Homework 0: Alohamora!

Rutwik S. Kulkarni
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: rkulkarni1@wpi.edu

***Abstract*—(1 late day)This homework involves developing a computer vision algorithm for boundary detection and comparing different neural network architectures for image classification. The first task is to create a simplified version of the Probability of Boundary (PB) algorithm, focusing on identifying image boundaries by analyzing brightness, color, and texture information. This version is expected to be more effective than traditional edge detection methods like Canny and Sobel, and its performance will be assessed using the Berkeley Segmentation Data Set 500 (BSDS500). The second task involves implementing various neural network architectures and evaluating them using the CIFAR-10 dataset, which consists of 60,000 color images in 10 classes. The evaluation criteria will include the number of parameters, training and test accuracies, providing a comparative analysis of the architectures. The goal is to gain practical understanding in advanced image processing and neural network design.**

I. Boundary Detection

A. Introduction

This report for boundary detection introduces a simplified version of the Probability of Boundary (PB) algorithm. The primary objective was to enhance the boundary detection process by using a combination of various filter banks - Oriented DoG filters, Leung-Malik Filters, and Gabor Filters. These filters are crucial for extracting detailed texture information from images. Additionally, we developed Texton, Brightness, and Color Maps to support the detection process. This approach aims to surpass traditional edge detection methods like Canny and Sobel by incorporating texture, brightness, and color gradients into the detection process. The effectiveness of our method was evaluated using the Berkeley Segmentation Data Set 500 (BSDS500).

B. Oriented Difference of Gradients Filter Banks

The Difference of Gaussian (DoG) filter is used in image processing for detecting edges. It is made by subtracting one Gaussian-blurred image from another Gaussian-blurred image with a different standard deviation. The formula for the DoG filter is $\text{DoG}(x, y) = G(x, y, \sigma_1) - G(x, y, \sigma_2)$, where $G(x, y, \sigma)$ is the Gaussian blur at point (x, y) with standard deviation σ. To create the DoG filter, two Gaussian filters with standard deviations σ_1 and σ_2 are first made. These filters are then used on the image, and their results are subtracted to get the final DoG filter output. The DoG filter finds edges by showing the parts of the image where the brightness changes a lot. In our implementation, we used orientations from 0 to 360 degrees and scales with $\sigma_1 = 2$ and $\sigma_2 = 4$. This allows the filter to detect edges in different directions and scales, which is useful for various tasks in computer vision.

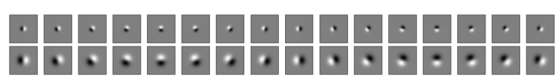

Fig. 1: Oriented Difference of Gaussian Filters

C. Leung-Malik Filters

The Leung-Malik (LM) filter bank is an useful tool in feature extraction, especially in texture analysis. It comprises a combination of Gaussian derivative filters and Laplacian of Gaussian (LoG) filters, varied across multiple scales and orientations. The Gaussian derivatives, generated by calculating first and second-order derivatives of a Gaussian function, are oriented in multiple directions to capture diverse edge and texture features. The Laplacian of Gaussian (LoG) filters, blending Gaussian smoothing with the Laplacian filter, effectively detect blob and edge features at different scales.

This multi-scale approach is realized by using a range of standard deviations for the Gaussian functions. In our implementation, two sets of LM filters are created: a 'small' set with standard deviations [1, $\sqrt{2}$, 2, 2$\sqrt{2}$] and a 'large' set with [$\sqrt{2}$, 2, 2$\sqrt{2}$, 4], each with a kernel size of 49. These filter banks are used for image processing tasks, such as texture classification and scene analysis.

D. Gabor Filter

Gabor filters are used for texture and feature extraction, excelling in capturing spatial and orientation-specific information. These filters are constructed with varying orientations and scales, defined by parameters such as standard deviations σ, kernel size, base orientation θ, wavelength λ, phase offset ψ, and aspect ratio γ. In our implementation, the Gabor filters are generated over multiple orientations by rotating the base kernel. The parameters used include standard deviations $[12, 9, 7, 5, 3]$, a kernel size of 49, base orientation of $\pi/12$, wavelength of 1, phase offset of 1, and aspect ratio of 1, with 8 different orientations. This configuration ensures that the filter bank is versatile for analyzing images across a variety of scales and orientations, making it particularly effective for texture and edge detection.

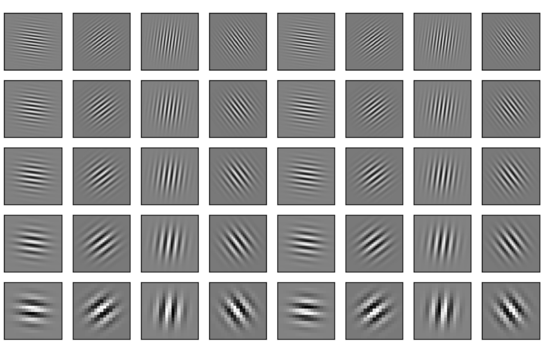

Fig. 2: Gabor Filters

E. Half Disc Masks

Half disc masks are crucial for local gradient computation and texture feature analysis. These masks are generated by dividing a circular shape to create a 'half disc', made for different scales and orientations. The process involves setting the kernel size based on scales, each defined by a radius, forming a circular mask, and then converting it into a half disc by zeroing out one half.

The implementation rotates these half disc masks across a set of predefined angles, including [180, 0, 210, 30, 225, 45, 240, 60, 270, 90, 300, 120, 315, 135, 330, 150] degrees. This rotation covers a range of directional orientations. Post-rotation, the masks are binarized to ensure distinct separation between masked and unmasked regions, essential for precise texture and gradient detection.

Half disc masks are created for scales [5, 15, 25], offering a variety of masks suitable for different image resolutions. These masks are instrumental in edge detection and texture analysis tasks, where they provide critical directional and local contrast information.

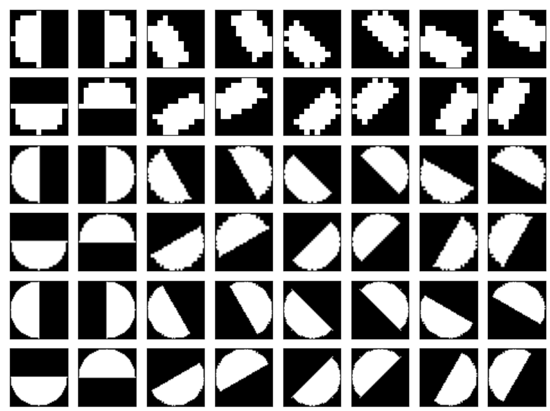

Fig. 3: Half-Disc Masks

F. Texton Maps

Texton maps are used for texture recognition and segmentation. The process for generating texton maps, involves the application of filter banks - Difference of Gaussian (DoG), Leung-Malik (LM), and Gabor filters - to an input image.

Each filter bank is applied to the image separately, and the resulting filtered images are combined to form a comprehensive texton map. This map integrates the textural information extracted by each individual filter, highlighting diverse aspects of the image's texture.

Following the application of all filter banks, the texton map undergoes KMeans clustering. This clustering process groups pixels based on their filter responses, categorizing them into clusters that represent different texture patterns in the image. For clustering, 64 clusters were chosen to balance between capturing texture details and avoiding complexity. This number ensures distinct texture differentiation without over-segmentation, effectively representing the image's textural features for analysis. The result of this clustering is a texton map where each pixel is assigned a cluster ID, indicating its textural characteristics. This final texton map serves as a detailed representation of the image's textural properties.

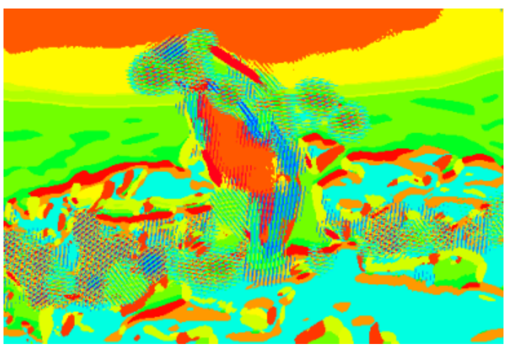

Fig. 4: Texton Map for Image 7

G. Texton Gradients

Texton gradients are a method of capturing the texture information in an image. They are created by applying a series of texture filters to the image, and then computing the gradient of the filter responses. This provides a detailed representation of the texture variation within the image, useful for tasks such as image segmentation and pattern recognition.

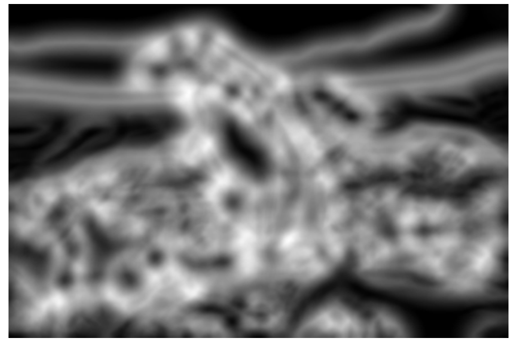

Fig. 5: Texton Gradient for Image 7

H. Brightness Maps

Brightness maps are generated by converting the image to grayscale and then clustering pixel intensities. This process simplifies the image, reducing it to its basic luminance structure, which can be crucial for analyzing images where color information is not as relevant.

Fig. 6: Brightness Map for Image 7

I. Brightness Gradients

Brightness gradients are derived from the brightness maps. They represent the rate of change of brightness across the image. By computing the gradient of the brightness map, we can highlight areas with significant luminance changes, which are often indicative of edges or transitions in the image.

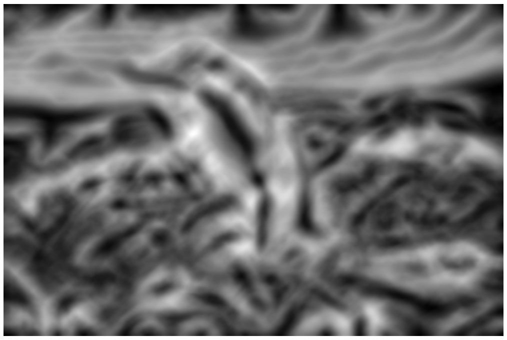

Fig. 7: Brightness Gradient for Image 7

J. Color Maps

Color maps are created by clustering pixel colors in the RGB color space. This process reduces the color complexity of the image, grouping similar colors together. This simplification can be particularly useful for tasks that require color-based segmentation or analysis.

Fig. 8: Color Map for Image 7

K. Color Gradients

Color gradients are similar to brightness gradients but are derived from color maps. They represent the rate of change in color information across the image. These gradients are useful for detecting color transitions and can provide insights into the color dynamics of the image.

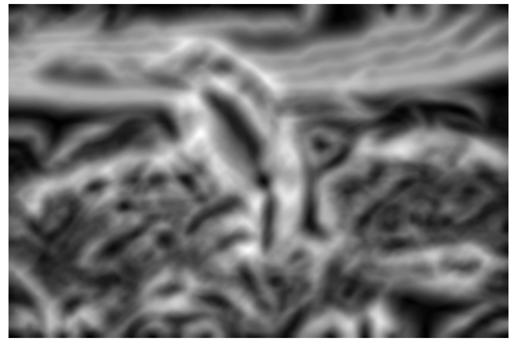

Fig. 9: Color Gradient for Image 7

L. Probability of Boundary Lite

PbLite is an advanced edge detection method that combines the strengths of several approaches, including texton, brightness, and color gradients. PbLite outputs are generated by integrating these various gradients, providing a more comprehensive and nuanced representation of the edges in an image, the drawback being that they are very slow for CPU operations.

Fig. 10: PbLite Output for Image 7

$$PbEdges = \frac{(\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g)}{3} \odot (w_1 * cannyPb + w_2 * sobelPb) \quad (1)$$

M. Comparison with Sobel and Canny Methods

PbLite provides a range of benefits over conventional techniques such as Sobel and Canny. While the Sobel method may occasionally overlook crucial image features, PbLite achieves equilibrium by integrating both texture and color data. In contrast to the Canny method, known for generating numerous false positives, the holistic approach of PbLite leads to more precise and dependable edge detection. Consequently, PbLite proves to be exceptionally effective in situations where accurate delineation of edges is essential.

N. Results

1. Comparison between Texton Maps, Color Maps, and Brightness Maps for all Images in the Test Set :

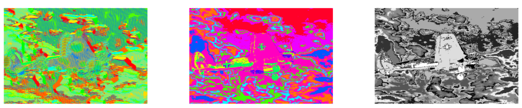

Fig. 11: Maps for Image 1

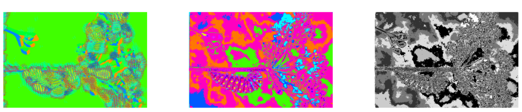

Fig. 12: Maps for Image 2

Fig. 13: Maps for Image 3

Fig. 14: Maps for Image 4

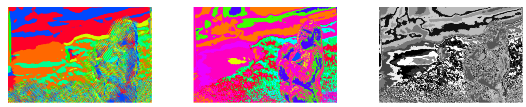

Fig. 15: Maps for Image 5

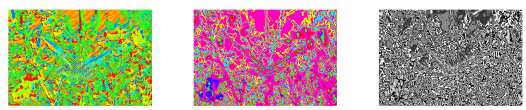

Fig. 16: Maps for Image 6

Fig. 17: Maps for Image 7

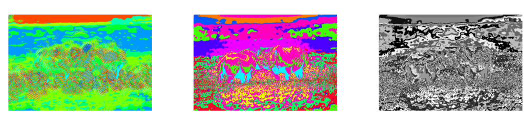

Fig. 18: Maps for Image 8

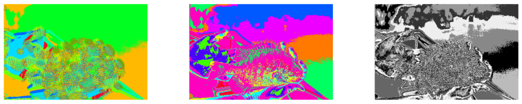

Fig. 19: Maps for Image 9

2. Comparison between Texton Gradients, Color Gradients and Brightness Gradients for all Images in the Test Set

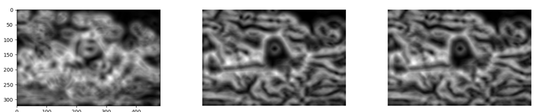

Fig. 20: Gradients for Image 1

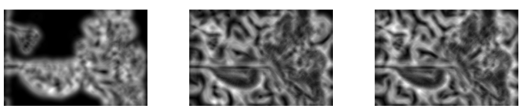

Fig. 21: Gradients for Image 2

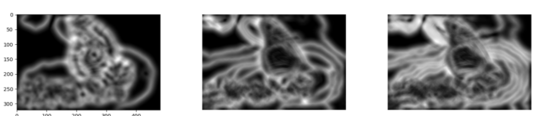

Fig. 22: Gradients for Image 3

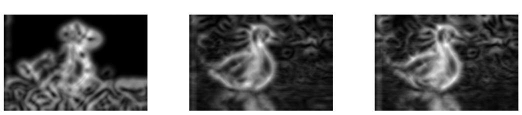

Fig. 23: Gradients for Image 4

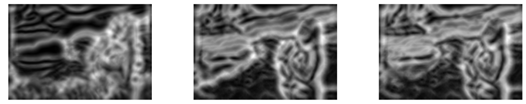

Fig. 24: Gradients for Image 5

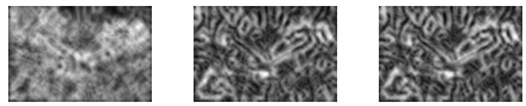

Fig. 25: Gradients for Image 6

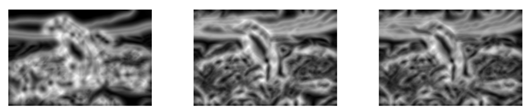

Fig. 26: Gradients for Image 7

Fig. 27: Gradients for Image 8

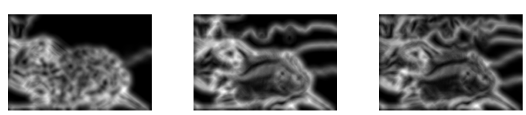

Fig. 28: Gradients for Image 9

Comparison between Canny Baselines, Sobel Baselines and Pblite Outputs for all Images in the Test Set

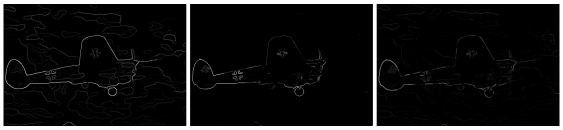

Fig. 29: Pblite Output for Image 1

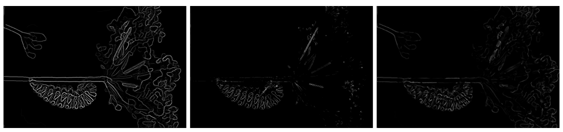

Fig. 30: Pblite Output for Image 2

Fig. 31: Pblite Output for Image 3

Fig. 32: Pblite Output for Image 4

Fig. 33: Pblite Output for Image 5

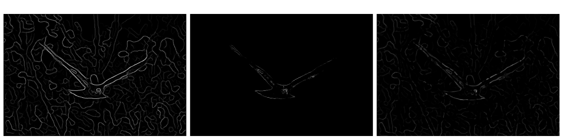

Fig. 34: Pblite Output for Image 6

Fig. 35: Pblite Output for Image 7

Fig. 36: Pblite Output for Image 8

Fig. 37: Pblite Output for Image 9

II. Deep Learning for CIFAR-10 Classification

A. Introduction

For image classification, we focus on the application and comparison of different neural network architectures using the CIFAR-10 dataset. The study included both basic and advanced architectures such as BasicNet1(custom architecture), BasicNet2(improvement on the custom architecture), ResNet(ResNet9), ResNeXt(ResNeXt9), and DenseNet. We analyzed the impact of various architectural choices on network performance, particularly looking at accuracy and computational efficiency. The study also involved testing different methods to enhance model accuracy, including data standardization, learning rate adjustment, and data augmentation.

B. BasicNet1

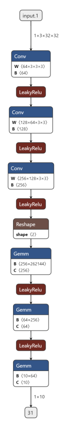

Fig. 38: BasicNet1 Model Architecture

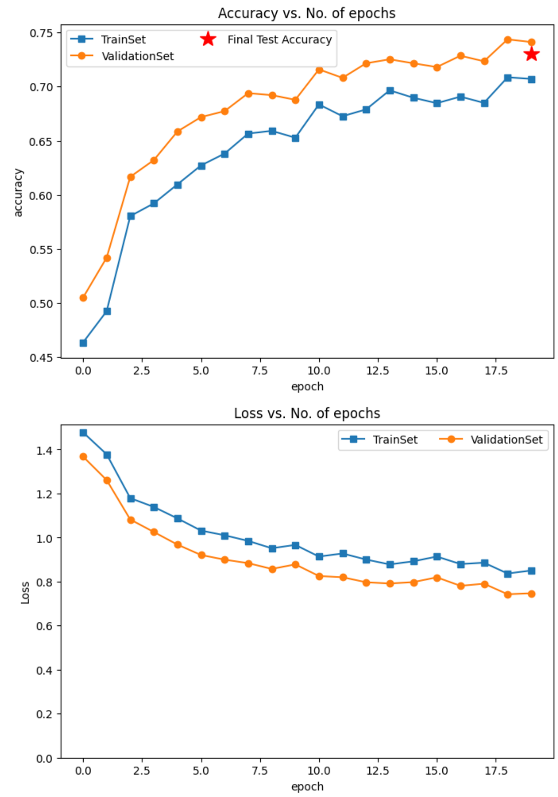

Fig. 39: Accuracy and Losses for BasicNet1

C. BasicNet2

Sr. No.	Hyperparameter	Value
1	Epochs	15
2	Learning Rate	1e-4
3	Batch Size	32
4	Optimizer	Adam
5	Weight Decay	1e-5

TABLE I: Hyperparameter Settings for BasicNet1

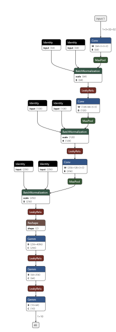

Fig. 40: BasicNet2 Model Architecture

Sr. No.	Hyperparameter	Value
1	Epochs	30
2	Learning Rate	1e-4
3	Batch Size	32
4	Optimizer	Adam
5	Weight Decay	1e-5

TABLE II: Hyperparameter Settings for BasicNet2

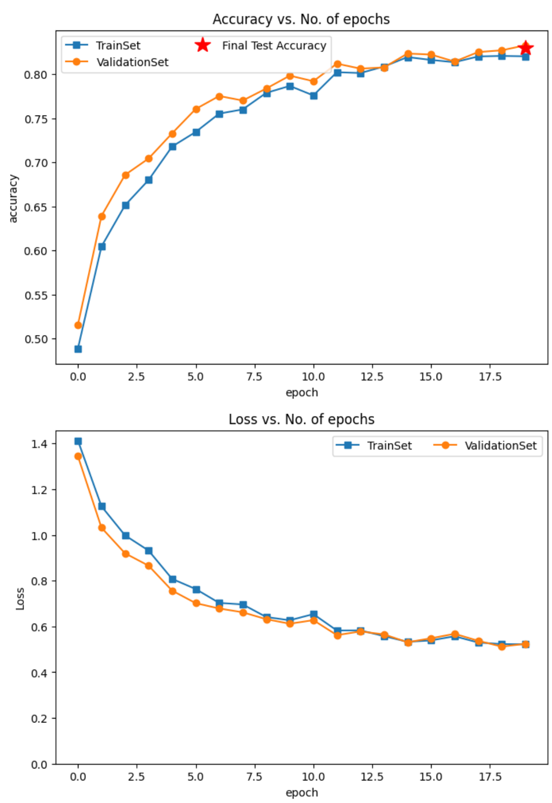

Fig. 41: Accuracy and Losses for BasicNet2

```
Number of parameters in this model are 27
100%                                      50000/50000 [01:25<00:00, 485.94it/s]
[4190   29  109   36   78   17   29   73  333  106] (0)
[  49 4338   12   11    8   10   14   15   89  454] (1)
[ 283    6 3478   91  500  159  266  144   56   17] (2)
[ 113   11  234 2785  476  766  291  238   50   36] (3)
[  65    1   97   64 4439   50  118  135   23    8] (4)
[  28    6  140  439  303 3685   79  291   13   16] (5)
[  16    9  148   94  133   41 4471   34   37   17] (6)
[  28    4   57   71  336  134    5 4331   10   24] (7)
[ 132   62   18   18   19    8   13   23 4632   75] (8)
[  87   95   21   14   18   15   18   78   92 4562] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 81.822 %
```

Fig. 42: BasicNet2 TrainSet Confusion Matrix

```
Number of parameters in this model are 27
100%                                      10000/10000 [00:13<00:00, 765.26it/s]
[844  10  34   6  13   1   4  14  54  20] (0)
[  6 899   6   3   1   2   1   2   9  71] (1)
[ 51   1 724  24  89  29  57  17   5   3] (2)
[ 18   4  60 556  98 138  53  52  12   9] (3)
[  6   0  21  14 895  10  30  22   2   0] (4)
[  8   2  26  81  64 759  11  45   2   2] (5)
[  5   2  32  21  17   9 902   4   5   3] (6)
[ 10   0  19  10  58  21   2 873   3   4] (7)
[ 41   8   7   7   5   1   1   4 916  10] (8)
[ 15  20   3   6   0   2   4   7  15 928] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 82.96 %
```

Fig. 43: BasicNet2 TestSet Confusion Matrix

D. *ResNet*

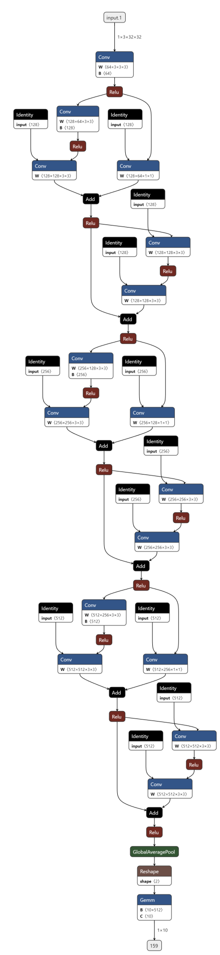

Fig. 44: ResNet Model Architecture

Sr. No.	Hyperparameter	Value
1	Epochs	30
2	Learning Rate	1e-4
3	Batch Size	32
4	Optimizer	Adam
5	Weight Decay	1e-5

TABLE III: Hyperparameter Settings for ResNet

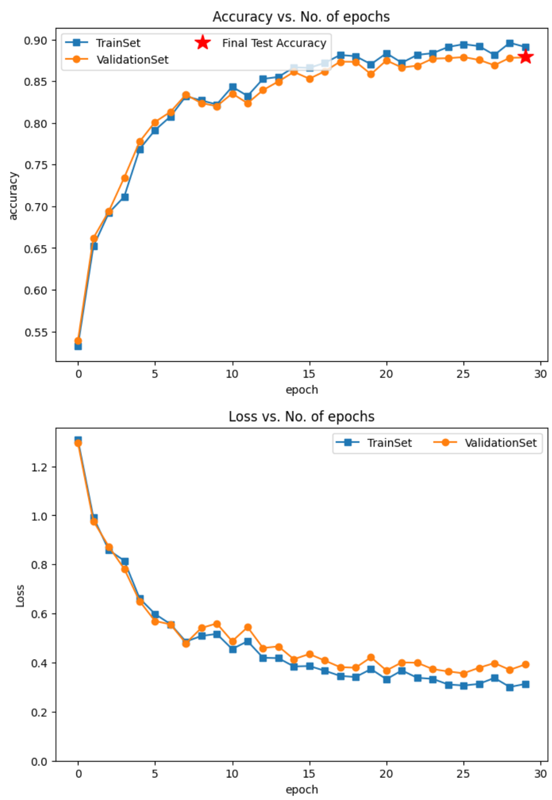

Fig. 45: Accuracy and Losses for Resnet

```
Number of parameters in this model are 98
100% 50000/50000 [03:03<00:00, 290.05it/s]
[4614   22  111   41   57    6   24   30   46   49] (0)
[  20 4812   11   10    8    8   13    4   28   86] (1)
[ 152    7 4433   85  142   60   83   29    6    3] (2)
[  66    9  170 4080  174  319   80   91    3    8] (3)
[  21    2   69   72 4733   26   28   46    2    1] (4)
[  16    4  109  504  166 4028   24  141    1    7] (5)
[  11    6  129  156  112   61 4503   19    0    3] (6)
[  29    8   64   71  180   60    6 4568    4   10] (7)
[ 262   49   69   41   44   36   39   10 4387   63] (8)
[  88  168   28   35   11   14   14   19   32 4591] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 89.498 %
```

Fig. 46: ResNet TrainSet Confusion Matrix

```
Number of parameters in this model are 98
100% 10000/10000 [00:36<00:00, 235.26it/s]
[902   3  42  10  12   2   4   3  16   6] (0)
[  5 946   4   3   2   2   1   0   4  33] (1)
[ 31   0 870  23  39  16  14   3   1   3] (2)
[ 11   3  42 811  36  62  15  12   2   6] (3)
[  3   2  17  18 940   5   8   7   0   0] (4)
[  6   0  27 111  40 796   6  14   0   0] (5)
[  5   0  43  55  23  13 859   1   0   1] (6)
[  7   0  20  13  49  24   1 883   0   3] (7)
[ 65  10  17  12   5   5   7   0 867  12] (8)
[ 13  36   3   3   1   3   4   2  10 925] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 87.99 %
```

Fig. 47: ResNet TestSet Confusion Matrix

E. ResNeXt

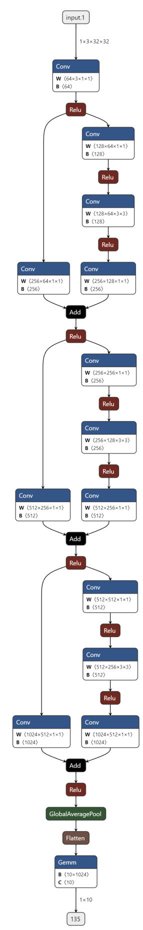

Fig. 48: ResNeXt Model Architecture

Sr. No.	Hyperparameter	Value
1	Epochs	30
2	Learning Rate	1e-4
3	Batch Size	32
4	Optimizer	Adam
5	Weight Decay	1e-5

TABLE IV: Hyperparameter Settings for ResNeXt

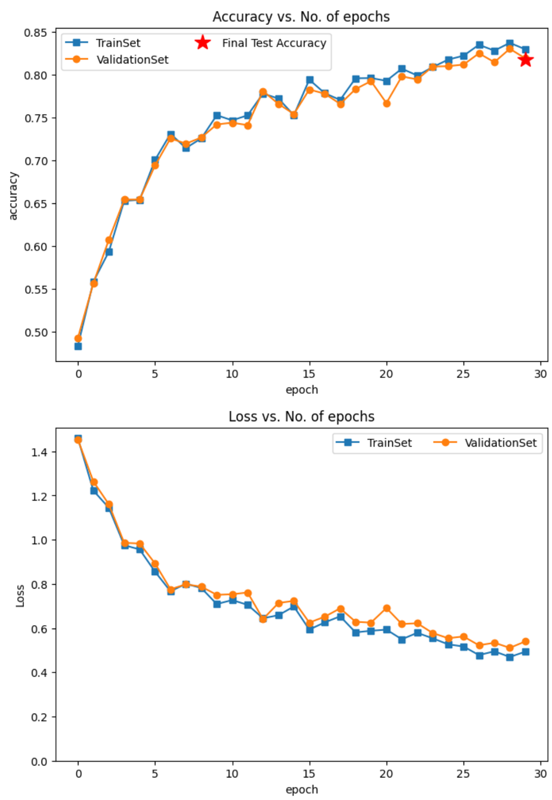

Fig. 49: Accuracy and Losses for Resnext

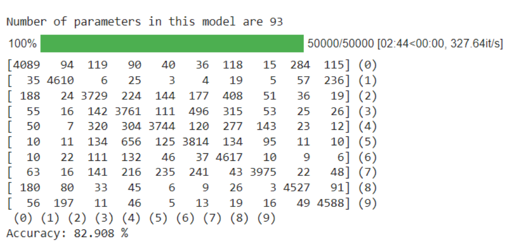

Fig. 50: ResNeXt TrainSet Confusion Matrix

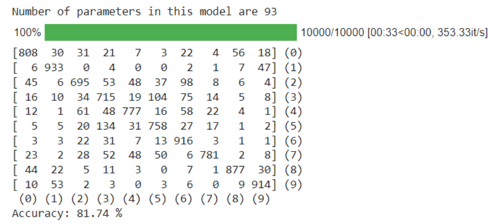

Fig. 51: ResNeXt TestSet Confusion Matrix

F. DenseNet

Sr. No.	Hyperparameter	Value
1	Epochs	30
2	Learning Rate	1e-4
3	Batch Size	32
4	Optimizer	Adam
5	Weight Decay	1e-5

TABLE V: Hyperparameter Settings for DenseNet

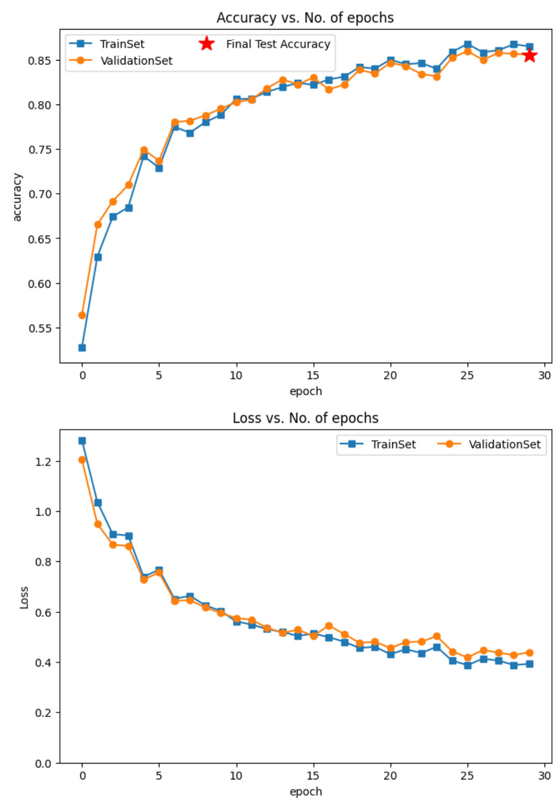

Fig. 52: Accuracy and Losses for Densenet

```
Number of parameters in this model are 527
100% 50000/50000 [12:55<00:00, 68.19it/s]
[4273  109   93   49   44   19   28   52  228  105] (0)
[  17 4878    7   11    0    5    7    1   13   61] (1)
[ 209   20 3772  192  197  207  156  180   45   22] (2)
[  47   16   98 3820  130  592  102  140   25   30] (3)
[  31    5   55  125 4313  102   59  264   36   10] (4)
[   9    8   52  441  108 4128   17  220    7   10] (5)
[  26   23   84  267   99   90 4364   31   11    5] (6)
[  11    6   28   96   75   98    9 4641   17   19] (7)
[ 102  166   19   17    8   17   17   12 4599   43] (8)
[  30  375    7   14    4    7   13   29   75 4446] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 86.468 %
```

Fig. 53: DenseNet TrainSet Confusion Matrix

```
Number of parameters in this model are 527
100% 10000/10000 [02:36<00:00, 56.83it/s]
[821  23  17  12  13   1   6  18  64  25] (0)
[  1 977   0   0   0   0   3   1   2  16] (1)
[ 43   1 728  32  59  45  39  34  13   6] (2)
[ 13   6  14 719  30 158  17  24  11   8] (3)
[  4   4  14  17 866  23  23  41   5   3] (4)
[  2   4   9  55  28 862   3  32   3   2] (5)
[  7   5  19  62  19  29 849   7   3   0] (6)
[  7   4   6  14  13  23   1 923   4   5] (7)
[ 31  37   4   6   1   0   1   1 911   8] (8)
[  6  81   2   1   0   3   0   2  12 893] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 85.49 %
```

Fig. 54: DenseNet TestSet Confusion Matrix

G. Number of Parameters

Model	Number of Parameters
BasicNet1	67,497,034
BasicNet2	1,437,642
ResNet	11,025,994
ResNeXt	3,270,794
DenseNet	342,340

TABLE VI: Comparison Between Number of Model Parameters in Each Architecture

H. Architectural Comparisons

- **(BasicNet1)**: This model is a straightforward convolutional neural network comprising three convolutional layers with increasing filter sizes (64, 128, 256). Each convolutional layer is followed by a leaky ReLU activation function, which helps the model learn non-linear relationships in the data. The absence of pooling or normalization layers makes this model less complex but could potentially limit its ability to generalize across varied datasets. It concludes with a series of linear layers that condense the high-dimensional feature maps into a final output for classification.
- **(BasicNet2)**: Building upon the design of Basicnet1, this model introduces batch normalization and max pooling layers in each convolution block. Batch normalization helps in stabilizing the learning process and normalizing the output of each convolution layer, which can lead to faster convergence and improved overall performance. Max pooling is used for reducing the spatial dimensions of the feature maps, which not only helps in reducing the computational load but also aids in achieving some level of translational invariance.
- **ResNet**: ResNet, or Residual Network, introduces a revolutionary concept of shortcut connections that skip one or more layers. These connections allow the gradient to flow directly through the network, addressing the problem of vanishing gradients in deep networks. ResNet's design enables the training of substantially deeper networks than was previously feasible. Each residual block in ResNet is a mini-network with convolution, batch normalization, and ReLU layers, and these blocks are stacked to form the complete architecture. ResNet is particularly effective in learning identity functions, ensuring that the added layers can at least maintain the performance of the network, if not improve it.
- **ResNeXT**: ResNeXT is an extension of the ResNet architecture, introducing the concept of grouped convolutions. This means that instead of a single set of filters being applied in the convolutional layer, the layer has multiple sets (or groups) of filters, with each set processing a subset of input channels. This cardinality (the number of groups) adds a new dimension to the network's architecture, allowing it to learn more complex features. ResNeXT manages to strike a balance between increasing the model's capacity and its complexity, often resulting in improved performance on various benchmarks.
- **DenseNet**: DenseNet, short for Densely Connected Convolutional Networks, is unique in the way each layer connects to every other layer in a feed-forward fashion. In DenseNet,

each layer receives feature maps from all preceding layers, concatenates them, and passes its feature map to all subsequent layers. This architecture leads to substantial feature reuse, which makes the network more parameter-efficient. Transition layers, consisting of batch normalization, convolution, and pooling, are placed between dense blocks to control the growth of the feature map sizes and to improve computational efficiency.

- **General Comparison**: When comparing these architectures, DenseNet stands out for its parameter efficiency and feature reuse capabilities, making it well-suited for tasks where model size and memory footprint are crucial. ResNet and ResNeXT are more adept at training deeper networks due to their shortcut connections, which alleviate the vanishing gradient problem. BasicNet1 and BasicNet2, being less complex, might be preferable for smaller datasets or when computational resources are limited, though they might not perform as well on more complex tasks. The choice among these architectures depends on a variety of factors including the complexity and size of the dataset, computational constraints, and specific requirements of the task at hand. Accuracy for 30 epochs is comparable for the rest, except DenseNet performs better than the rest.

III. Acknowledgment

The author would like to thank Prof. Nitin Sanket and the TA of this course RBE549- Computer Vision.

References

[1] RBE549 - Computer Vision Website**Link Link**