$

$$LMFilter = LaplacianFilter * G(x,y) \tag{7}$$

$$RotatedLMFilter = LaplacianFilter * RotatedG(x,y) \tag{8}$$

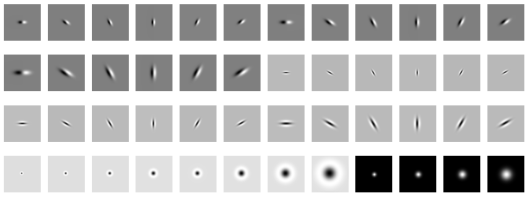

Fig. 3. LM Small filter bank

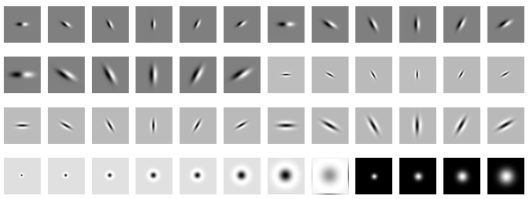

Fig. 4. LM Large filter bank

3) Gabor Filters: Gabor Filters, which are tailored to mimic filters found in the human visual system. A Gabor filter is essentially a Gaussian kernel function that is modulated by a sinusoidal plane wave. This design allows the filter to be sensitive to variations in both spatial frequencies and orientations, making it particularly effective in capturing features similar to those perceived by the human visual system. The utilization of Gabor Filters in my implementation enhances the ability to extract relevant information from images, contributing to improved pattern recognition and feature detection.

$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{9}$$

$$x' = x\cdot\cos(\theta) + y\cdot\sin(\theta) \tag{10}$$

$$sinusoidal = \cos(2\pi f x') \tag{11}$$

$$\text{Gabor Filter} = \text{sinusoidal} \times G(x,y) \tag{12}$$

$$\text{RotatedGabor Filter} = \text{RotatedSinusoidal} \times G(x,y) \tag{13}$$

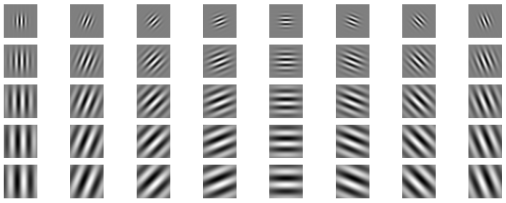

Fig. 5. Garbor filter bank

B. Half discs

Half discs are just binary images of half discs with different orientation. These discs are used to calculate the gradient i.e. Chi-Square distance for the further filtering operations. For this implementation, 8 orientations and 3 scales, [5, 10, 20] are considered.

$$\chi^2(g,h) = \frac{1}{2}\sum_{i=1}^{K}\frac{(g_i - h_i)^2}{g_i + h_i} \tag{14}$$

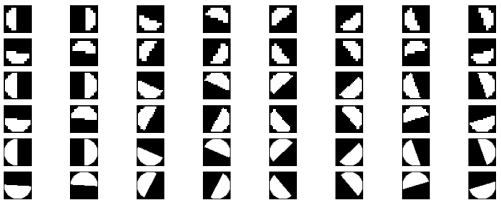

Fig. 6. Half Discs Masks

C. Texture, Brightness and Color Maps

Each pixel recieves N filter responses from N filters in the filter bank. By substituting a discrete texton ID for each N-dimensional vector, we can simplify this representation.

$$Total filters = DoG + LM + Garbor = 168 \quad (15)$$

Our reshaped input to kmeans is as [pixel, Nfilters] ie [321×481 , 168]. We can use kmeans to cluster the filter responses at all image pixels into K textons ie [321×481 , 64]. This generates the Texton map by predicting value from these 64 clusters thus reducing dimension of output to [321×481 , 1].

In the similar way a brightness map and a color map are also constructed using Kmeans clustering with 16 clusters. The concept of the brightness map is to capture the brightness changes in the image and similarly for color map, the color changes are to be captured.

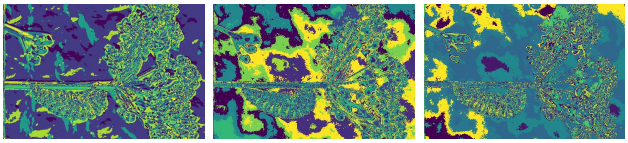

Fig. 7. Texton, Brightness and Color map for image 2

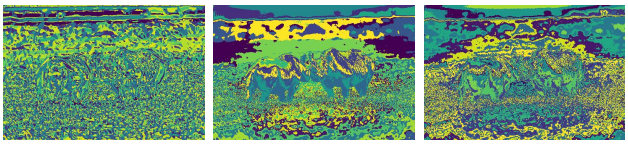

Fig. 8. Texton, Brightness and Color map for image 8

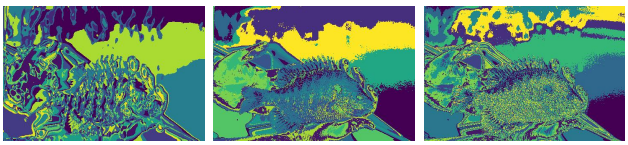

Fig. 9. Texton, Brightness and Color map for image 9

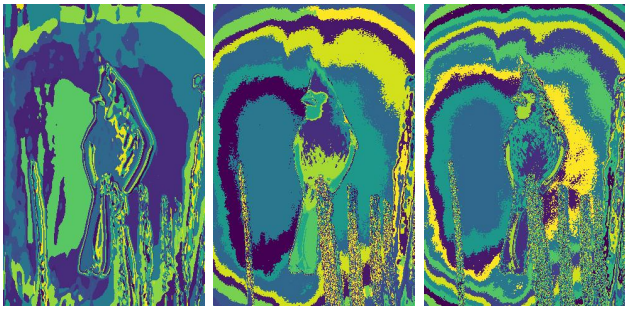

Fig. 10. Texton, Brightness and Color map for image 10

D. Texture, Brightness and Color Gradients T_g, B_g, C_g

Using the generated half-disc masks along with the Chi-square distance, we can compute the gradient map for texture, brightness, and color of the input image. We obtain the gradient of the pixel by applying pairs of half-disk masks to each pixel of the maps and computing the mean of Chi square distances.

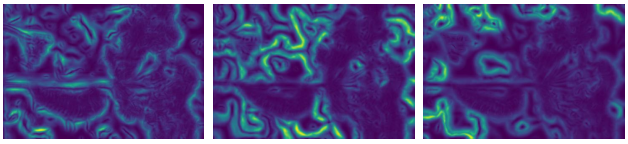

Fig. 11. Texton, Brightness and Color gradient for image 2

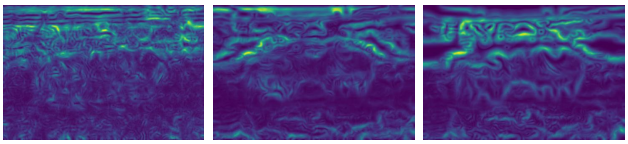

Fig. 12. Texton, Brightness and Color gradient for image 8

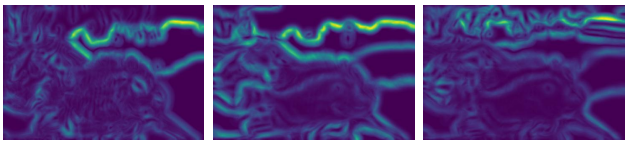

Fig. 13. Texton, Brightness and Color gradient for image 9

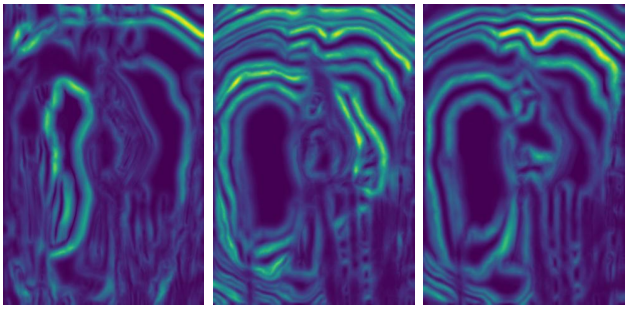

Fig. 14. Texton, Brightness and Color gradient for image 10

E. Combining information from features with sobel and canny baselines

In the final step of the pb-lite algorithm, the information from the texture, brightness and color gradient is combined with the Canny and Sobel baselines using the Hadamard product operator and choosing suitable weights. The results are shown for each image.

$$\text{PbEdges} = (T_g + B_g + C_g)^3 \odot (w_1 * \text{cannyPb} + w_2 * \text{sobelPb}) \quad (16)$$

In this equation:

- PbEdges represents the output.
- T_g, B_g, and C_g are terms raised to the power of 3.
- $\odot$ denotes element-wise multiplication.
- w_1 and w_2 are weights.
- $*$ represents convolution.
- cannyPb and sobelPb are baseline outputs.

Fig. 15. Canny Baseline, Sobel Baseline and Pb-lite output (Threshold) for image 2

Fig. 16. Canny Baseline, Sobel Baseline and Pb-lite output (Threshold) for image 8

Fig. 17. Canny Baseline, Sobel Baseline and Pb-lite output (Threshold) for image 9

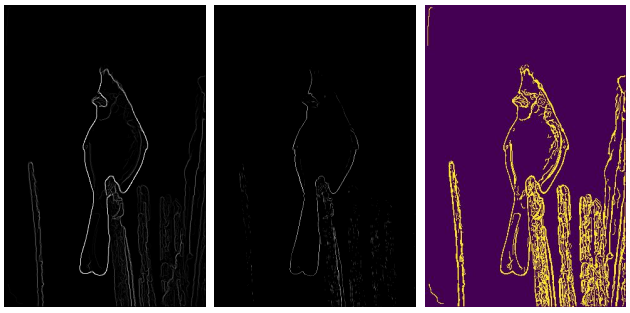

Fig. 18. Canny Baseline, Sobel Baseline and Pb-lite output (Threshold) for image 10

F. CONCLUSION AND ANALYSIS

When comparing contours generated by the PbLite algorithm against Sobel and Canny baselines, the following observations can be made:

- From the output comparative images, Canny edge detector seems to have better output.
- The pb-lite algorithm performs very well to reduce the background noise from an image in some images.
- The performance of edge detectors often depends on the appropriate tuning of parameters. Some methods might be more robust to variations in parameter settings, while probability of boundary require careful adjustment.

II. PHASE 2: DEEP DIVE ON DEEP LEARNING

For phase 2, I have trained a baseline convolutional neural network, implemented techniques to improve its accuracy, and implemented ResNet, ResNeXt and DenseNet architectures. For each model, I present details on the training and testing accuracy over batch and epochs, loss over batch and epoch, number of parameters, model architecture, and confusion matrices. Finally, I provide a comparison of all the models on key metrics.

A. CIFAR-10 Dataset

CIFAR-10 is a widely used dataset in the field of computer vision and machine learning. It consists of a collection of 60,000 small, 32x32-pixel color images, which are divided into 10 distinct classes. Each class contains 6,000 images, with 5,000 images used for training and 1,000 images reserved for testing. The dataset is specifically designed for image classification tasks, where the goal is to train a model to correctly identify and categorize images into one of the ten predefined classes.

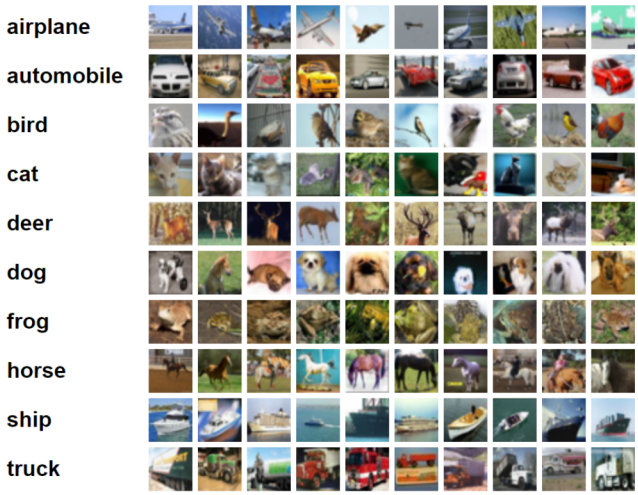

Fig. 19. CIFAR-10 Dataset

B. Methodology

For this phase of the assignment I have selected optimizer Stochastic Gradient Descent (SGD) based on its popularity and easy implementation. The loss function used here is cross entropy loss. For CIFAR data, which consists of small

images categorized into different classes, cross-entropy loss is often used in combination with softmax activation for the output layer of a neural network. *torch.nn.CrossEntropyLoss* automatically applies the softmax activation function to the model's output and computes the cross-entropy loss.

Four network architecture are implemented for this assignment which are :

- Baseline Neural Network
- ResNet Neural Network
- ResNeXt Neural Network
- DenseNet Neural Network

and analyzed based on their test accuracy. For improving accuracy the images are standardized before appending in batch. Fixed learning rate of 0.001 is used for training each model. Also, data augmentation is done to improve the accuracy and the results are shown in the analysis section.

C. Baseline Neural Network

The Baseline Neural Network is a convolutional neural network (CNN) designed for image classification on the CIFAR-10 dataset. It has two convolutional layers with max-pooling, followed by three fully connected layers. The first convolutional layer takes RGB images and produces 6 feature maps, while the second layer increases the depth to 16. ReLU activation functions are applied after each layer to introduce non-linearity. The output is flattened before passing through three fully connected layers, progressively reducing the number of neurons. The final layer produces the output with a size specified by OutputSize. The architecture is shown in the 35

D. ResNet Neural Network

The ResNet Neural Network for image classification, specifically ResNet18. The network comprises basic building blocks that are stacked together in different layers to form the complete ResNet structure. Each basic block consists of two convolutional layers with batch normalization and a shortcut connection. The ResNet18 architecture consists of a initial convolutional layer followed by four layers of basic blocks, with decreasing spatial dimensions and increasing depth. The final layer is a fully connected layer with a linear activation function, producing the output for classification. The ResNet18 is designed to handle 3-channel images (e.g., RGB) and has a total of 18 weight layers, including both convolutional and fully connected layers. The architecture employs average pooling before the final fully connected layer. This modular structure and skip connections (shortcuts) help mitigate the vanishing gradient problem, enabling the training of deep neural networks.

E. ResNeXt Neural Network

The ResNeXt Neural Network architecture for image classification, specifically ResNeXt29. It employs grouped convolutions within basic blocks. Each basic block consists of three convolutional layers with batch normalization and ReLU activation, incorporating a grouped convolution strategy. The

Fig. 20. Baseline model architecture

```
PS C:\sam\WPI\COURSEWORK\CV\Assignemnts\HW0\samane_hw0\Phase2\Code>
Number of parameters in this model are 10
  0%|          | 0/10000 [00:00<?, ?it/s]
[783  24  45  12  29   5   5   9  72  21] (0)
[  8 848   3   6  11   1   8   2  35  52] (1)
[ 48   6 632  52 140  45  31  40  24  14] (2)
[ 17   7  55 567  84 167  40  32  22  25] (3)
[ 16   5  28  28 822  16  11  45   9  19] (4)
[  7   3  41 138  73 605  14  48   3   5] (5)
[ 15  10  38  61  64  34 781   5  14   8] (6)
[ 15   3  17  30  72  34   2 809   4  15] (7)
[ 27  35   9   5   7   2   3   1 924  12] (8)
[ 15  58   7   5  10   1   3   9  32 841] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 76.12 %
```

Fig. 21. Confusion Matrix of the baseline model on training data

ResNeXt architecture, comprises multiple layers of these basic blocks, with the number of blocks in each layer specified by the num_blocks parameter. The model's cardinality (number of groups) and bottleneck width dynamically increase within each layer, contributing to its representational power. The forward pass includes convolutional and pooling operations, culminating in a linear layer for classification. ResNeXt29 is configured with three stages of layers and achieves increased width in the bottleneck with a cardinality of 32.

F. DenseNet Neural Network

The DenseNet Neural Network architecture for image classification, specifically DenseNet121 is designed leveraging a dense connectivity pattern where each layer receives input from all preceding layers. The building blocks, represented by

```
Number of parameters in this model are 10
  0%|          | 0/10000 [00:00<?, ?it/s]
[652  25  68  23  58   5  16  15 104  34] (0)
[ 17 787   7  13   7   4  10   6  56  93] (1)
[ 85   9 490  55 151  91  35  41  24  19] (2)
[ 32  16  59 441 100 180  61  44  29  38] (3)
[ 23  11  52  46 687  44  31  76  11  19] (4)
[ 14   7  71 158  99 539  25  70   9   8] (5)
[ 11  21  48  89  91  42 660   9  15  14] (6)
[ 15   6  27  37 109  68   8 694  11  25] (7)
[ 51  53  12  17   6   3  15   2 813  28] (8)
[ 50  90   6  13  24   5   4  17  73 718] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 64.81 %
```

Fig. 22. Confusion Matrix of the baseline model on testing data

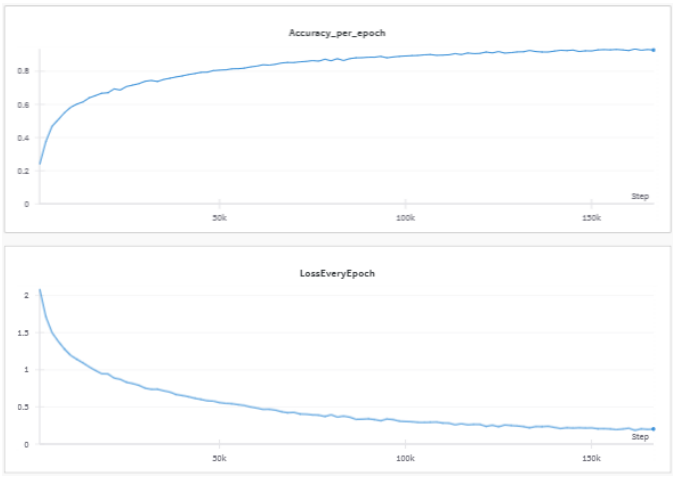

Fig. 23. Baseline: Plot of validation accuracy and loss over Epochs

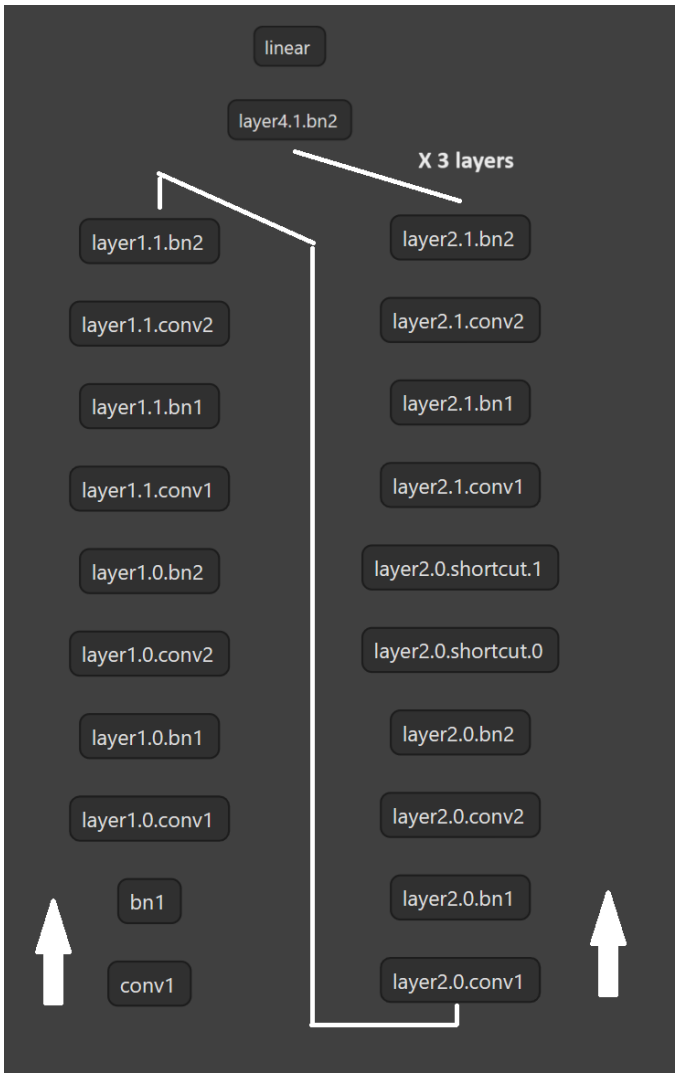

Fig. 24. ResNet model architecture

```
PS C:\sam\WPI\COURSEWORK\CV\Assignemnts\HW0\samane_hw0\Phase2\Code> python3 .\Test.py
Number of parameters in this model are 122
  0%|          | 0/10000 [00:00<?, ?it/s]
[998   0   1   1   0   0   0   0   3   2] (0)
[  5 954   0   0   0   0   3   2   4   6] (1)
[  15    0 1008    2    3    1    1    1    1    0] (2)
[  7   0   9 970   9  11   3   3   1   3] (3)
[  7   0   6   3 980   0   0   3   0   0] (4)
[  0   0   5  29   4 887   0  12   0   0] (5)
[  2   1  13  14   6   4 989   0   0   1] (6)
[  4   1   1   2  20   2   0 969   1   1] (7)
[   7    3    0    5    0    0    0    0 1008    2] (8)
[  9   1   0   1   0   0   0   0   3 967] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 97.3 %
```

Fig. 25. Confusion Matrix of the ResNet model on training data

the Bottleneck class, consist of two convolutional layers with batch normalization and ReLU activation. Transition blocks, defined by the Transition class, reduce spatial dimensions through average pooling and adjust channel dimensions. The overall DenseNet structure, orchestrated by the DenseNet class, comprises multiple dense blocks interleaved with transition blocks, progressively increasing the number of channels. DenseNet121 concludes with global average pooling and a fully connected layer for classification, offering a compact yet expressive architecture for image recognition tasks.

G. Analysis

1) Training and Test Accuracy for Different Models: The models demonstrate high training accuracy, indicating effective learning during training. The data shown in 36 is the best performed model for individual architecture. Test accuracy reveals varying performance, with DenseNet achieving the highest at 82.06%, surpassing ResNeXt and ResNet. ResNet and ResNeXt show competitive test accuracies, highlighting their efficacy for image classification tasks. The baseline

```
PS C:\sam\WPI\COURSEWORK\CV\Assignemnts\HW0\samane_hw0\Phase2\Code>
Number of parameters in this model are 122
  0%|          | 0/10000 [00:00<?, ?it/s]
[875   7  36  18  11   3   4   5  27  14] (0)
[ 25 863   3   6   2   4  10   1  22  64] (1)
[ 70   0 742  33  56  35  36  21   4   3] (2)
[ 38   2  71 603  59 118  33  49  19   8] (3)
[ 28   1  66  46 795  23   7  29   5   0] (4)
[ 17   2  77 140  35 649  16  49   1  14] (5)
[ 18   3  52  65  31  24 789   6   6   6] (6)
[ 28   2  40  27  79  38   2 776   2   6] (7)
[ 73  14   9   8   2   0   5   3 868  18] (8)
[ 37  40   7  14   1   4   2  13  18 864] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 78.24 %
```

Fig. 26. Confusion Matrix of the ResNet model on testing data

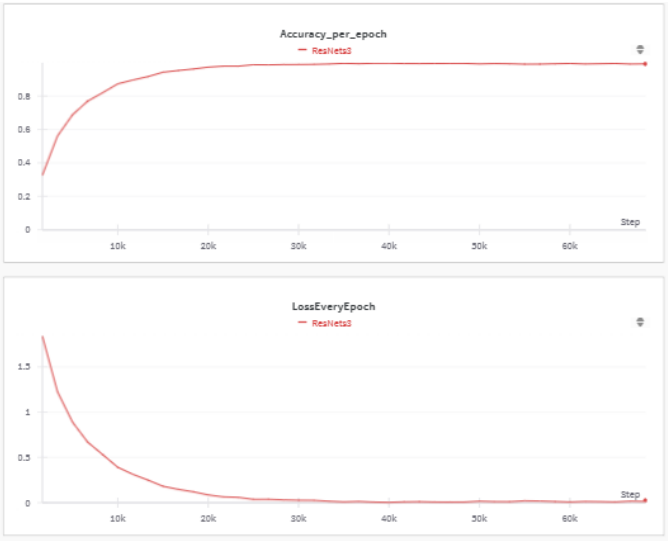

Fig. 27. ResNet: Plot of validation accuracy and loss over Epochs

model lags in test accuracy compared to the more complex architectures but still achieves reasonable performance.

2) Model size and number of parameters for Different Models: The models size and number of parmaters are compared in the 37. The analysis of the model characteristics reveals distinct trade-offs between size and complexity. The baseline model, with a modest size of 0.49 MB and only 10 parameters, showcases efficiency in terms of both memory footprint and model simplicity. In contrast, the ResNeXt model, while significantly larger at 37.487 MB, achieves a higher level of complexity with 188 parameters, illustrating the inherent compromise between model size and expressiveness. The ResNet model strikes a balance with a moderate size of 87.34 MB and 122 parameters. DenseNet, with 27.74 MB and 722 parameters, reflects the challenge of maintaining a compact model size while accommodating a more intricate architecture. Overall, the analysis underscores the need to consider both model size and parameter count in selecting an optimal model for a given application.

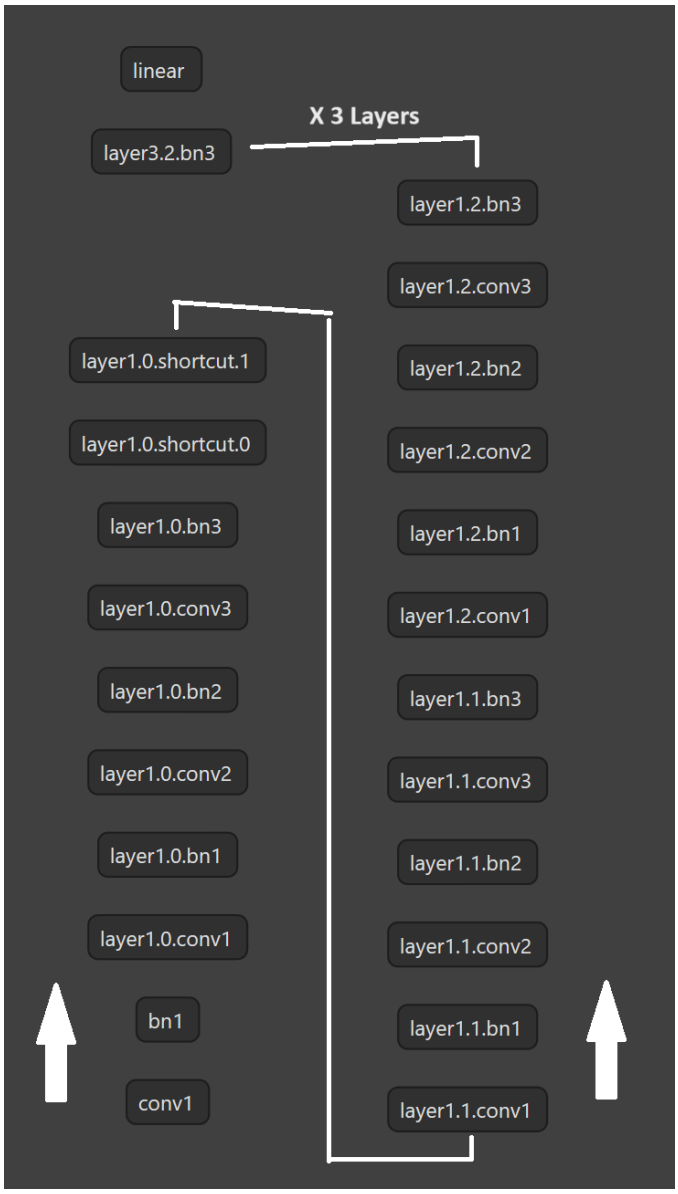

Fig. 28. ResNeXt model architecture

```
PS C:\sam\WPI\COURSEWORK\CV\Assignemnts\HW0\samane_hw0\Phase2\Code> python3 .\Test.py
Number of parameters in this model are 188
  0%|          | 0/10000 [00:00<?, ?it/s]
[991   1   5   0   0   0   0   0   2   6] (0)
[  2 916   0   4   0   1   2   0   4  45] (1)
[  9   0 1010   4   1   3   3   0   0   2] (2)
[  4   0   7 978   2  15   7   0   0   3] (3)
[ 10   0   7   8 969   1   2   1   0   1] (4)
[  1   0   4  18   2 905   4   0   0   3] (5)
[  2   1   4   9   0   2 1008   0   3   1] (6)
[  7   0  14   4   5   7   3 953   0   8] (7)
[ 14   1   3   3   0   2   2   0 984  16] (8)
[  1   0   0   2   0   1   0   0   0 977] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 96.91 %
```

Fig. 29. Confusion Matrix of the ResNeXt model on training data

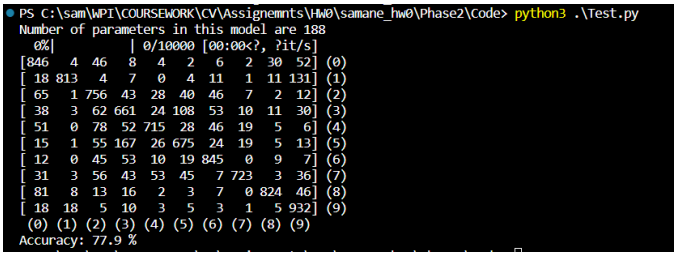

Fig. 30. Confusion Matrix of the ResNeXt model on testing data

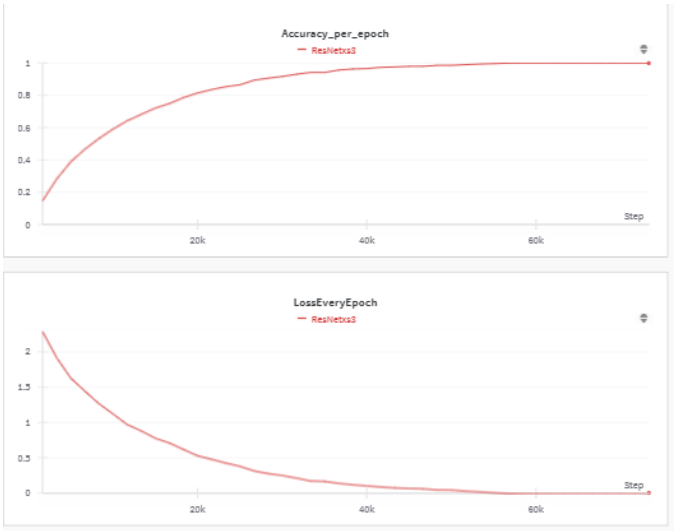

Fig. 31. ResNeXt: Plot of validation accuracy and loss over Epochs

3) Accuracy trend over epochs : The shown results in 38 suggest a delicate balance between underfitting and overfitting during the model training process. Models trained for 20, 30, and 40 epochs exhibit competitive test accuracies, indicating a sweet spot for generalization. However, the decline in accuracy for models trained for 55, 70, and 95 epochs suggests overfitting, where the models start memorizing the training data excessively. This underscores the importance of optimal epoch selection to achieve a well-generalized model without succumbing to the pitfalls of underfitting or overfitting. The findings emphasize the need for a nuanced understanding of the trade-off between model complexity and generalization performance in machine learning.

Conclusion

In conclusion, the implementation and comparison of multiple neural network architectures on the CIFAR-10 dataset have provided valuable insights into their performance metrics. The analysis covered key aspects such as the number of parameters, training and test set accuracies, and model size. Notably, the baseline model demonstrated efficiency with a small size and minimal parameters, suitable for resource-constrained environments. ResNeXt, exhibiting a larger size and parameter count, showcased increased complexity, potentially beneficial

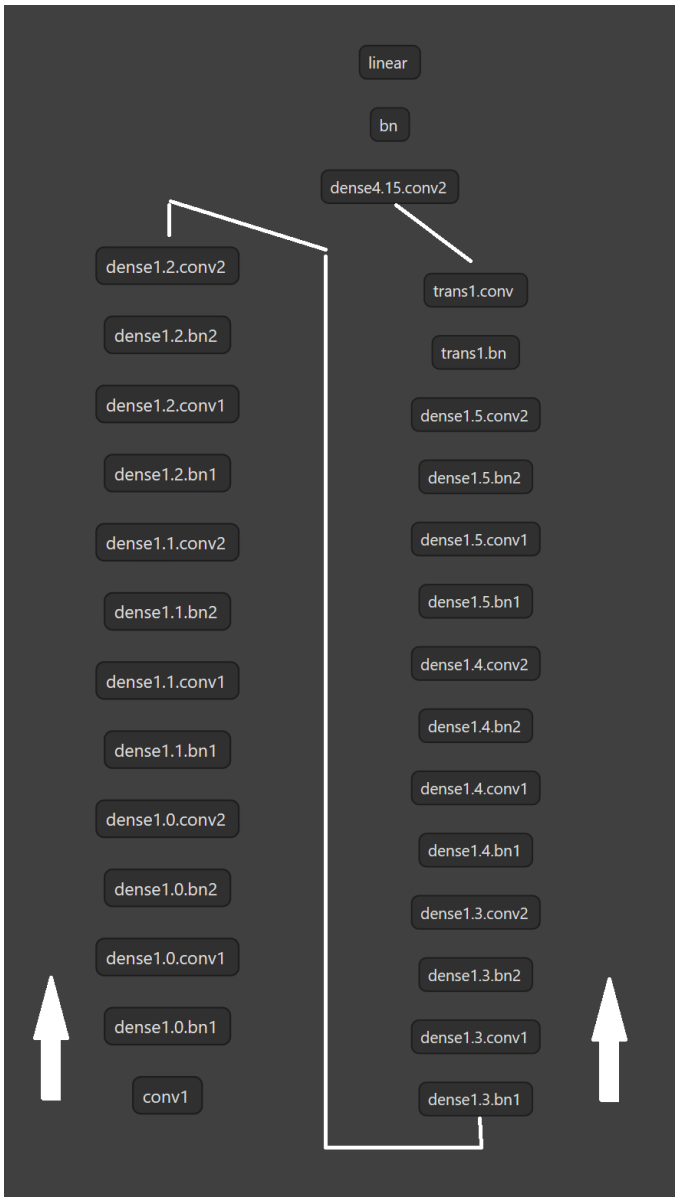

Fig. 32. DenseNet model architecture

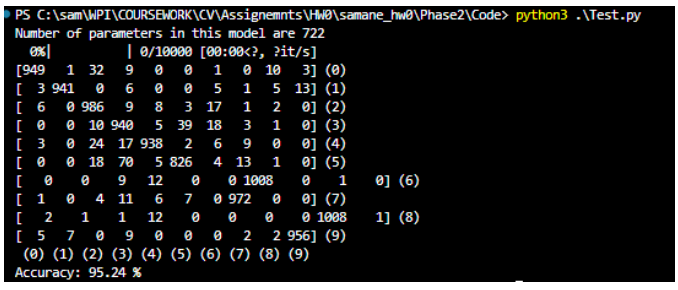

Fig. 33. Confusion Matrix of the DenseNet model on training data

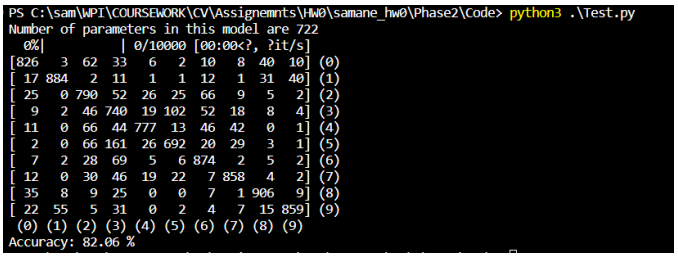

Fig. 34. Confusion Matrix of the DenseNet model on testing data

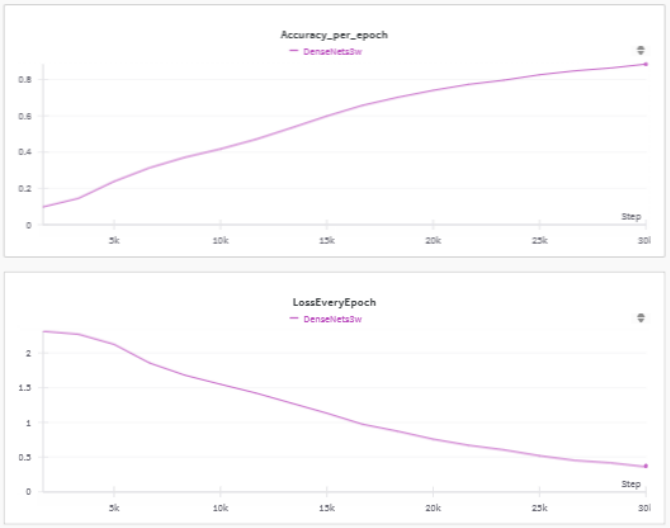

Fig. 35. DenseNet: Plot of validation accuracy and loss over Epochs

Fig. 36. Training and Test Accuracy for Different Models

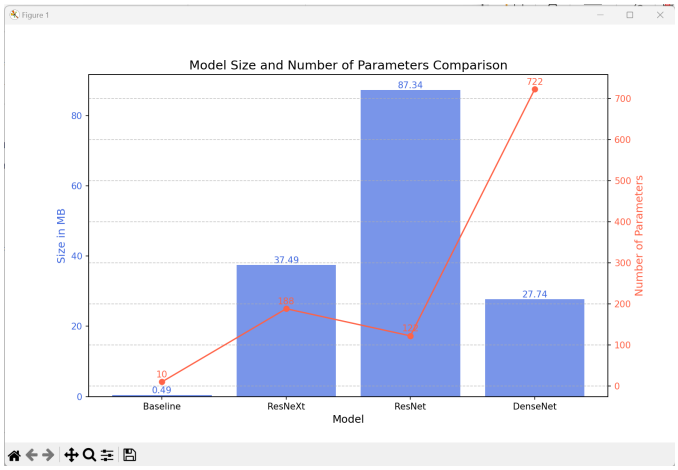

Fig. 37. Model size and number of parameters for Different Models

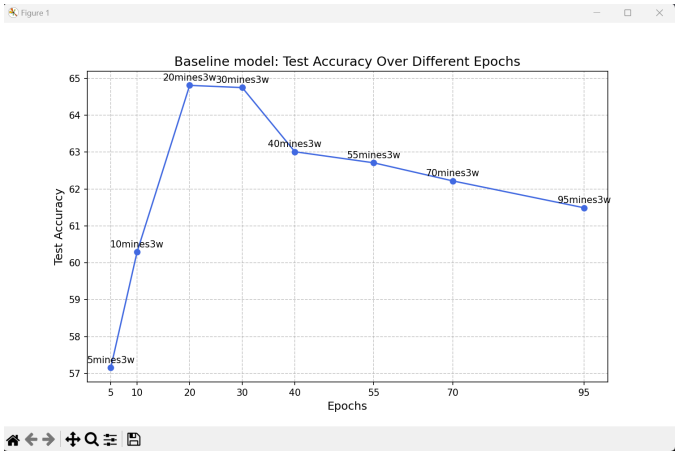

Fig. 38. Accuracy trend over epochs

in cloud-based services. The ResNet model found a balance between size and complexity, making it versatile for mobile applications. DenseNet, despite its intricate architecture, presented challenges in managing size while capturing complex patterns. These findings offer a nuanced understanding of how architectural choices impact model performance in various application scenarios, aiding in informed decision-making for future machine learning projects.

REFERENCES

[1] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks." arXiv, Jan. 28, 2018. doi: 10.48550/arXiv.1608.06993.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." arXiv, Dec. 10, 2015. doi: 10.48550/arXiv.1512.03385.
[3] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks." arXiv, Apr. 10, 2017. doi: 10.48550/arXiv.1611.05431.
[4] https://github.com/kuangliu/pytorch-cifar
[5] PyTorch Tutorial 14 - Convolutional Neural Network (CNN)
[6] OpenAI. (2022). ChatGPT. https://www.openai.com/chatgpt