Pb-Lite Edge detection and Deep-Feature Learning

Mayank Deshpande
University of Maryland
College Park, MD
Email: msdeshp4@umd.edu

***Abstract*—This report contains detailed results for the implementation of pb-lite algorithm and multiple neural network architectures for edge detection.**

I. Phase I: Shake my Boundary

In the phase I of the homework the aim was to implement a lite version of the pb boundary detection algorithm. The pb boundary detection algorithm outputs a per-pixel probablity of an edge. This algorithm tends to outperform the classical edge detection algorithms like canny and sobel by considering texture and color discontinuities in an image and decreasing the overall false positives. There are 4 steps in the implementing this algorithm: Generate Filter banks $->$ Compute Texton, brightness and color maps $->$ Compute Texton, Brightness and color gradients $->$ Compute pb-lite output.

A. Filter Banks

The Filter banks are a set of filters which when convoluted over the image generate sets of features. In this project 3 filter banks are used for generating the textural features: 1) Oriented DoG filters, 2) Leung-Malik Filter and 3) Gabor filter.

B. Oriented DoG Filters

These filters are simple Derivative of Gaussian filters (DoG) in certain scales and Orientations. These filters can be created by simply convolving the sobel operator over a gaussian kernel and rotating these filters based on the number of required filters. The sobel operators in x and y directions are convoluted with a Gaussian kernel and then to generate the DoG the following formula has been used:

$$DoG = |G_x| * \cos(\theta) + |G_y| * \sin(\theta)[1]$$

where G_x and G_y are first derivatives of Gaussian along x and y axis respectively, θ is the orientation of filter (in degrees). Fig.1 shows the Oriented DoG filterbank.

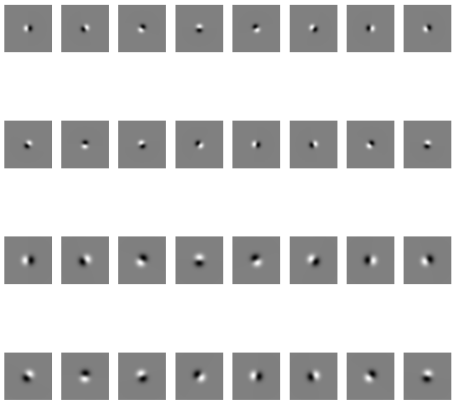

Fig. 1. Oriented DoG Filter Bank

C. Leung-Malik Filters

The Leung-Malik filters are a set of multi-scale, multi-oriented filter bank with 48 filters. These can classified into two type LM-small and LM-large based on the scale factors used for generating the filters. The LM filters are generated by using first and second order DoG filters, Laplacian of Gaussian and Gaussian filters. Initially the first order DoG filters were computed by convolution of the 2D Gaussian kernel with elongation factor of 3 with the sobel operators and these first order DoG filters were then convoluted again with the sobel operators to obtain the second order DoG filters. To obtain the Laplacian of Gaussian filters the 2D gaussian kernel were convoluted with the Laplacian filter over the 8 scales and the gaussian filters were computed over the base scales. LML filters are computed the same way but the scales $\sqrt{2}$ times the LMS base scales. Fig.2 shows the LMS, LML and LM filters.

D. Gabor Filters

A gabor filter is basically sinusoidal signal of particular frequency and orientation, modulated by a Gaussian wave [2]. Fig.3 shows the Gabor filters generated.

E. Texton Map

Texton map can be seen as the texture encoding of the image which we get from the filters that we discussed above. We can use any combination of filter banks to generate the texton map, here we have used all three filter banks which gives us a vector of filter responses. This vector of filter response associated with each pixel contains the texture information of that filter. We use K-means clustering (K = 64) to cluster together the pixel with similar responses and the output of this k-means clustering is the Texton map (τ).

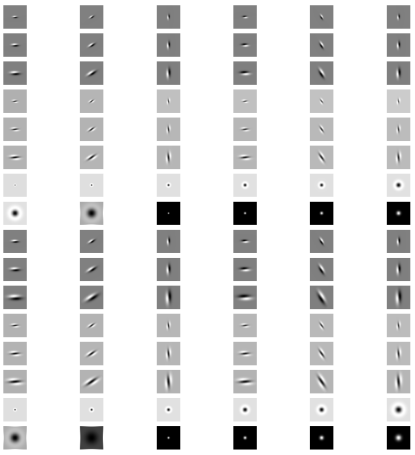

Fig. 2. Leung-Malik Filter Bank

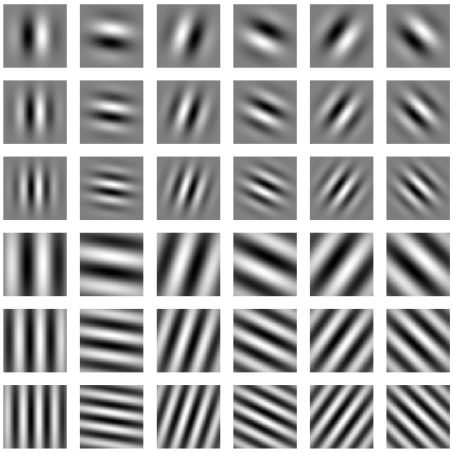

Fig. 3. Gabor Filter Bank

F. Brightness Map

The image's brightness values are grouped using k-means clustering, with a parameter K set to 16. The Brightness Map indicates variations in light intensity for each pixel.

G. Color Map

The image's color values are grouped using k-means clustering, with a parameter K set to 16. The Color Map denotes the RGB color properties present in each pixel.

H. Texton, Brightness and Color Gradient

In the preceding section, the Texton, Color, and Brightness maps were generated to represent each feature at every pixel.

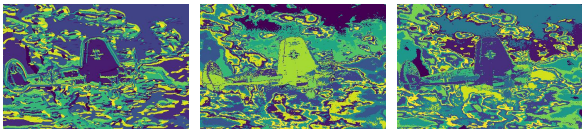

Fig. 4. Texton, Brightness and Color Map for Image 1

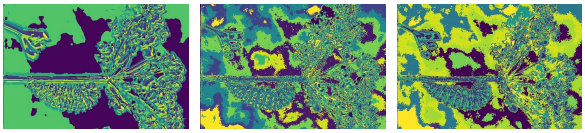

Fig. 5. Texton, Brightness and Color Map for Image 2

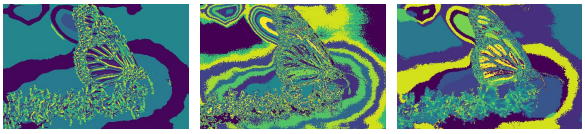

Fig. 6. Texton, Brightness and Color Map for Image 3

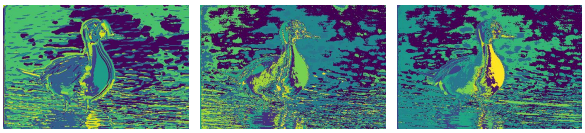

Fig. 7. Texton, Brightness and Color Map for Image 4

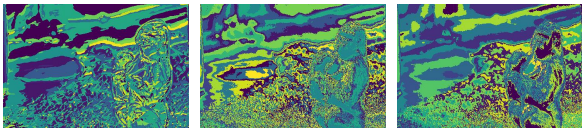

Fig. 8. Texton, Brightness and Color Map for Image 5

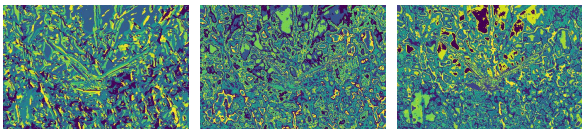

Fig. 9. Texton, Brightness and Color Map for Image 6

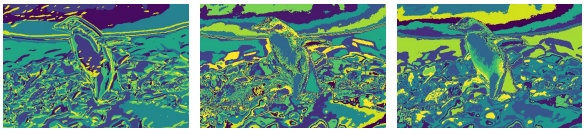

Fig. 10. Texton, Brightness and Color Map for Image 7

Gradient measurement is then conducted to assess the degree of change in the distribution of all features at a given pixel. To achieve this, binary half-disc masks are employed. These masks consist of binary images depicting half circles with varying scales and rotation values. Gradients are produced by

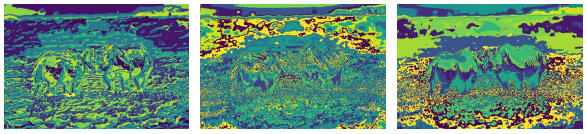

Fig. 11. Texton, Brightness and Color Map for Image 8

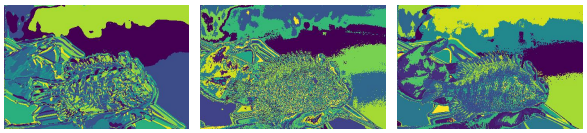

Fig. 12. Texton, Brightness and Color Map for Image 9

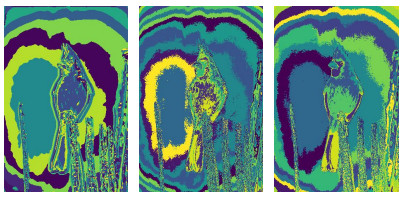

Fig. 13. Texton, Brightness and Color Map for Image 10

convolving these masks with the map generated in the previous step. The chi-squared distance between the mask pair and the binary image is calculated. This distance metric is utilized for comparing two histograms and is computed as follows:

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

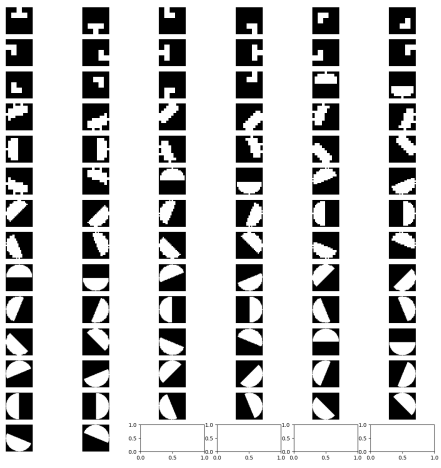

Fig. 14. Half Disc Masks

I. pbLite Output

In the last stage, the features extracted from the baseline methods (Canny and Sobel operators) were merged with the gradients of τ, β, and C using the equation provided.

$$PbEdges = \frac{t+g+b}{3} \circ (w_1 * cannyPb + w_2 * sobelPb)$$

The symbol $\circ$ denotes the Hadamard product operator, which signifies element-wise multiplication between two matrices. The weights w1 and w2 are both set to 0.5, resulting in a simple average for the weighted average. The output of the pb-lite algorithm is then displayed alongside the Sobel and Canny baseline results.

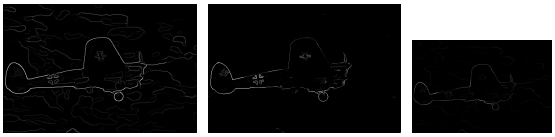

Fig. 15. Canny, Sobel and pb-lite outputs of Image 1

Fig. 16. Canny, Sobel and pb-lite outputs of Image 2

Fig. 17. Canny, Sobel and pb-lite outputs of Image 3

Fig. 18. Canny, Sobel and pb-lite outputs of Image 4

Fig. 19. Canny, Sobel and pb-lite outputs of Image 5

Fig. 20. Canny, Sobel and pb-lite outputs of Image 6

Fig. 21. Canny, Sobel and pb-lite outputs of Image 7

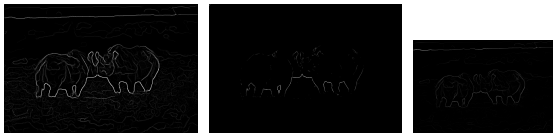

Fig. 22. Canny, Sobel and pb-lite outputs of Image 8

Fig. 23. Canny, Sobel and pb-lite outputs of Image 9

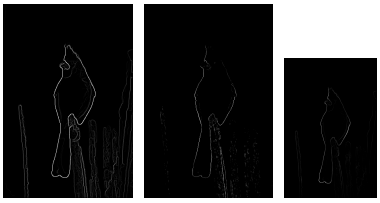

Fig. 24. Canny, Sobel and pb-lite outputs of Image 10

J. Conclusion

The weights assigned to the Canny and Sobel detectors are determined through a process of trial and error. It's observable that increasing the weight of the Canny edge detector strengthens the edges significantly, but it also tends to suppress the background entirely, whereas the Sobel detector maintains the overall pixel intensity. From this, we can deduce that the Pb-Lite detector outperforms other detectors, as it offers greater control over various features crucial for edge detection.

II. Phase2: Deep Dive on Deep Learning

In the second phase of this assignment, you'll be constructing several neural network architectures and assessing them based on different criteria such as parameter count, accuracy's on training and testing sets. Additionally, you'll offer a comprehensive analysis to explain why one architecture outperforms another.

A. Basic Neural Network

In the basic neural network there are total of four layers: 2 conv layers followed by two fully connected layers. The model was trained on CIFAR 10 dataset with a minibatch size of 64. The softmax. The optimization method employed is Adam's optimizer, and the Softmax function is utilized to compute the loss. Fig. shows the architectural diagram of the model.

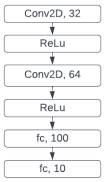

Fig. 25. Architecture Diagram of Base Model

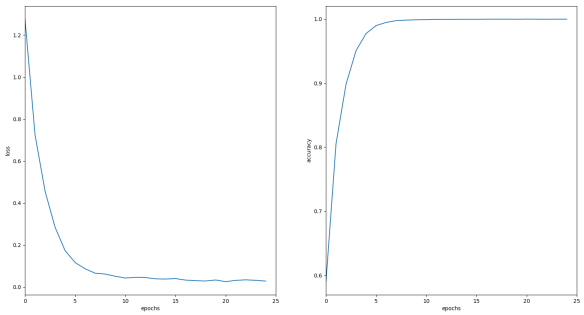

Fig. 26. Loss vs. Epochs and Train accuracy vs. Epochs for Basic model

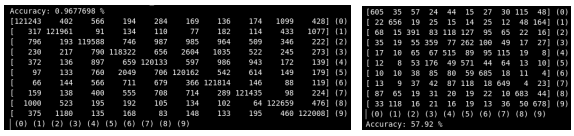

Fig. 27. Train and test confusion matrix for basic model

B. Modified Neural Network

The modified neural network architecture consists of three convolutional layers with increasing numbers of filters followed by batch normalization layers and ReLU activations. The convolutional layers are designed to capture hierarchical features from the input images. Subsequently, the feature maps are flattened, and two fully connected layers are applied, followed by dropout regularization and batch normalization to prevent overfitting. Finally, the output layer produces logits, which are then converted to probabilities using softmax activation for classification.

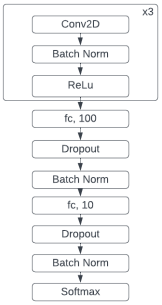

Fig. 28. Architecture Diagram of Modified Model

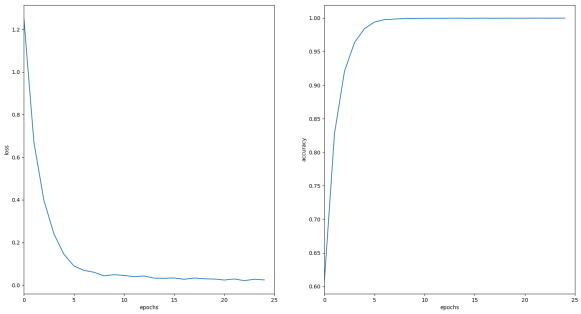

Fig. 29. Loss vs. Epochs and Train accuracy vs. Epochs for modified model

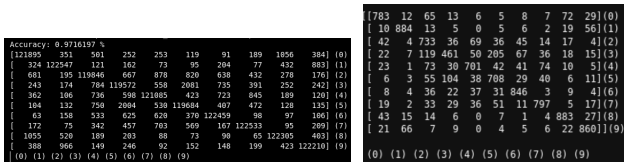

Fig. 30. Train and test confusion matrix for modified model

C. ResNet

ResNet, short for Residual Neural Network, is a deep learning architecture renowned for its ability to tackle the vanishing gradient problem in very deep neural networks. It introduces skip connections, allowing the network to learn residual functions instead of directly learning desired mappings. This enables effective training of extremely deep networks, leading to improved performance in various tasks such as image recognition and classification.

D. ResNext

ResNeXt is an extension of the ResNet architecture that introduces a ”cardinality” parameter, which controls the width of the network. By using grouped convolutions with multiple cardinality values, ResNeXt achieves state-of-the-art performance on various computer vision tasks while maintaining model efficiency. It allows for more flexible and scalable

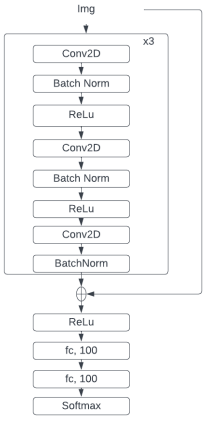

Fig. 31. Architecture Diagram of ResNet

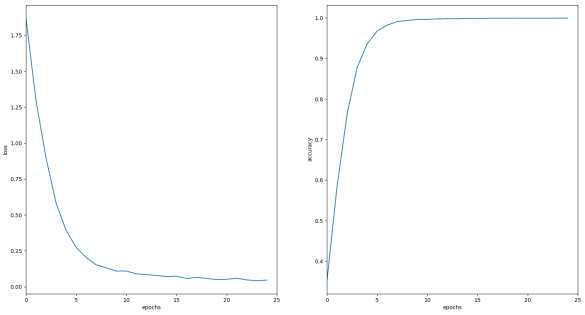

Fig. 32. Loss vs. Epochs and Train accuracy vs. Epochs for ResNet

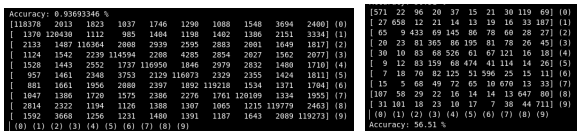

Fig. 33. Train and test confusion matrix for ResNet

architectures by leveraging parallel paths within each layer to capture diverse features effectively.

E. DenseNet

DenseNet, short for Dense Convolutional Network, is a deep learning architecture where each layer is connected to every other layer in a feed-forward fashion. It fosters feature reuse by concatenating feature maps from all previous layers, creating dense connections between layers. This results in highly efficient parameter usage and encourages feature propagation throughout the network, leading to improved gradient flow, feature learning, and ultimately, better performance on tasks such as image classification and segmentation.

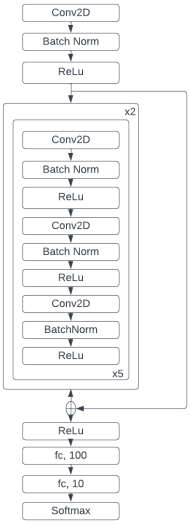

Fig. 34. Architecture Diagram of ResNext

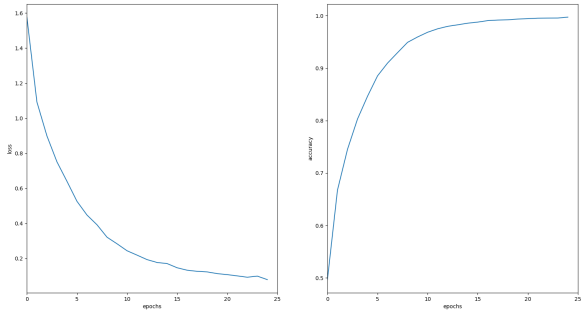

Fig. 35. Loss vs. Epochs and Train accuracy vs. Epochs for ResNext

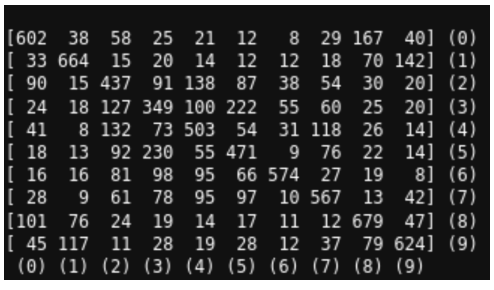

Fig. 36. Train and test confusion matrix for ResNext

III. Conclusion

The PB Lite algorithm was utilized for image edge detection, and I successfully trained and tested my initial neural network to address a classification task using the CIFAR10 dataset, incorporating adjustments to enhance its accuracy.

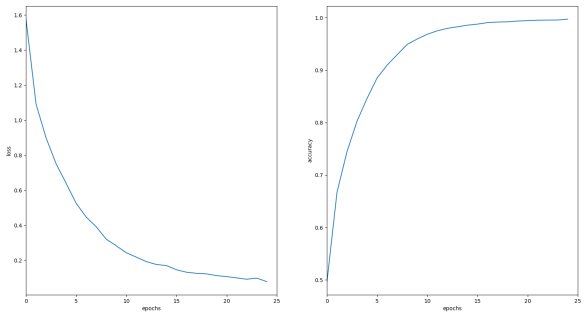

Fig. 37. Loss vs. Epochs and Train accuracy vs. Epochs for DenseNet

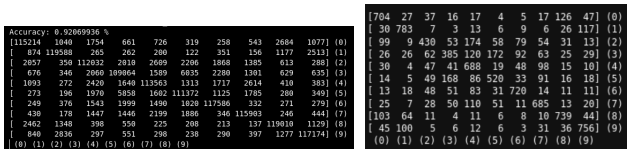

Fig. 38. Train and test confusion matrix for DenseNet

Model	parameters	Train Accuracy	Test Accuracy
Basic	4002569	96.77%	57.92%
Modified	3156988	97.16%	59.21%
Resnet	6790774	93.69%	56.51%
ResNext	8611274	92.88%	54.7%
DenseNet	3896546	92.06%	60.89%

The table presents a comparison of various models based on their parameters, training accuracy, and test accuracy. The basic model, with 4,002,569 parameters, achieves a training accuracy of 96.77% but exhibits a lower test accuracy of 57.92%, possibly indicating overfitting. In contrast, the modified model, with 3,156,988 parameters, demonstrates slightly improved training and test accuracies of 97.16% and 59.21%, respectively. Resnet, ResNext, and DenseNet models, with parameter counts of 6,790,774, 8,611,274, and 3,896,546, respectively, show varying degrees of performance. Resnet and ResNext models exhibit lower accuracies despite their higher parameter counts, suggesting potential over-complexity, while DenseNet, with fewer parameters, achieves a higher test accuracy of 60.89%, indicating its effectiveness in this context.

References

[1] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).
[2] $https://medium.com/@anujshah/through-the-eyes-of-gabor-filter-17d1fdb3ac97$
[3] $https://github.com/tonyjo/LMfilterbankpython$
[4] https://blog.paperspace.com/popular-deep-learning-architectures-densenet-mnasnet-shufflenet/