# CMSC733 HW0: Alohomora

Vikram Setty
University of Maryland, College Park
Email: vikrams@umd.edu

***Abstract*—This report contains the premise, methodology, results, and discussions for the Homework 0: Alohamora assignment of the course Computer Processing of Pictorial Information (CMSC733) offered by the Computer Science Department at the University of Maryland. The first phase contains an overview of a probability-based edge detection (pb-lite) implementation whereas the second implements, and then evaluates and compares the performance of various convolutional neural network architectures for image classification on the CIFAR-10 dataset.**

## I. Phase 1: Shake my Boundary

### A. Introduction

Edge detection, though well explored, is still not a completely solved or optimized task. Since edges in an image are subjective and can be examined through changes in depth, texture, brightness, and color, it is often observed that even well-known edge detection techniques fail to discover all relevant edges or add redundant/noisy edges to many image styles. This leads to the motivation to use techniques that use a combination of different baselines. The premise of the first phase of this report is exploring probability-based edge detection, described in detail on the CMSC733 HW0 webpage [1]) based on an assignment from Brown University [2]). Computing the PbLite edges uses gradients in texture, brightness, and color to assign probabilities to the presence of an edge detected by Canny and Sobel edge detection baselines. The following sections go over the various steps used in this pipeline tested on ten sample images to evaluate performance.

### B. Filter Banks

The first step in generating the gradients in the image texture is obtaining a set of filters called the filter bank, which collectively work to obtain different texture features on convolving with the grayscale version of the original image. The various filters used to make the filter bank are described in the following subsections

*1) Oriented Derivative of Gaussian (DoG) Filters:* The basic filters used in edge detection include the DoG Filter that is obtained by convolving a Gaussian Filter (which smooths the image) with Sobel Filters (which calculates intensity gradients in the horizontal and vertical directions). By rotating the basic DoG filter to different orientations at different filter sizes and scales (standard deviation of the Gaussian smoothing function on a pixel scale), we get a set of oriented DoG filters that are useful in bringing about different scales of gradient changes in various directions of the image. A total of 32 oriented DoG Filters are used as shown in Fig 1. Two scales ($\sigma$ = [2,3]) and sixteen orientations (equally spaced angles starting from 0) are used for generating the DoG Filters of size 21*21.

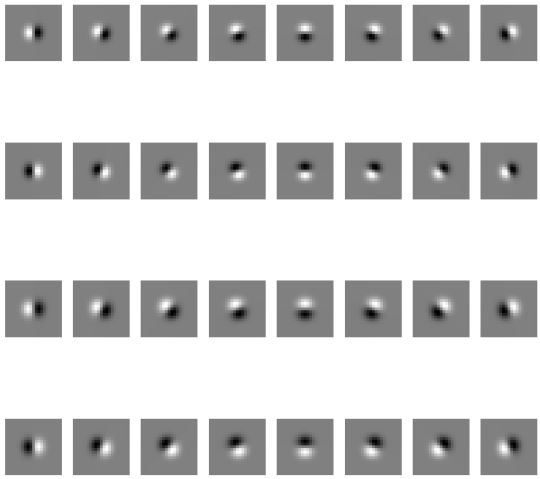


Fig. 1. Oriented Derivative of Gaussian (DoG) Filters

*2) Leung-Malik (LM) Filters:* The LM filter set contains 48 filters which consist of the first and second derivatives at six orientations (equally spaced angles starting at 0) and three scales of a Gaussian Filter (36 filters), Laplacian of Gaussian (LoG) Filters at eight scales (8 filters), and simple Gaussian Filters at four scales. The scales used in these filters come from a set $\sigma$ = [$\sigma_1$, $\sigma_2$, $\sigma_3$, $\sigma_4$]. The first and second derivative of Gaussians use [$\sigma_1$, $\sigma_2$, $\sigma_3$], whereas the LoG filters use [$\sigma_1$, $\sigma_2$, $\sigma_3$, $\sigma_4$, 3*$\sigma_1$, 3*$\sigma_2$, 3*$\sigma_3$, 3*$\sigma_4$], and the Gaussian filters use [$\sigma_1$, $\sigma_2$, $\sigma_3$, $\sigma_4$]. For generating our filter bank for pb-lite, two sets of LM Filters are used, the Leunf-Malik Small (LMS) Filter Set using $\sigma_S$ = [1,$\sqrt{2}$,2,2$\sqrt{2}$] and the Leung-Malik Large (LML) Filter Set using $\sigma_L$ = [$\sqrt{2}$,2,2$\sqrt{2}$,4], making a total of 96 filters (of size 21*21), as can be seen in Fig 2.

*3) Gabor Filters:* A Gabor Filter is a Gaussian Filter modulated by a sine wave at different frequencies and orientations. It is based on the human visual system and targets specific frequency variations in the texture of an image. Six orientations (equally spaced angles starting at 0), two scales ($\sigma$ = [10,25]), and three frequencies (f = [2/N,3/N,4/N], where N=21 is the filter size) are used to generate the set of Gabor Filters shown in Fig 3.

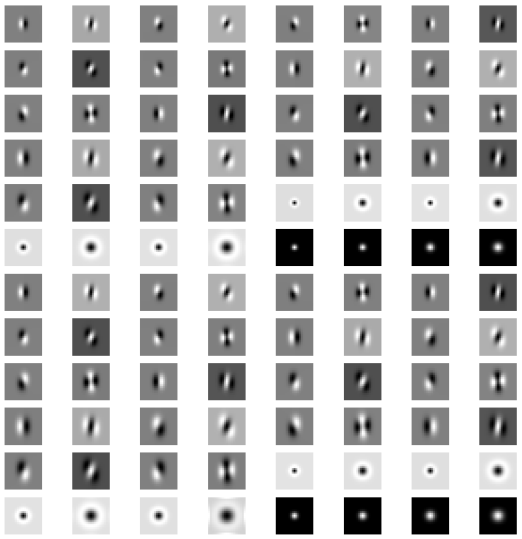


Fig. 2. Leung-Malik (LM) Filters (LM Small + LM Large)

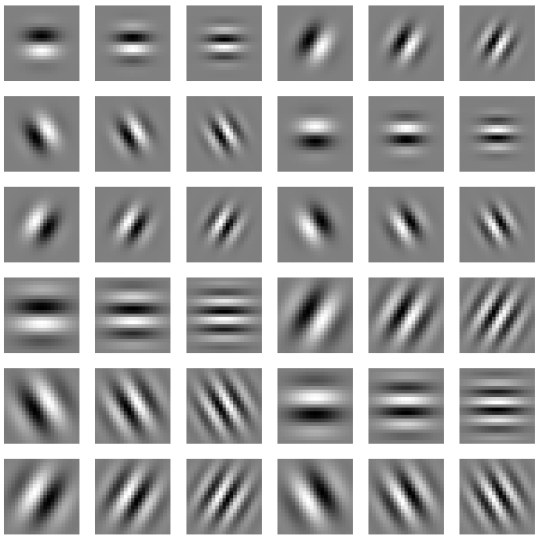


Fig. 3. Gabor Filters

### C. Texton, Brightness, and Color Maps

*1) Texton Maps ($\mathcal{T}$):* The texton map ($\mathcal{T}$) contains various texture properties of the image and can be obtained by convolving all the filters in the filter bank with the grayscale version of the image to get a new set of images, equal in number to the number of filters in the filter bank, each with dimensions equal to the original image. This can be visualized as each pixel of the grayscale image corresponding to a set of features of dimension equal to the number of filters. Using K-Means clustering with 64 clusters and two initialization runs, we can assign a specific cluster/bin to each pixel of the original image that is called the texton map. The texton maps of each image are shown in Fig 4(a)-Fig 13(a).

*2) Brightness Maps ($\mathcal{B}$):* Just as we obtained the texton map, we can also generate the brightness map $\mathcal{B}$ that contains brightness properties within the grayscale image. To do this, we follow a similar procedure to generating the texton map, except that we don't apply the filter banks and use one-dimensional information (grayscale pixel intensity value) to do K-Means clustering with 16 clusters/bins. The brightness maps of each image are shown in Fig 4(b)-Fig 13(b).

*3) Color Maps ($\mathcal{C}$):* Similar to the brightness map, a color map $\mathcal{C}$ displaying the color properties of an image can be obtained by performing a similar K-Means clustering operation with 16 clusters/bins on the original RGB image, thus using three-dimensional features for each pixel. The color maps of each image are shown in Fig 4(c)-Fig 13(c).

### D. Map Gradients ($\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$)

Upon generating the three (texton, brightness, and color) maps, it is important to calculate their gradients as the gradients in each of those properties are what contribute to the presence of an edge in the image. The gradients of the maps ($\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$) can be smartly computed by convolving half-disk filters/masks in pairs of opposite orientations over the map. The half-disk filters are generated at a filter size of 21*21, and radii of r=[2,5,10,20,30]. The set of half-disk filters/masks used can be seen in Fig 14.

The chi-square distance between the images is obtained by convolving each half-disk mask of a pair is calculated over all the clusters/bins of the map, summed up for all half-disk mask pairs, and averaged for each pixel to get a two-dimensional gradient map with the same shape as the original map. This operation is done for all three (texton, brightness, and color) of the maps and the way to calculate it is shown in Eq 1. The texton, brightness, and color map gradients maps of each image are shown in Fig 15-Fig 24.

$$\chi^2 = \frac{1}{2}\Sigma_{i=1}^{K}\frac{(g_i - h_i)^2}{g_i + h_i} \tag{1}$$

### E. Probability-based Edge Detection (Pb-lite)

Using the gradient maps of each of the three (texton, brightness, and color) maps along with the Canny and Sobel baseline outputs, we can calculate the probability-based edge detection (pb-lite) output using a Hadamard operator as shown in Eq 2. The net baseline used can be adjusted by weighting the Canny and Sobel baselines (with weights $w_1$ and $w_2$). Using a weight of 0.5 for each works well for a general-purpose edge detection task. We can also add weight to the texton, brightness, and color maps. Using an average of the three for each pixel works well in this scenario too. The Canny baseline,

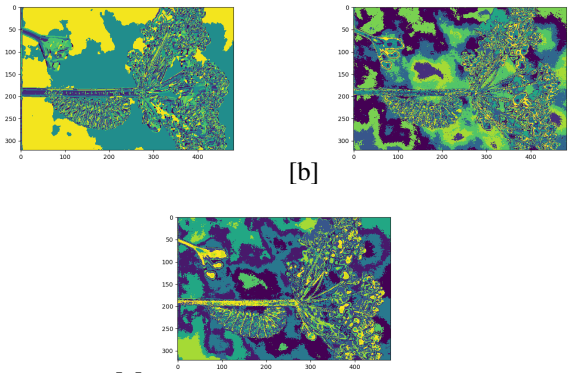



Fig. 4. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 1

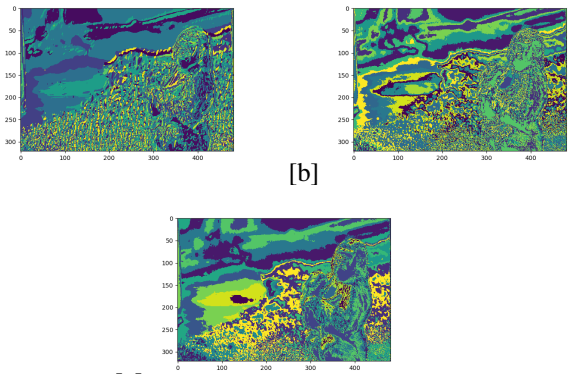



Fig. 5. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 2

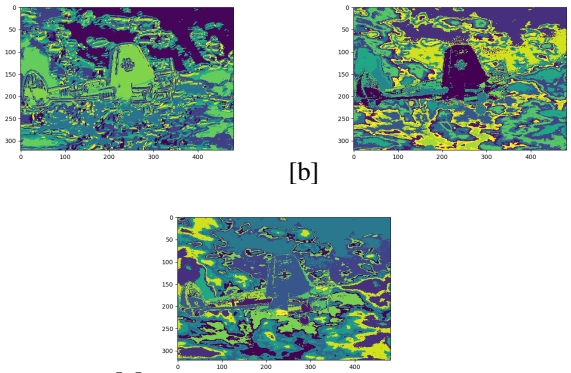



Fig. 6. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 3

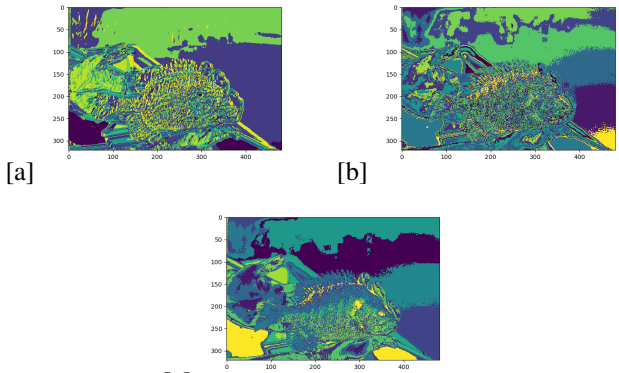



Fig. 7. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 4

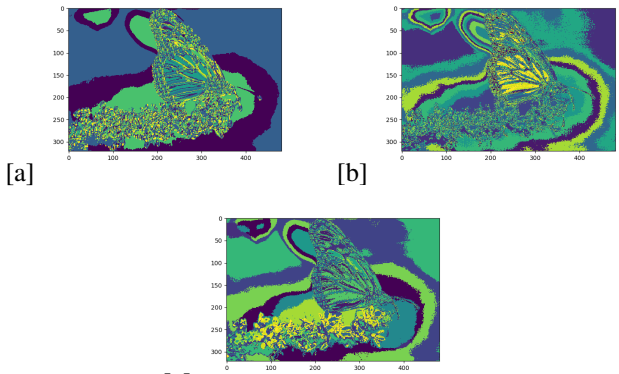



Fig. 8. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 5

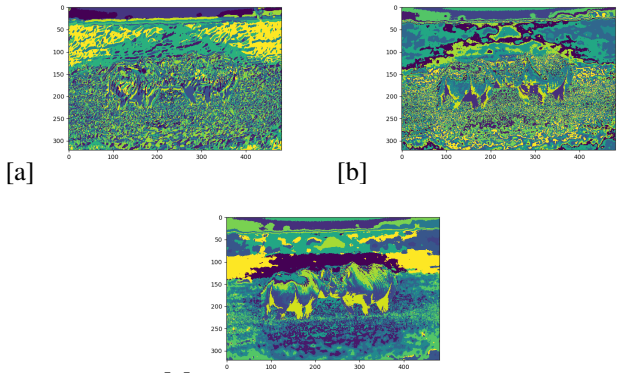



Fig. 9. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 6

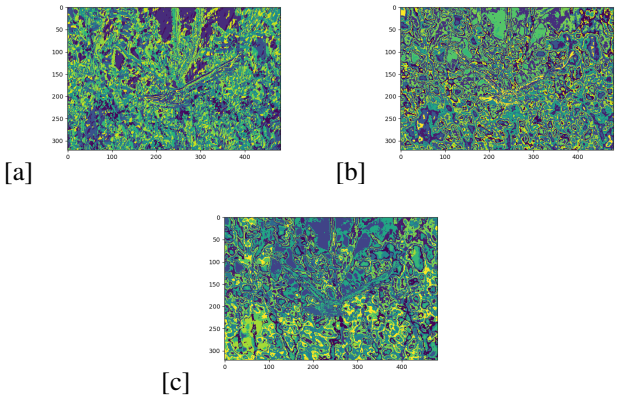



Fig. 10. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 7

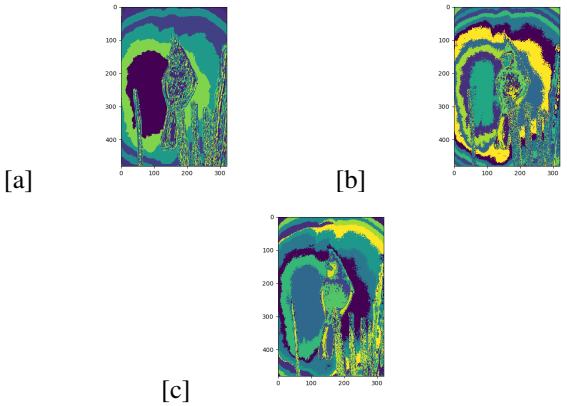



Fig. 11. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 8

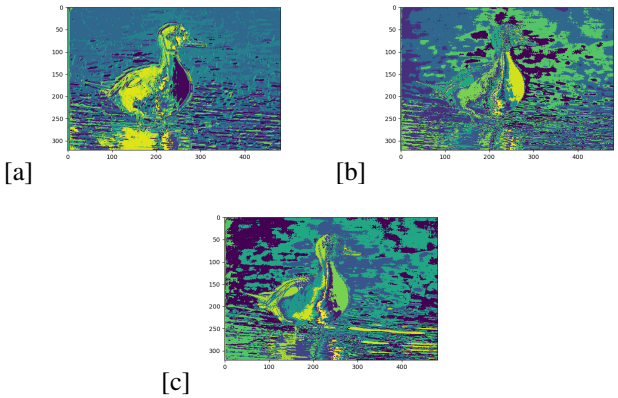



Fig. 12. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 9

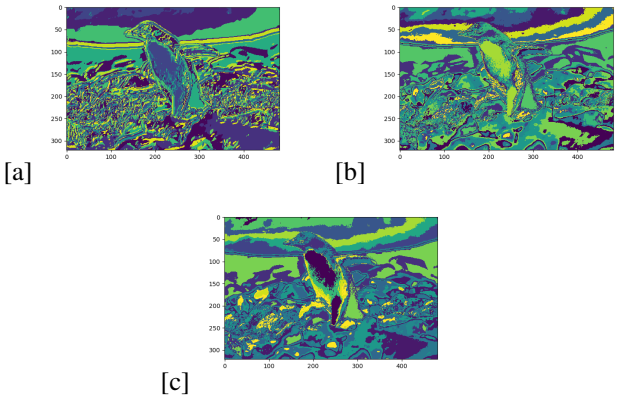



Fig. 13. $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{C}$ for Image 10

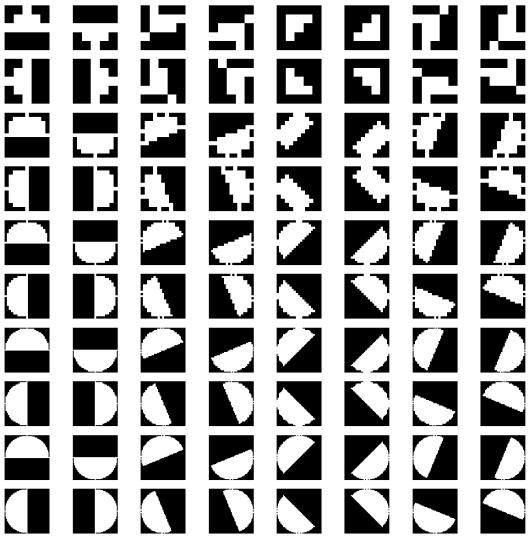


Fig. 14. Half Disk Masks

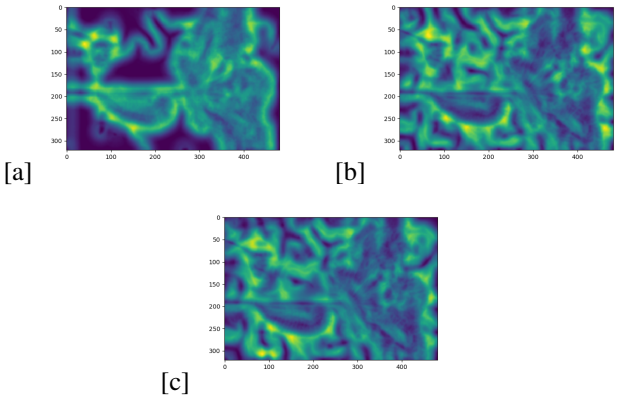
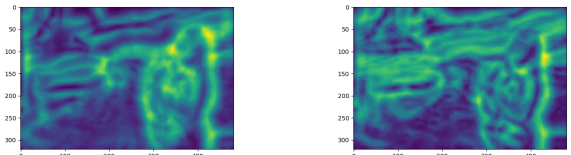



Fig. 15. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 1



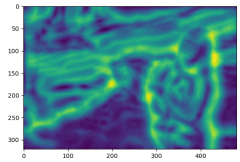
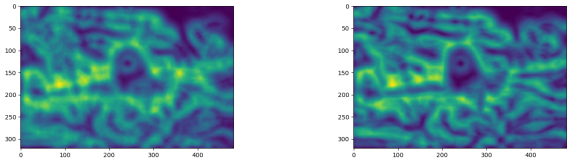




Fig. 16. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 2



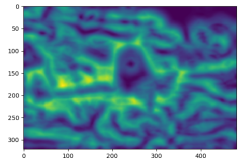
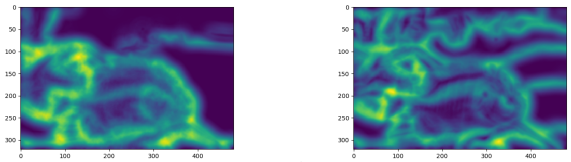




Fig. 17. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 3



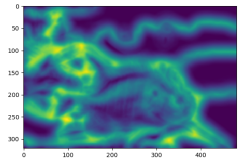
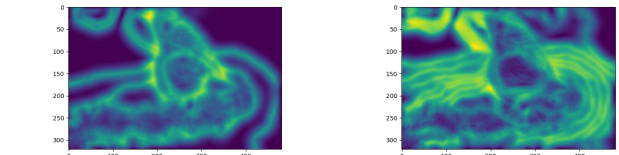




Fig. 18. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 4



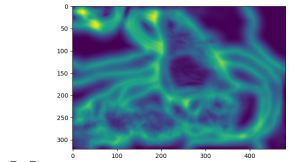
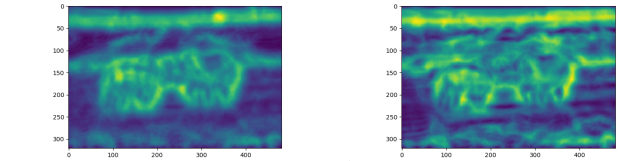




Fig. 19. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 5



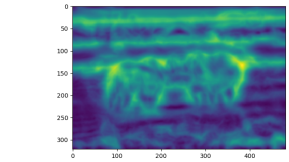
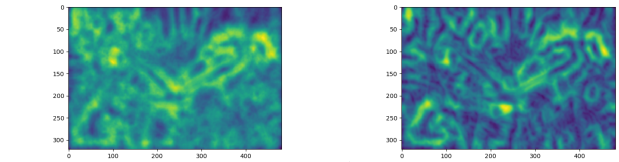




Fig. 20. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 6



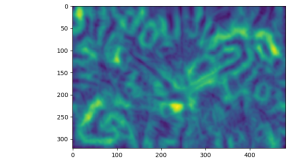




Fig. 21. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 7

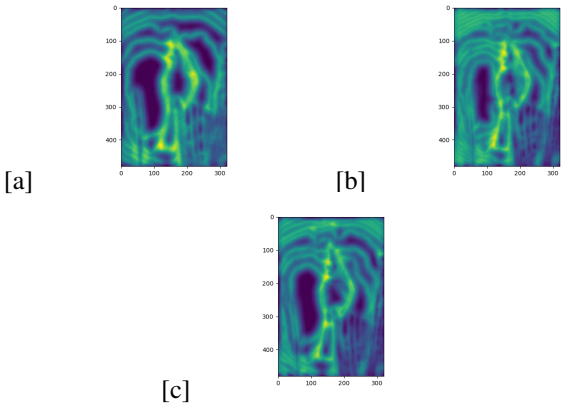


Fig. 22. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 8

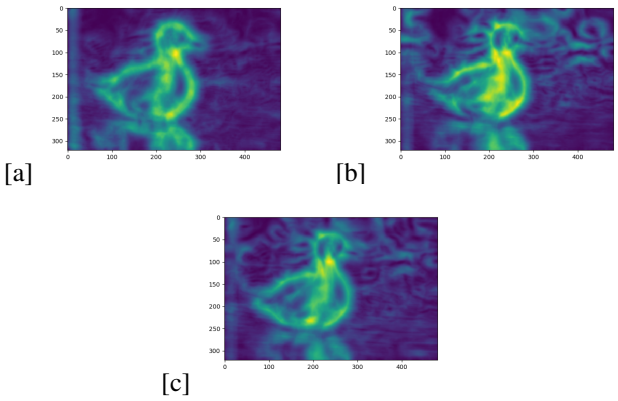


Fig. 23. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 9

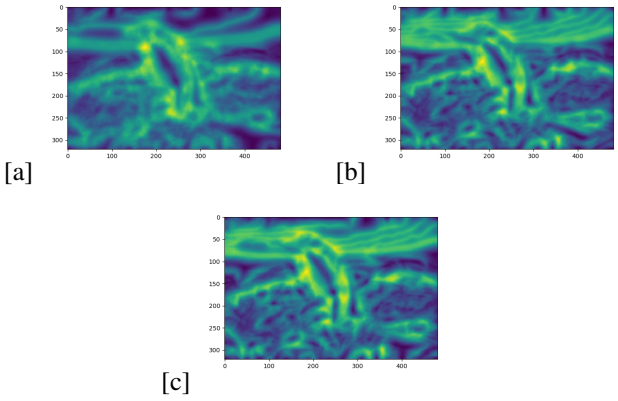


Fig. 24. $\mathcal{T}_g$, $\mathcal{B}_g$, and $\mathcal{C}_g$ for Image 10

Sobel baseline, and generated Pb-lite images are compared and shown next to each other in Fig 25-Fig 34.

$$PbLite = \frac{\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g}{3} \circ (w_1 * Canny + w_2 * Sobel) \quad (2)$$

### F. Performance Analysis

As this method assigns probabilities to the presence of an edge output from Canny and Sobel baselines, the overall set of edges detected by pb-lite is much less prominent than either of them. This is because this method neglects the minor effect of changes in texture, brightness, and color that Canny and Sobel baselines do not account for. This leads to a reduction in noisy edges. Though not a complete elimination of noise, it is a significant step in improvement. Further optimizations to this method can be done by varying the filter sizes, scales, orientations, Gabor frequencies, and half-disk radii used in generating the filter bank. However, using more filters comes at the cost of increased computation. Overall, the pb-lite method can be generalized to be a fairly efficient and effective method for edge detection in RGB (color) images.

## II. Phase 2: Deep Dive on Deep Learning

### A. Introduction

Neural networks can present themselves in various architectures, each having a specialty of its own. Even minor changes in network architecture or other hyperparameters can have a significant effect on model performance. This section of the assignment focuses on developing, running, and comparing the performance of different convolutional neural network (CNN) models on classifying images from the CIFAR-10 dataset. The different model architectures discussed in this assignment include a basic CNN, an optimized CNN with more layers and batch normalization, a ResNet-based model, a ResNext-based model, and a DenseNet-based model. Each of these architectures is discussed and compared in the following sections.

### B. Model 1: A Basic CNN

The first network model implemented includes a basic CNN architecture with a set of convolutional and max-pooling layers followed by two fully connected layers after flattening the intermediate convolutional layers as can be seen in Fig 37. When trained over 20 epochs and tested, this model observes very high training accuracy with a comparatively lower testing accuracy indicating the presence of model overfitting. The training loss, training accuracy, and testing accuracy when trained and tested over 20 epochs with an Adam optimizer and softmax-cross-entropy loss and a learning rate of 0.0015 over a minibatch size of 64 is shown in Fig 35. Further, the confusion matrix displaying the correct and incorrect predictions over the training and testing set is shown in Fig 36.

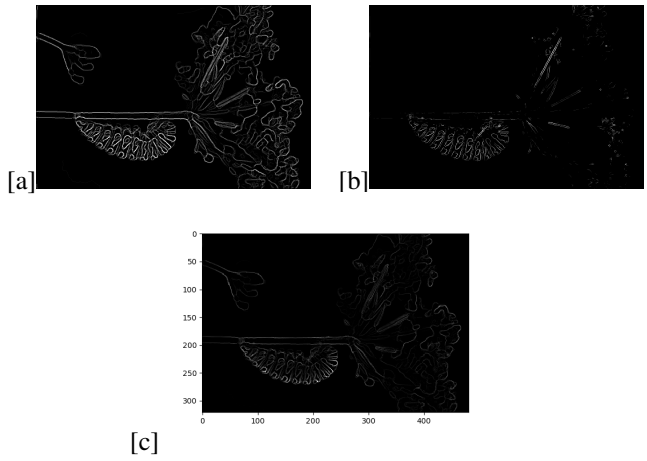



Fig. 25. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 1

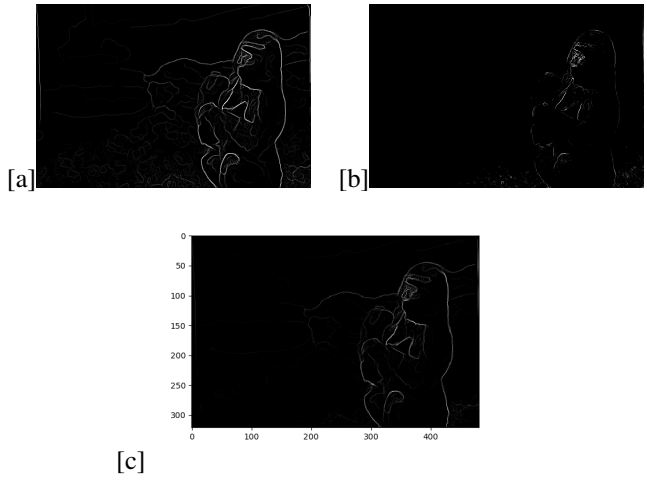



Fig. 26. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 2

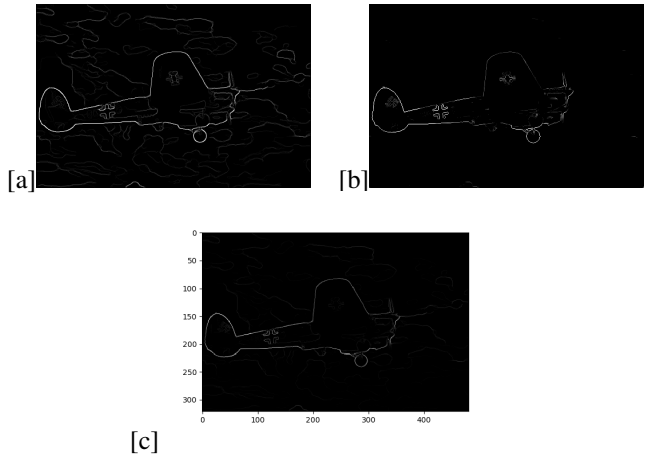



Fig. 27. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 3

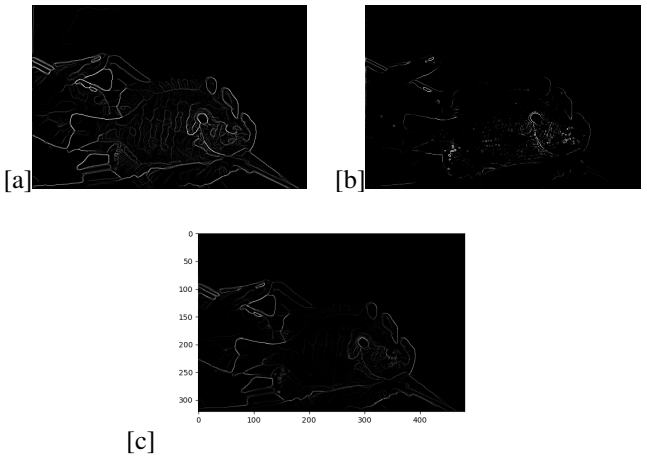



Fig. 28. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 4

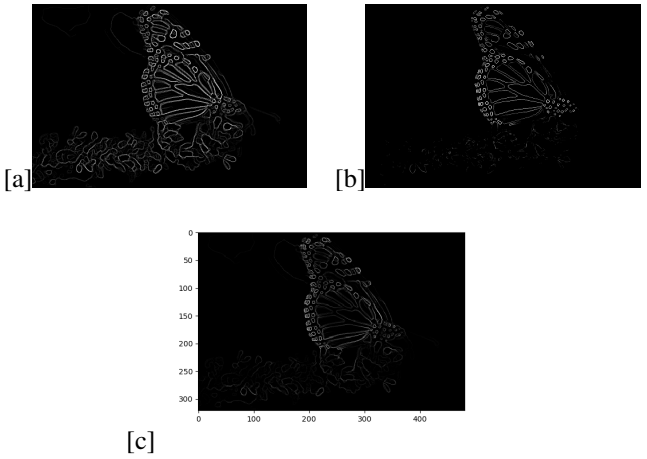



Fig. 29. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 5

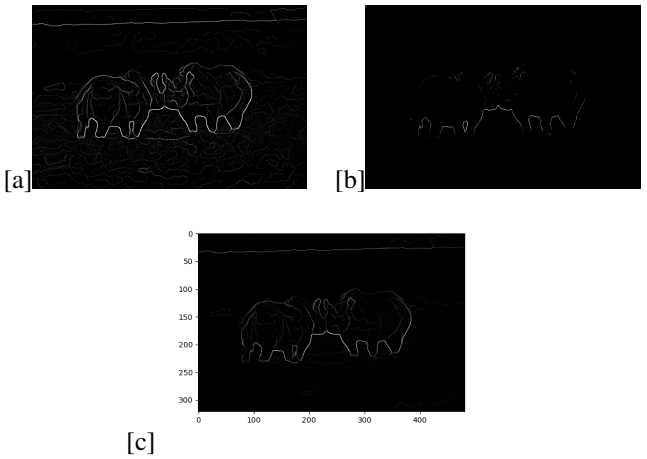



Fig. 30. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 6

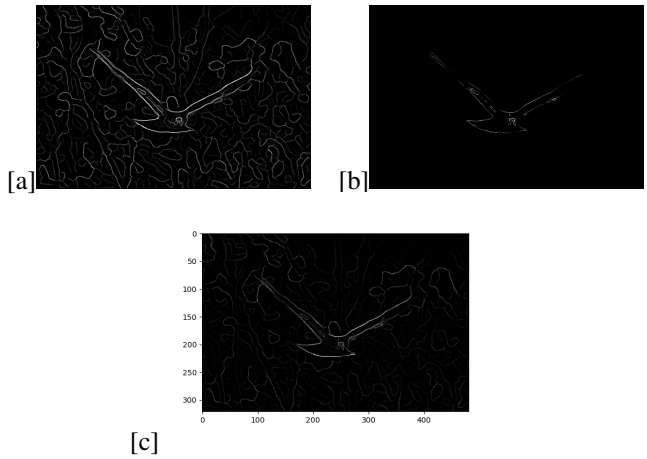



Fig. 31. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 7

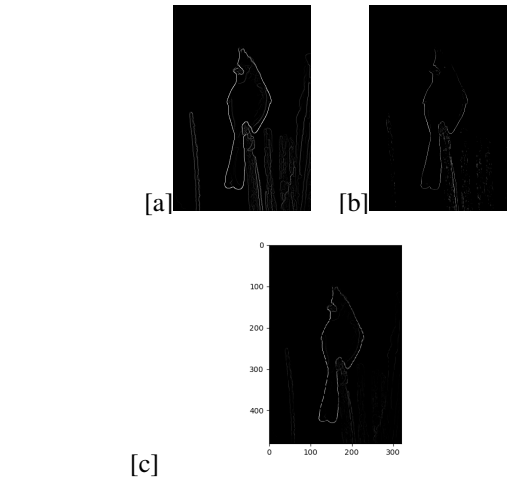



Fig. 32. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 8

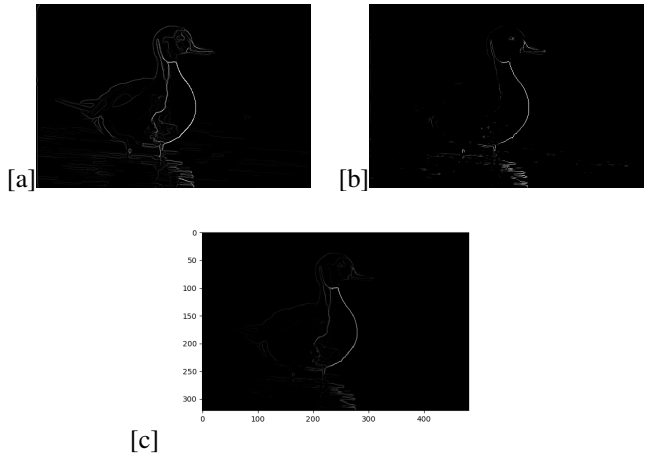



Fig. 33. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 9

### C. Model 2: An Optimized CNN

The second model implemented includes an optimized version of the basic CNN used earlier. In this architecture, batch normalization is introduced to every second convolutional layer to better center the intermediate layer outputs. the network architecture of this model is shown in Fig 40 Further, the number of layers is almost doubled and the fully connected layer neurons are increased as well. When trained under similar conditions as the basic CNN, this model outputs negligibly less training accuracy with a higher testing accuracy indicating a lower amount of overfitting and higher overall performance. The training loss, training accuracy, and testing accuracy when trained and tested over 20 epochs with an Adam optimizer and softmax-cross-entropy loss and a learning rate of 0.0015 over a minibatch size of 64 is shown in Fig 38. Further, the confusion matrix displaying the correct and incorrect predictions over the training and testing set is shown in Fig 39.

### D. Model 3: A ResNet-based CNN Architecture

This model is based on the ResNet architecture [3]. Taking inspiration from ResNet's skip connections, this network implementation includes two skip connection blocks (containing convolutional and max-pooling layers) followed by fully connected layers. The skip connections aid in gradient flow and this can be verified with the highest training accuracy among all models though a slightly lower testing accuracy. The training loss, training accuracy, and testing accuracy when trained and tested over 20 epochs with an Adam optimizer and softmax-cross-entropy loss and a learning rate of 0.0015 over a minibatch size of 64 is shown in Fig 41. Further, the confusion matrix displaying the correct and incorrect predictions over the training and testing set is shown in Fig 42. Further, the model architecture is shown in Fig 43.

### E. Model 4: A ResNext-based CNN Architecture

This model takes inspiration from the famous ResNext model architecture [4]. It is similar to the ResNet architecture except for the fact that the skip connections are slightly modified to have two branches of the network running in parallel. These blocks can be visualized from the network architecture shown in Fig 46. This model gives an intermediate training accuracy with the best testing accuracy indicating the least amount of overfitting, a characteristic that can be attributed to it having the lowest number of trainable parameters among all the models. The training loss, training accuracy, and testing accuracy when trained and tested over 20 epochs with an Adam optimizer and softmax-cross-entropy loss and a learning rate of 0.0015 over a minibatch size of 64 is shown in Fig 44. Further, the confusion matrix displaying the correct and incorrect predictions over the training and testing set is shown in Fig 45.

### F. Model 5: A DenseNet-based CNN Architecture

The DenseNet-based model [5] takes inspiration from inception modules where multiple filter sizes with constant padding

are used and their outputs are concatenated as displayed in Fig 49. The implementation used in this assignment uses two inception blocks to get a high training accuracy and intermediate testing accuracy. The training loss, training accuracy, and testing accuracy when trained and tested over 20 epochs with an Adam optimizer and softmax-cross-entropy loss and a learning rate of 0.0015 over a minibatch size of 64 is shown in Fig 47. Further, the confusion matrix displaying the correct and incorrect predictions over the training and testing set is shown in Fig 48.

### G. Performance Analysis

The variation in the number of trainable parameters, total epochs trained on learning rate ($\alpha$), and the final training and testing accuracies are shown in Table I. All models train at a similar rate, at about 35 iterations/second on an Intel I9 CPU system with 32 GB of RAM. With most hyperparameters being the same for all the models, the biggest factors causing differences in performance (training and testing accuracies) are the model size (number of trainable parameters) and architecture.

On preliminary analysis, it is easy to see that the first two models (the basic and optimized CNN) perform better than the other ones. However, it is important to consider how architectures based on ResNet, ResNext, and DenseNet are built for dealing with a lot more layers, larger training and testing sets, as well as other optimizations like dropout and L2 regularization. Even without these optimizations, the ResNext-based architecture performs the best on the testing set owing to it being a lighter model with fewer parameters. Further, using more epochs for heavier and more sophisticated models could aid model training to decrease bias and help boost model performance overall. This would especially help the DenseNet-based model which generally is known to work better than the other architectures compared with in this assignment. However, implementing a lot of these ideas and optimizations is limited to having access to appropriate hardware to train and test these models.

## III. Conclusion

In the first phase of this assignment, probability-based edge detection has successfully been implemented for a set of ten images. In the second phase, five convolutional neural network models have been implemented and after reaching reasonable performance levels, have been successfully compared with appropriate reasoning and inferences for the observed results.

## References


[1] Cmsc733 hw0 webpage. https://cmsc733.github.io/2022/hw/hw0/.

[2] Pblite assignment brownu. https://cs.brown.edu/courses/cs143/2011/proj2/.

[3] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[4] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2016.

[5] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2016.

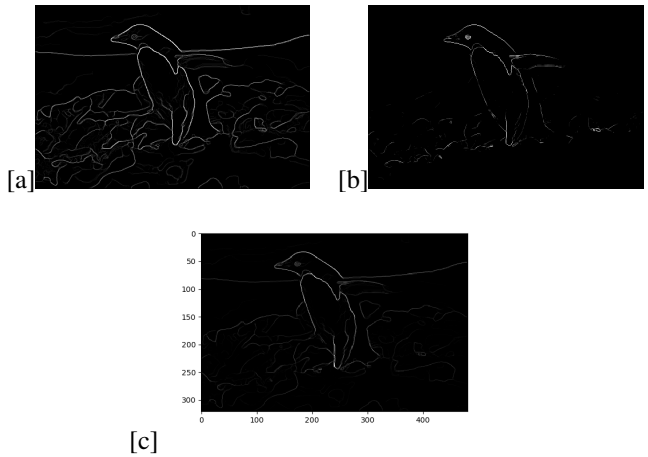



Fig. 34. Canny Baseline, Sobel Baseline, and Pb-lite Output for Image 10

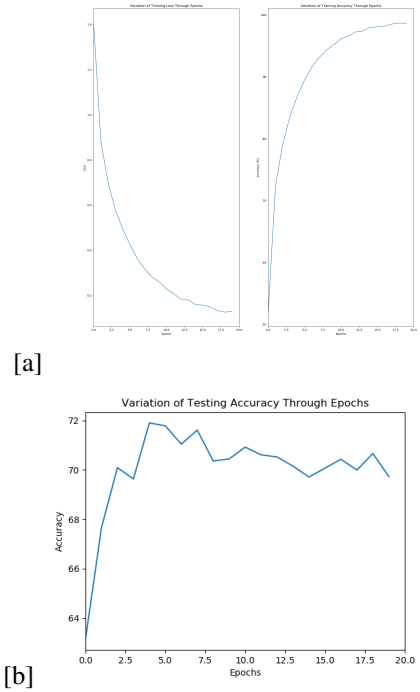
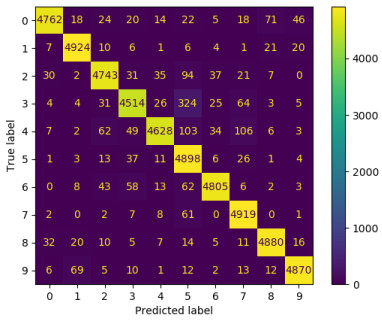


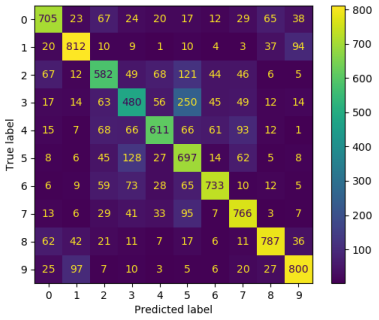

Fig. 35. Training Loss, Training Accuracy, and Testing Accuracy over Epochs for Model 1

TABLE I
Performance, Training, and Efficiency Analysis of the Different CNN Models

| Model | Parameters | Epochs | $\alpha$ | Train Acc. | Test Acc. |
|---|---|---|---|---|---|
| Basic CNN | 328986 | 20 | 0.0015 | 98.64% | 69.73% |
| Optimized CNN | 407130 | 20 | 0.0015 | 97.69% | 71.89% |
| ResNet-based | 593242 | 20 | 0.0015 | 98.85% | 65.86% |
| ResNext-based | 120282 | 20 | 0.0015 | 94.19% | 72.63% |
| DenseNet-based | 170698 | 20 | 0.0015 | 96.85% | 67.86% |




[a]




[b]

Fig. 36. Training and Testing Confusion Matrices for Model 1

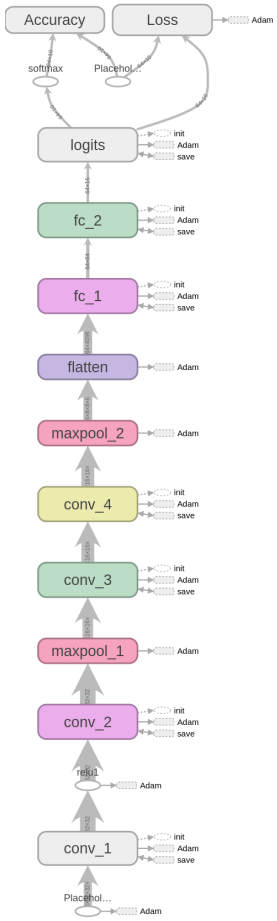
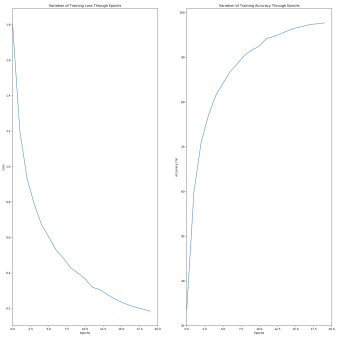
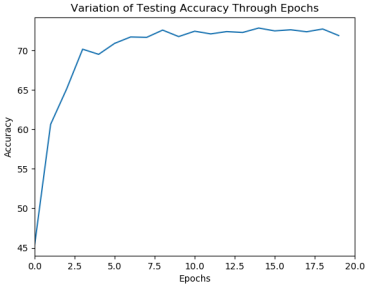



Fig. 37. CNN Architecture of Model 1

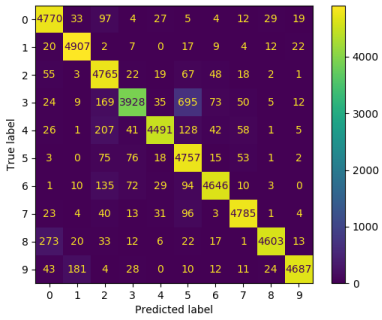


[a]

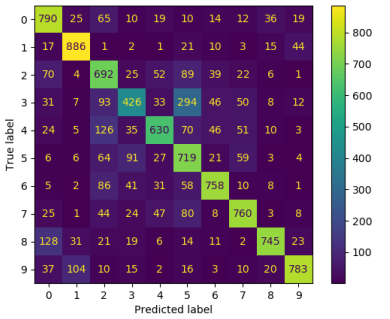



[b]

Fig. 38. Training Loss, Training Accuracy, and Testing Accuracy over Epochs for Model 2

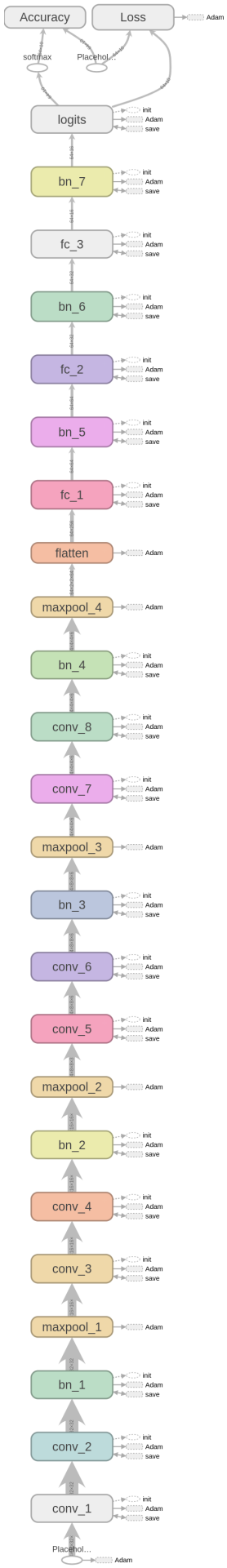


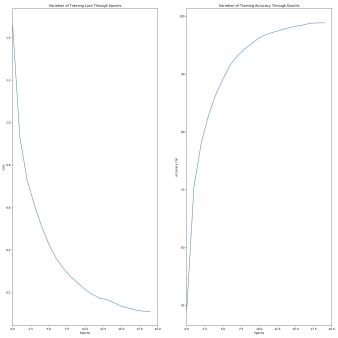

[a]



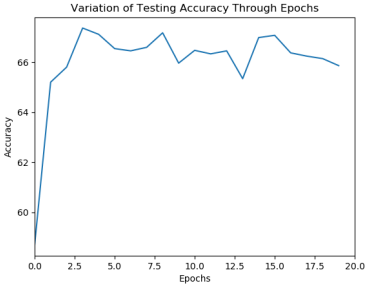

[b]

Fig. 39. Training and Testing Confusion Matrices for Model 2



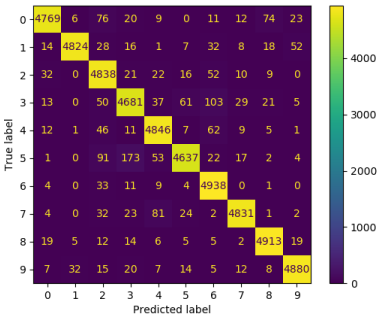


[a]

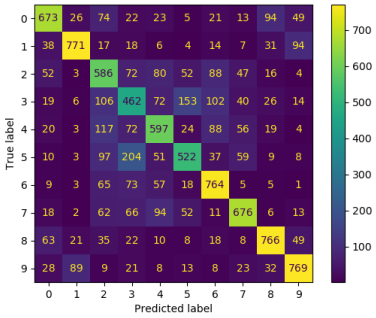



[b]

Fig. 41. Training Loss, Training Accuracy, and Testing Accuracy over Epochs for Model 3




[a]




[b]

Fig. 42. Training and Testing Confusion Matrices for Model 3

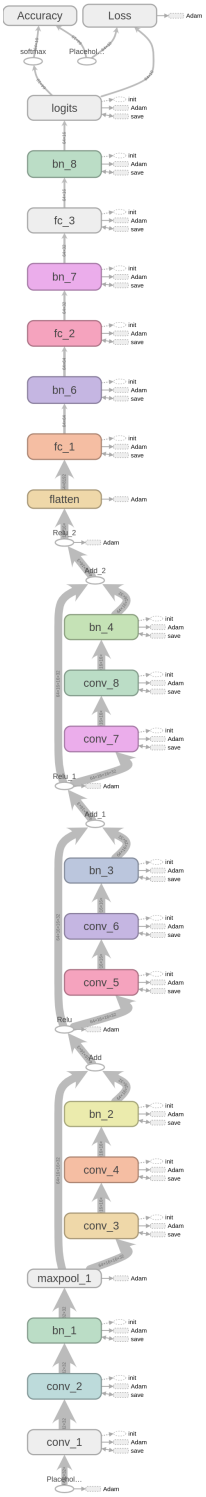
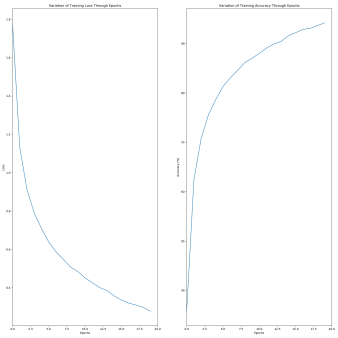
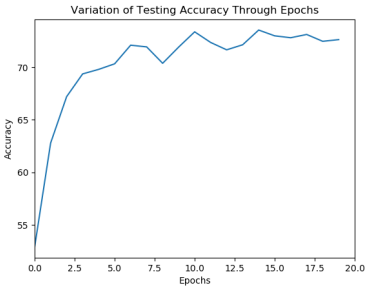



Fig. 43. CNN Architecture of Model 3

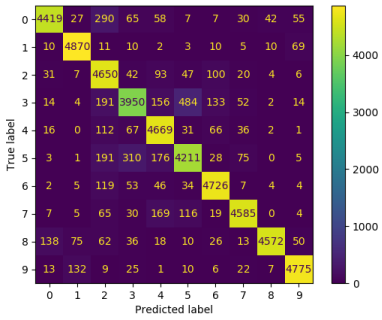


[a]

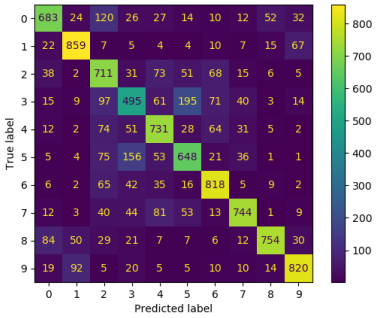



[b]

Fig. 44. Training Loss, Training Accuracy, and Testing Accuracy over Epochs for Model 4




[a]




[b]

Fig. 45. Training and Testing Confusion Matrices for Model 4

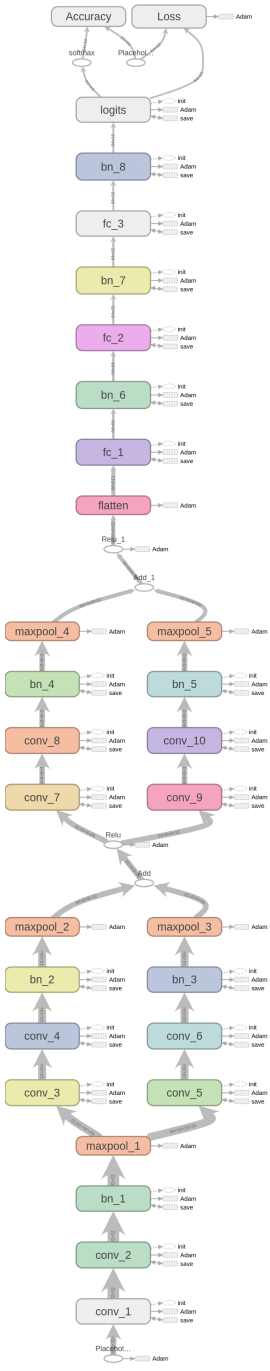
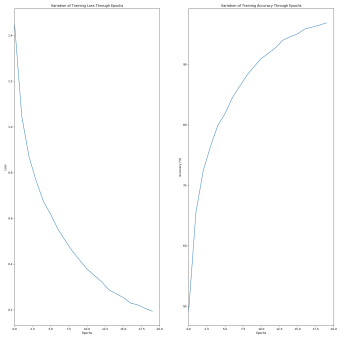
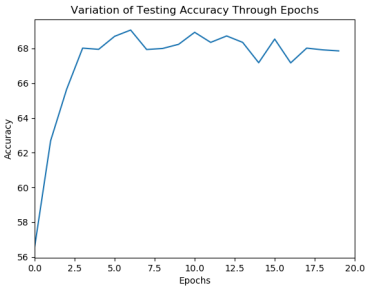



Fig. 46. CNN Architecture of Model 4

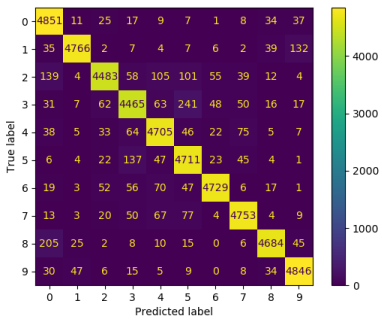


[a]

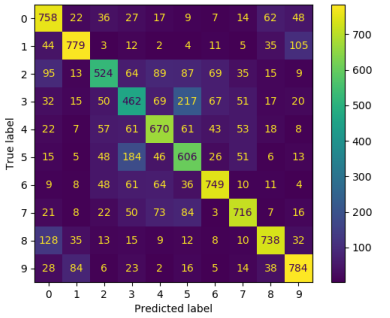



[b]

Fig. 47. Training Loss, Training Accuracy, and Testing Accuracy over Epochs for Model 5




[a]




[b]

Fig. 48. Training and Testing Confusion Matrices for Model 5

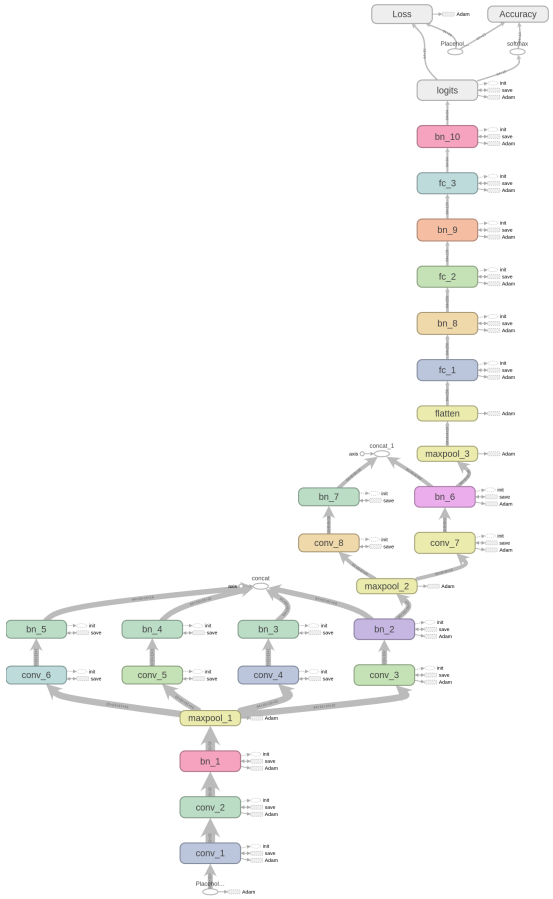



Fig. 49. CNN Architecture of Model 5