CMSC733: Homework 0 - Alohomora

Khoi Viet Pham
Email: khoi@terpmail.umd.edu
Use 6 late days

I. Phase 1: Shake My Boundary

In this section, I shall present a detailed analysis of my implementation of the pb-lite boundary detection algorithm. The idea of the algorithm is to use texture information in order to improve the boundary detection result of the Sobel and Canny baseline. The method consists of 4 main steps: 1) filter bank generation 2) texton, brightness, color map computation 3) texture, brightness, color gradient map computation 4) boundary detection.

A. Filter bank generation

The first step in the pb-lite detection pipeline is to generate a filter bank so that we can use these filters to capture texture information in the input image. There are 3 types of filters that are used in this algorithm: oriented derivative of Gaussian (DoG) filter, Leung-Malik filter, and Gabor filter. Illustration of these filters are presented in figure 1, 2, 3.

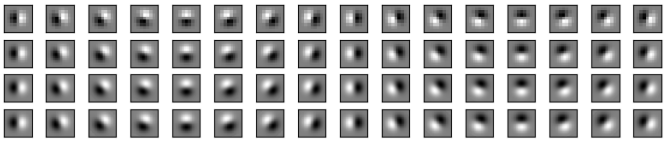

Fig. 1. Oriented derivative of Gaussian filter, generated with $\sigma = 1, 3, 5, 7$.

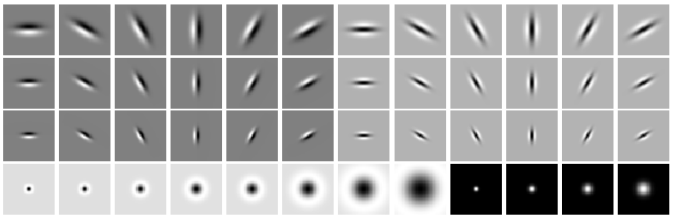

Fig. 2. Large Leung-Malik filter.

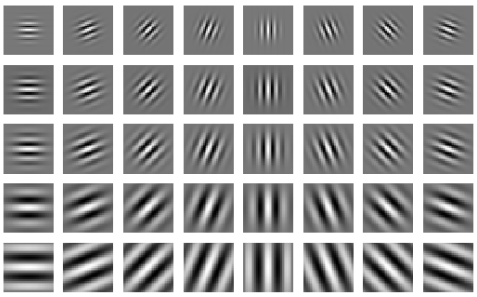

Fig. 3. Gabor filter.

B. Texton, brightness, color map computation

The next step is to filter the input image with each element in our generated filter bank. Suppose we have N filters, this filtering step gives us N filter responses at each pixel (each pixel is represented by an N-dimensional vector). We then can use KMeans clustering on the vectors of all pixels in the input image, which shall give us a discrete texton id for each pixel. Replacing each pixel in the image with its corresponding texton id gives us the texton map.

Similar to the texton map computation above, we can do the same to generate the brightness and color map. For brightness, we can use the grayscale pixel value and for color, we can use the RGB value of each pixel.

Texton, brightness, and color map of all 10 provided images are illustrated from figure 4 to figure 13.

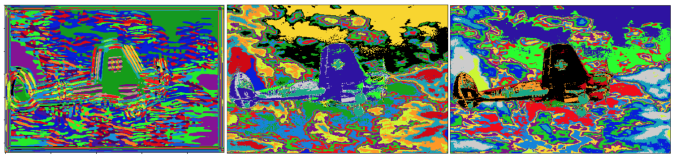

Fig. 4. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 1.

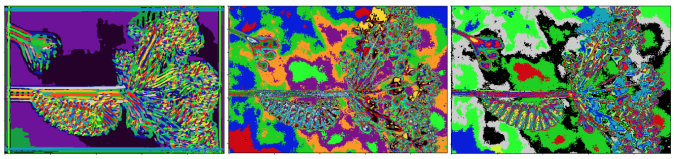

Fig. 5. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 2.

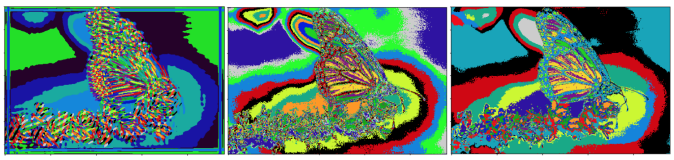

Fig. 6. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 3.

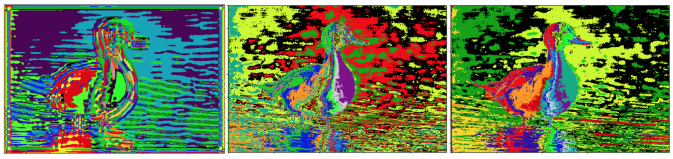

Fig. 7. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 4.

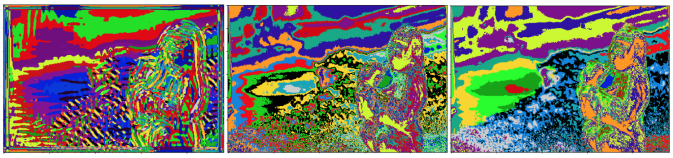

Fig. 8. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 5.

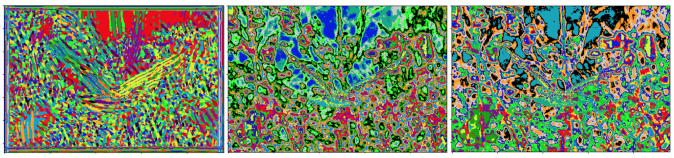

Fig. 9. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 6.

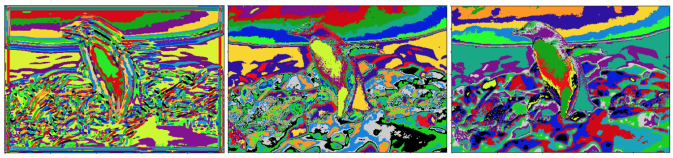

Fig. 10. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 7.

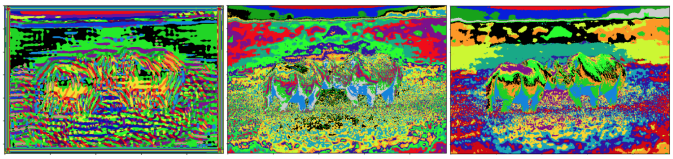

Fig. 11. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 8.

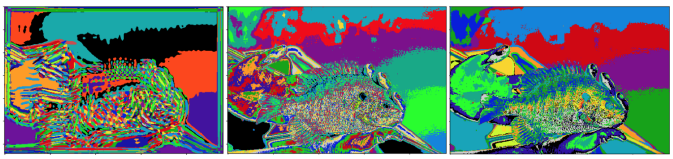

Fig. 12. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 9.

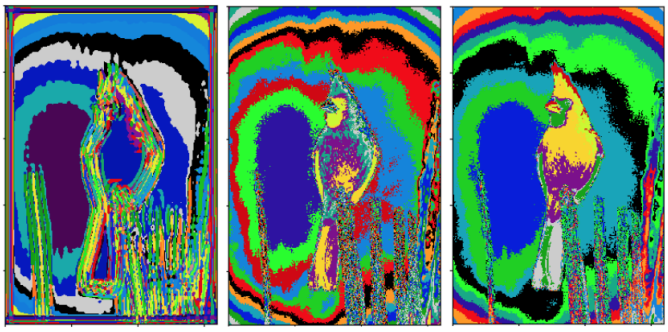

Fig. 13. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ of image 10.

C. Texture, brightness, color gradient map computation

In order to compute the texture, brightness, and color gradient map, we have to use the idea of half-disc masks. These masks are displayed in figure 14.

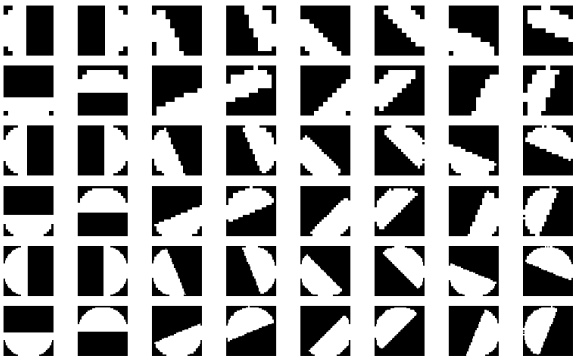

Fig. 14. Half-disc masks generated with radius 1, 3, 5, 7.

Using these half-disc masks along with the Chi-square distance, we can compute the gradient map for all texture, brightness, and color information of the input image. These gradient maps $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of all provided images are displayed from figure 15 to figure 3.

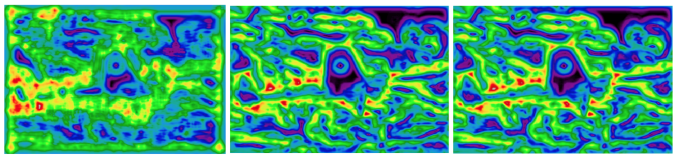

Fig. 15. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 1.

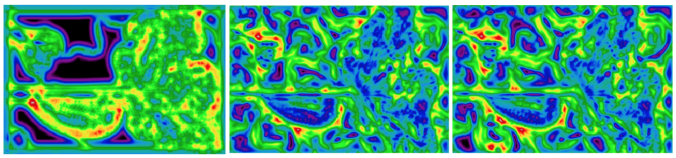

Fig. 16. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 2.

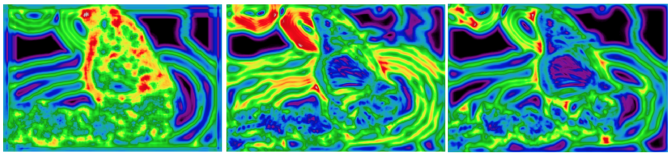

Fig. 17. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 3.

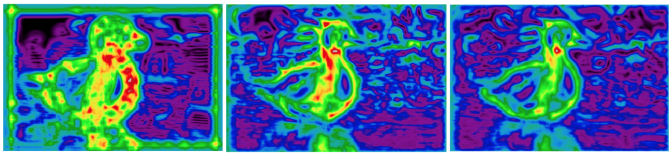

Fig. 18. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 4.

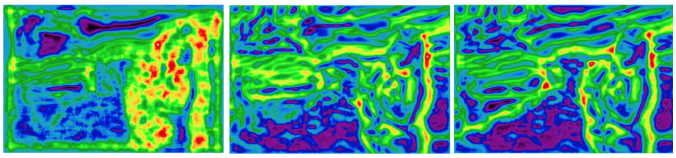

Fig. 19. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 5.

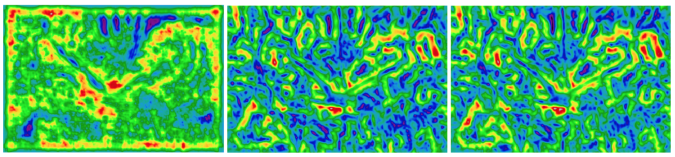

Fig. 20. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 6.

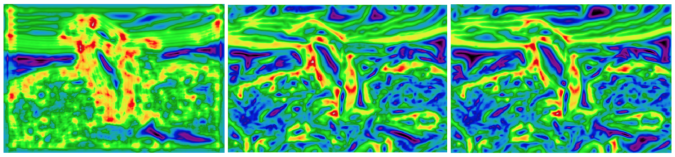

Fig. 21. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 7.

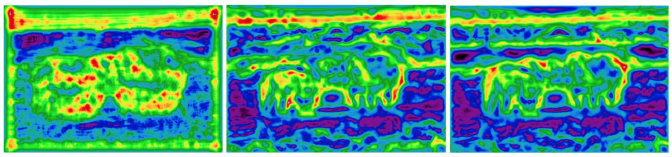

Fig. 22. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 8.

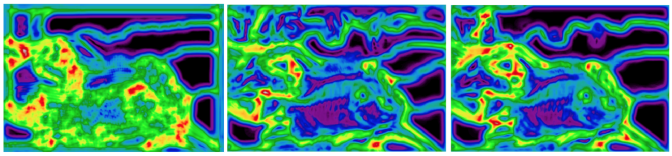

Fig. 23. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 9.

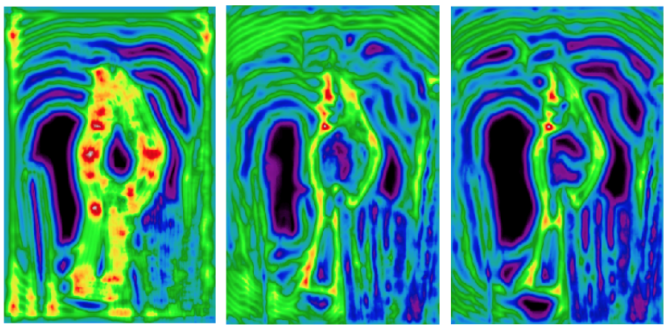

Fig. 24. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ of image 10.

D. Boundary detection

The final step is to combine the Sobel and Canny baseline with our generated texture, brightness, and color gradient map to produce the final boundary detection result. The results of all 10 provided images are illustrated from figure 25 to 34.

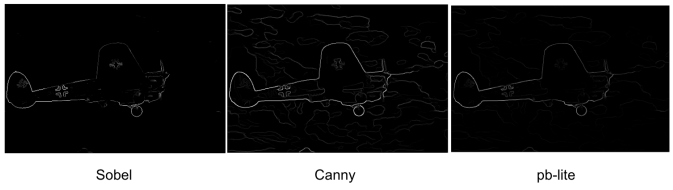

Fig. 25. Sobel, Canny, and pb-lite result of image 1.

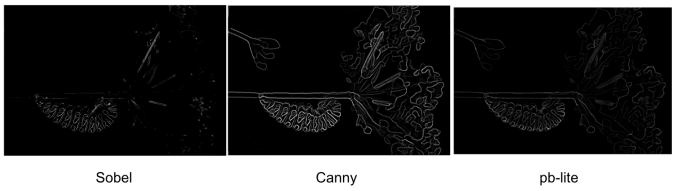

Fig. 26. Sobel, Canny, and pb-lite result of image 2.

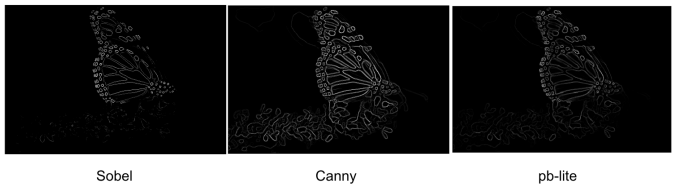

Fig. 27. Sobel, Canny, and pb-lite result of image 3.

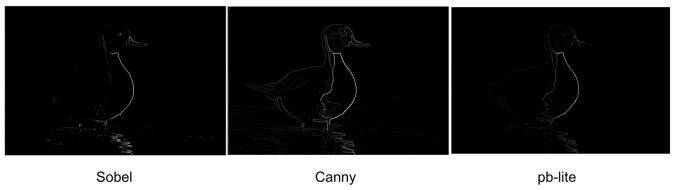

Fig. 28. Sobel, Canny, and pb-lite result of image 4.

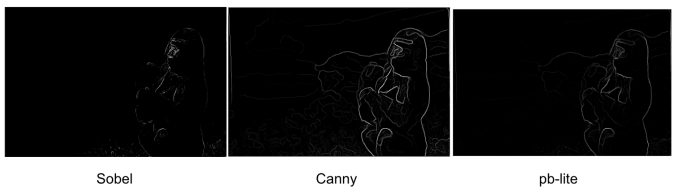

Fig. 29. Sobel, Canny, and pb-lite result of image 5.

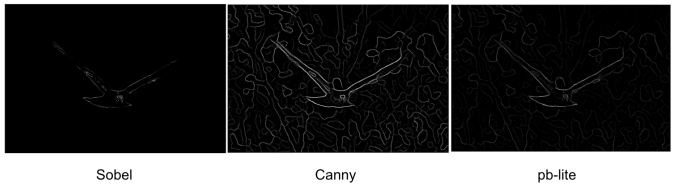

Fig. 30. Sobel, Canny, and pb-lite result of image 6.

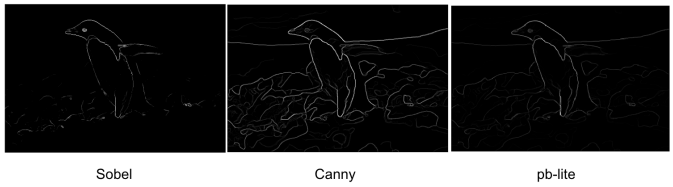

Fig. 31. Sobel, Canny, and pb-lite result of image 7.

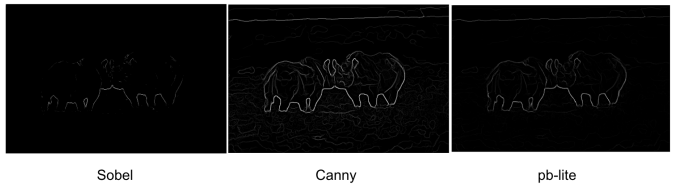

Fig. 32. Sobel, Canny, and pb-lite result of image 8.

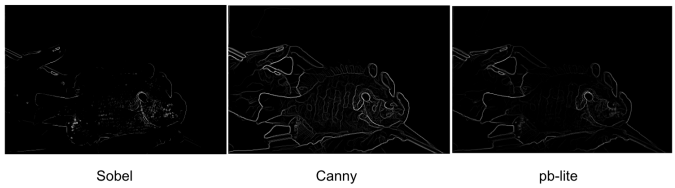

Fig. 33. Sobel, Canny, and pb-lite result of image 9.

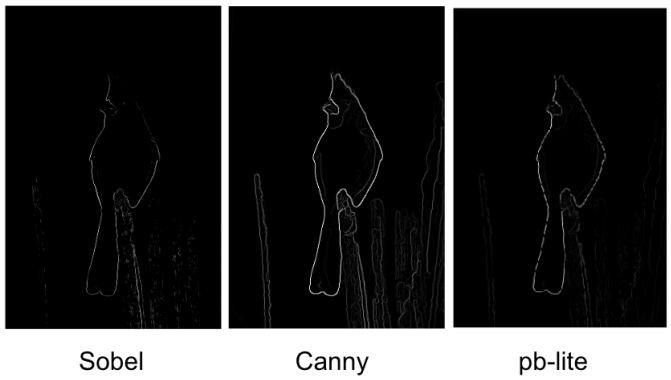

Fig. 34. Sobel, Canny, and pb-lite result of image 10.

E. Analysis

In my opinion, the output from my pb-lite implementation actually does not look as good as the Canny baseline. For image with a lot of texture, Sobel and Canny baseline tend to produce bad results because they generate too many "false positives". Pb-lite therefore comes to the rescue to suppress edges detected at those regions with lots of texture. This is why it has been shown that Pb-lite is better than Sobel and Canny. However, my implementation might have suppressed too much information and made the output not as clear as the Canny baseline. I have tried to increase the size of the filter bank by using more scale values for the Gaussian filter, however, the result only improves slightly.

II. Phase 2: Deep Dive on Deep Learning

In this section, I shall present a detailed explanation of how I approach to train neural network models on the CIFAR-10 dataset.

A. My first neural network

First, as guided by the instruction, I quickly implement and train a simple neural network. The architecture of this network is presented in figure 35 (Left). Noted that after each convolutional layer is a ReLU layer (as I want to simplify the figure, I omit adding ReLU into the figure). The number of parameters in the network is 430102. I use Adam optimizer with learning rate $1e-3$, batch size 64 to train the network. Here, I don't use any dropout or batch normalization layer since I just want to quickly see the initial performance with this simple architecture.

The train/test accuracy over epochs is plotted in figure 36, and the train/test loss over epochs is plotted in figure 37.

From the plots, we can see that my model overfits to the training set. The possible next step to improve it is to use data augmentation, dropout, batch normalization layer.

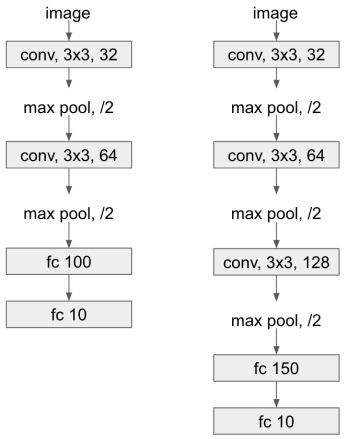

Fig. 35. (Left) My first architecture (Right) My modified architecture.

	0	1	2	3	4	5	6	7	8	9
0	4750	21	54	10	5	16	12	10	114	8
1	39	4886	5	6	2	7	6	5	15	29
2	142	6	4524	63	43	84	64	29	35	10
3	61	7	163	4170	43	348	102	59	41	6
4	111	6	408	157	3856	104	98	235	17	8
5	46	3	168	244	32	4402	36	61	6	2
6	21	10	134	117	18	34	4621	16	26	3
7	27	3	46	55	28	45	4	4786	3	3
8	64	34	13	10	3	4	6	2	4858	6
9	83	195	8	26	3	21	7	15	49	4593

TABLE I
CONFUSION MATRIX OF MY INITIAL MODEL ON THE TRAINING SET (ACCURACY: 90.892%)

	0	1	2	3	4	5	6	7	8	9
0	758	30	42	21	8	6	8	12	82	33
1	28	844	9	10	2	10	3	4	31	59
2	87	11	621	59	54	56	51	41	10	10
3	30	17	80	517	43	183	61	40	16	13
4	45	4	142	77	518	39	69	91	13	2
5	24	4	56	161	24	641	21	53	11	5
6	11	11	77	68	20	31	759	8	12	3
7	29	4	39	32	40	62	3	779	3	9
8	57	39	13	19	8	8	4	5	836	11
9	49	116	13	24	5	11	5	16	35	726

TABLE II
CONFUSION MATRIX OF MY INITIAL MODEL ON THE TESTING SET (ACCURACY: 69.99%)

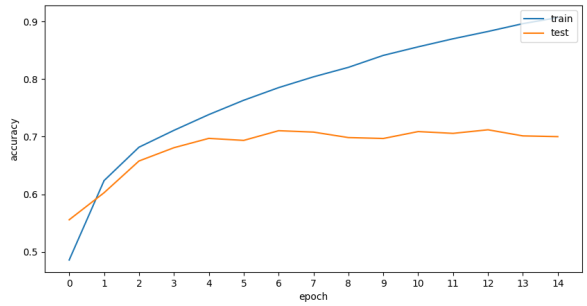

Fig. 36. Accuracy over epochs of my first neural network on the training/testing set.

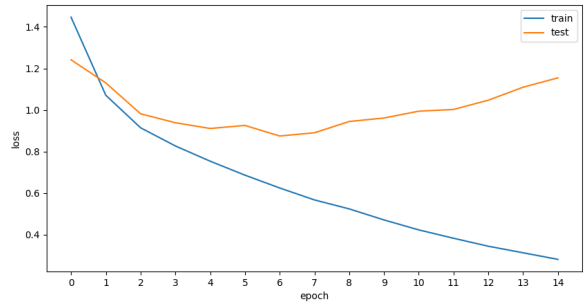

Fig. 37. Loss value over epochs of my first neural network on the training/testing set.

The confusion matrix on the training/testing set are shown in table I and II. Overall, my initial model achieves 69.99% accuracy on the testing set.

B. My improved neural network

Next, I modify the architecture a little bit by adding one extra convolutional layer at the end, as I suspect that only 2 convolutional layers might not be enough to extract really useful high-level features. As I make the model more complex, and since it was overfitting (shown in the previous subsection), it is intuitive to add dropout and batch normalization layers in order to mitigate the overfitting problem as well as to help train more stable. I add dropout after each pooling layer and batch normalization after each ReLU layer.

The number of parameters in my model is 402856. I also perform data augmentation by random flipping the image horizontally. I also notice that scaling the data into the range $[0, 1]$ helps improve slightly. Here, I also use Adam optimizer with learning rate $1e-3$ and batch size 64 to train the model.

The train/test accuracy over epochs is plotted in figure 38, and the train/test loss over epochs is plotted in figure 39.

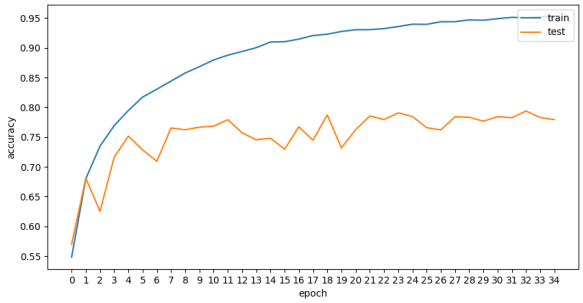

Fig. 38. Accuracy over epochs of my modified neural network on the training/testing set.

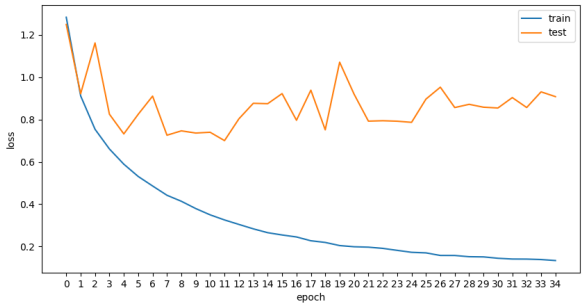

Fig. 39. Loss value over epochs of my modified neural network on the training/testing set.

The confusion matrix on the training/testing set are shown in table III and IV. Overall, my improved model achieves 79.38% accuracy on the testing set.

C. DenseNet

I implement the DenseNet architecture based on the paper. The architecture I use is not exactly the same as presented in the paper. I present my architecture in figure 40. The yellow block is the densely connected block as described in the paper. I use 4 dense blocks, each of which has 6, 12, 24, 16 processing units. Each unit in a dense block is a composite

	0	1	2	3	4	5	6	7	8	9
0	4983	1	2	0	3	0	3	1	5	2
1	6	4967	0	1	1	0	0	0	3	22
2	86	0	4868	2	17	1	17	7	2	0
3	20	0	32	4791	52	15	66	7	13	4
4	9	0	20	3	4958	1	4	2	1	2
5	6	1	27	59	37	4797	41	19	8	5
6	4	0	8	0	4	0	4982	1	1	0
7	5	0	5	3	13	3	0	4968	1	2
8	23	1	0	0	2	0	0	0	4973	1
9	4	14	1	0	1	0	0	1	3	4976

TABLE III
CONFUSION MATRIX OF MY MODIFIED MODEL ON THE TRAINING SET (ACCURACY: 98.526%)

	0	1	2	3	4	5	6	7	8	9
0	879	9	16	7	9	3	12	7	35	23
1	9	875	6	5	1	4	4	1	29	66
2	78	2	696	30	70	31	57	21	9	6
3	24	8	58	576	80	100	82	36	21	15
4	16	1	48	35	803	13	40	37	6	1
5	13	4	50	154	55	609	46	47	9	13
6	4	6	35	27	24	7	882	4	7	4
7	12	2	25	21	54	38	7	822	5	14
8	39	14	9	2	7	1	4	1	910	13
9	23	37	5	6	4	4	6	3	26	886

TABLE IV
CONFUSION MATRIX OF MY MODIFIED MODEL ON THE TESTING SET (ACCURACY: 79.38%)

unit that consists of a batch normalization layer, a ReLU layer, and a 3×3 convolutional layer. I also add a dropout layer after each convolutional layer. The number of parameters in my model is 1158282. I use Adam optimizer with learning rate 1e-3, batch size 64 to train the network.

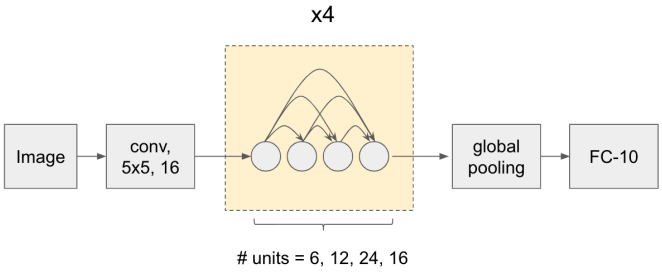

Fig. 40. Architecture of my implemented DenseNet.

The train/test accuracy and loss over epochs are plotted in figure 41 and 42.

	0	1	2	3	4	5	6	7	8	9
0	4890	2	26	4	11	4	2	8	14	39
1	10	4921	0	1	0	1	1	1	3	62
2	52	2	4822	13	45	15	14	34	1	2
3	23	4	32	4417	81	346	20	47	3	27
4	7	0	21	5	4871	21	3	69	0	3
5	8	0	24	79	38	4744	3	103	1	0
6	14	7	64	22	26	4	4848	5	1	9
7	0	0	10	6	4	5	1	4973	0	1
8	71	25	7	1	1	2	2	2	4821	68
9	4	13	2	0	2	0	0	0	2	4977

TABLE V
CONFUSION MATRIX OF MY IMPLEMENTED DENSENET ON THE TRAINING SET (ACCURACY: 96.568%)

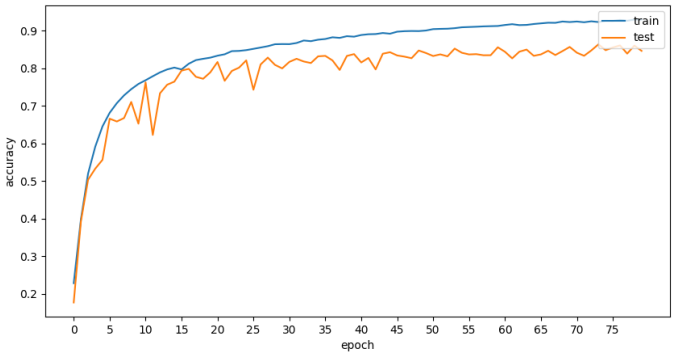

Fig. 41. Accuracy over epochs of my implemented DenseNet on the training/testing set.

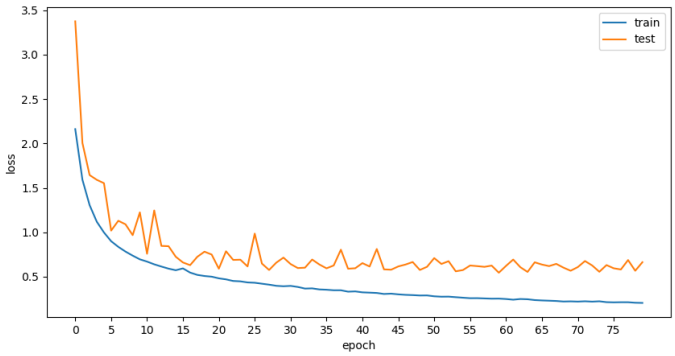

Fig. 42. Loss over epochs of my implemented DenseNet on the training/testing set.

The confusion matrix are shown in table V and VI. Overall, my implementation of DenseNet achieves 86.34% on the testing set.

D. ResNet

I also implement the idea in ResNet, however, I do not use exactly the same architecture as presented in the paper. Even though it has been reported in the ResNet paper that ResNet achieves very good result on CIFAR-10, my implementation of ResNet cannot achieve that. The architecture is presented in figure 43. Each convolutional layer is followed by a batch normalization and a ReLU layer. The number of parameters in the model is 166218.

	0	1	2	3	4	5	6	7	8	9
0	884	5	33	6	9	0	2	10	25	26
1	5	941	0	0	1	1	0	0	7	45
2	35	0	817	11	48	32	25	22	5	5
3	17	5	36	690	35	120	37	38	2	20
4	9	2	30	23	854	14	11	52	2	3
5	4	6	38	69	25	807	3	39	1	8
6	9	3	33	24	27	8	882	7	0	7
7	9	1	11	12	19	14	0	930	1	3
8	47	20	5	5	1	1	1	3	883	34
9	6	32	2	4	1	0	0	2	7	946

TABLE VI
CONFUSION MATRIX OF MY IMPLEMENTED DENSENET ON THE TESTING SET (ACCURACY: 86.34%)

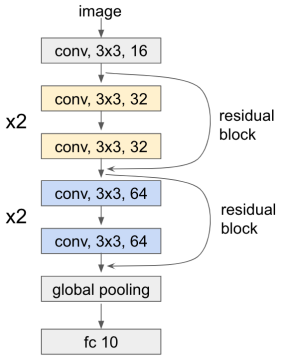

Fig. 43. Architecture of my implemented ResNet.

The train/test accuracy and loss over epochs are plotted in figure 44 and 45.

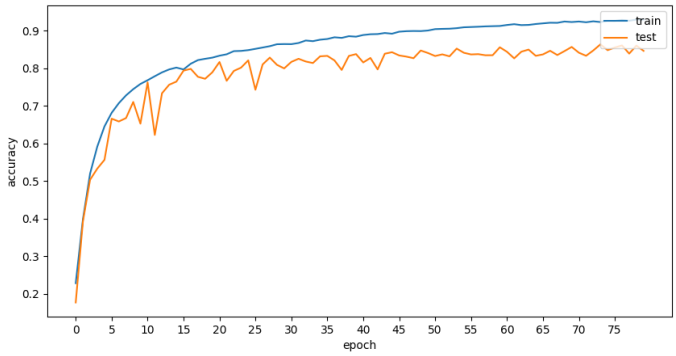

Fig. 44. Accuracy over epochs of my implemented ResNet on the training/testing set.

	0	1	2	3	4	5	6	7	8	9
0	4584	48	7	30	95	23	40	33	26	114
1	1	4950	0	0	0	0	7	0	1	4
2	88	13	3900	36	213	179	464	75	10	22
3	19	3	15	4101	88	267	403	25	10	69
4	1	0	3	5	4641	26	307	9	1	7
5	4	2	2	12	21	4864	65	19	3	8
6	3	1	1	5	2	9	4971	3	2	3
7	1	5	0	5	69	42	92	4753	2	31
8	31	74	2	16	38	15	83	6	4666	69
9	1	38	0	0	1	3	6	0	2	4949

TABLE VII
CONFUSION MATRIX OF MY IMPLEMENTED RESNET ON THE TRAINING SET (ACCURACY: 92.758%)

	0	1	2	3	4	5	6	7	8	9
0	764	31	20	14	39	9	23	13	40	47
1	0	926	1	2	2	1	9	4	4	51
2	47	7	535	36	87	78	167	27	7	9
3	12	15	28	509	49	174	157	22	3	31
4	6	2	13	13	760	38	140	20	2	6
5	7	3	6	50	35	800	52	31	2	14
6	7	3	3	11	7	10	947	1	2	9
7	4	3	4	11	67	51	37	803	3	17
8	35	41	4	10	12	8	28	3	818	41
9	6	67	1	4	3	2	7	2	7	901

TABLE VIII
CONFUSION MATRIX OF MY IMPLEMENTED RESNET ON THE TESTING SET (ACCURACY: 77.63%)

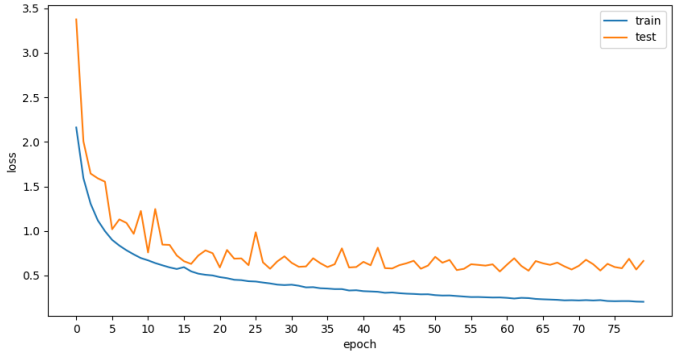

Fig. 45. Loss over epochs of my implemented ResNet on the training/testing set.

The confusion matrix are shown in table VII and VIII. Overall, my implementation of ResNet achieves 86.34% on the testing set.

E. Analysis

Table IX shows comparison between the above models in terms of train/test accuracy and number of parameters.

My simple model performs worse than my modified one even though it has more training parameters. This can be

Model	# of params	Train acc	Test acc
Simple	430102	90.892%	69.99%
Modified	402856	98.526%	79.38%
ResNet	166218	92.759%	77.63%
DenseNet	1158282	96.568%	86.34%

TABLE IX
SUMMARY OF ALL MODELS

because my simple model has not been trained to reach a training accuracy as high as my modified one. However, I found it difficult to reach $\approx 98\%$ training accuracy for my simple model due to the fact that it only has 2 convolutional layers (it is shallow). Adding one extra convolutional layer as well as batch normalization and dropout easily improves my model a lot.

My implementation of ResNet does not perform as expected. I haven't had the time to tune the hyperparameters some more but I believe I can make it work better if I had time. The problem I am having now with this ResNet implementation is that it is overfitting to the training set. I believe reduce the number of residual blocks and the number of filters in the convolutional layers will help.

My implementation of DenseNet achieves the best performance among these models.