RBE 549: Homework0 - Alohamora

Tript Sharma
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: tsharma@wpi.edu
Using 1 late day

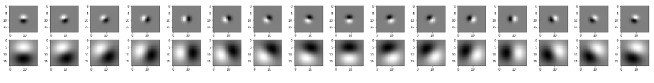

Fig. 1. DoG Filters (with σ = 1,3)

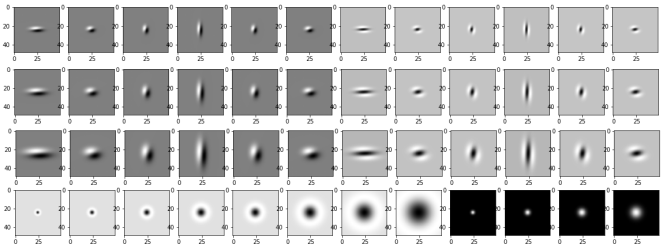

Fig. 2. LM (Large) Filters

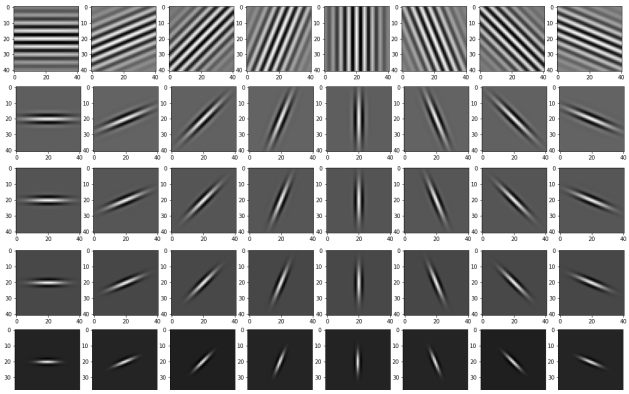

Fig. 3. Gabor Filters

I. Phase 1: Shake My Boundary

The first task of the assignment was to perform pb-lite edge-detection, a simplified version of the pb boundary detection algorithm, on a set of images and compare them with the traditional edge-detection algorithms like sobel and canny. Pb-lite algorithm uses more than just intensity discontinuities in the image. It analyses multiple properties of the images like the intensity, color and texture discontinuities returning a probabistic output classifying each pixel as part of an edge or not. The algorithm has four stages:

1) Filter bank generation
2) Texton, Brightness and Color map generation
3) Gradient map generation
4) pb-lite edge output

A. Filter bank generation

Filter bank is an array of bandpass filters that separate the the input signal into multiple components **??**. In other words, it is a set of maps that activate when they find a similar distribution of pixels in the image. Here, I used a combination of DoG filters, LM filters and Gabor filters to create the filter bank. Each consisting of 32, 96 (2 sets of 48 filters in different scales) and 40 filters respectively. Figures 1, 2 and 3 show the Derivative of Gradient (DoG), Leung-Malik and Gabor filters respectively.

B. Texton, Brightness and Color Maps Generation

Texton maps are created by passing the input image through our filter bank followed by dimensionality reduction. This results in the generation of a N dimension vector for each pixel. Hence, the out is a (R,W,N) matrix. To convert this N dim vector in a 1D vector, we perform K-Means cluster to obtain discrete texxton ids for each pixel. Similarly, we perform K-Means clustering on grayscale and RGB images of the input image to generate brightness and color maps respectively. Figures 4 to 13 show the brightness, color and texton maps respectively for each input figure. The image numbers might not correspond to the ones given in the set due to how Python traverses over filenames in a folder.

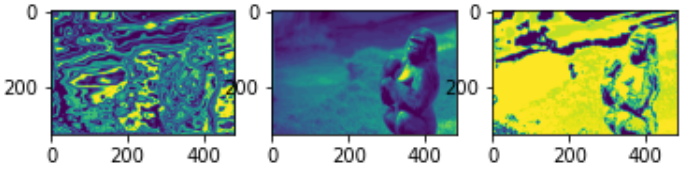

Fig. 4. Brightness, color and texton Maps for Image 1

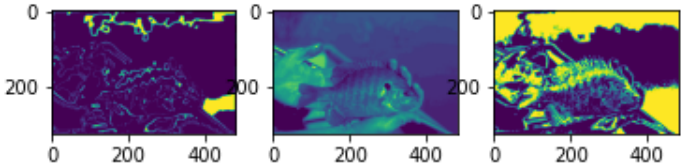

Fig. 5. Brightness, color and texton Maps for Image 2

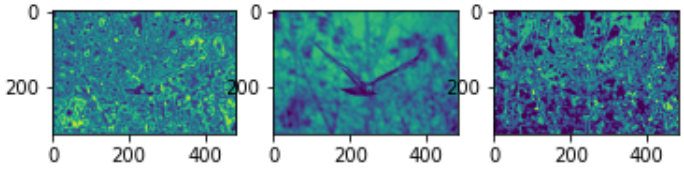

Fig. 6. Brightness, color and texton Maps for Image 3

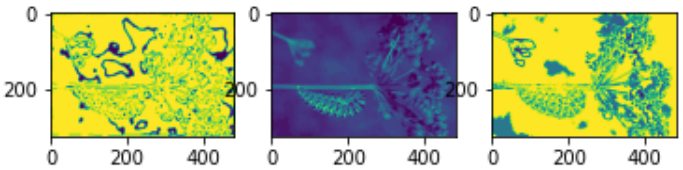

Fig. 7. Brightness, color and texton Maps for Image 4

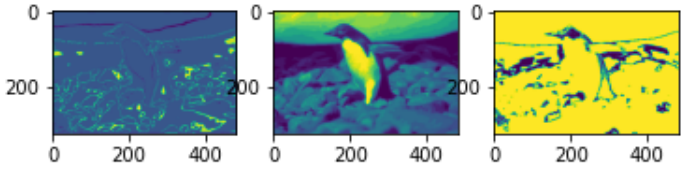

Fig. 8. Brightness, color and texton Maps for Image 5

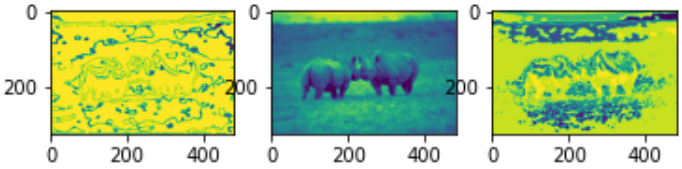

Fig. 9. Brightness, color and texton Maps for Image 6

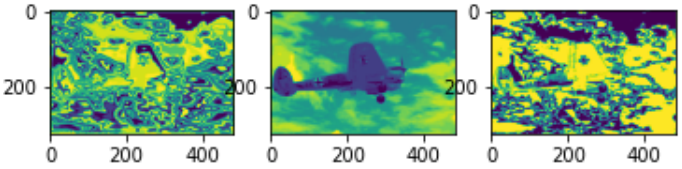

Fig. 10. Brightness, color and texton Maps for Image 7

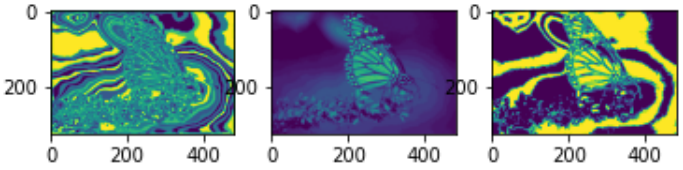

Fig. 11. Brightness, color and texton Maps for Image 8

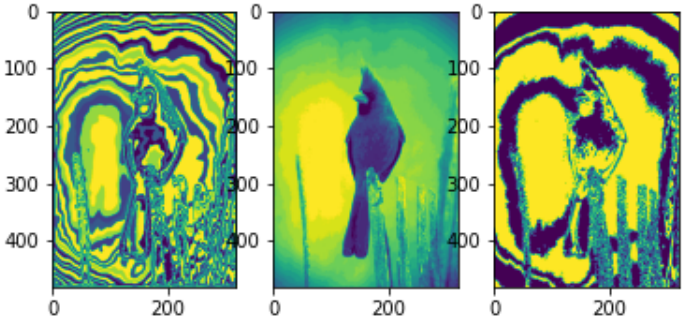

Fig. 12. Brightness, color and texton Maps for Image 9

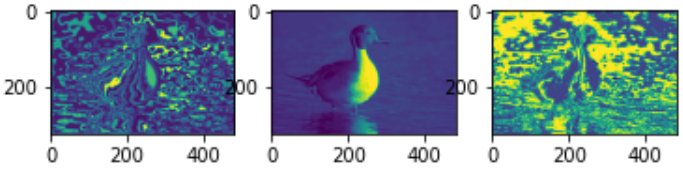

Fig. 13. Brightness, color and texton Maps for Image 10

C. Gradient map generation

The generated texton, color and brightness maps are convoluted with half disk mask pairs. Half disk masks are binary images generated by rotating a semicircle. I generated these disks by first creating a semicircle. I traversed in the 2D space of the kernel and calculated euclidean distance from the origin for each pixel. This was followed by using the OpenCV rotate function to obtain the half-disk mask bank.

The maps generated in Section I-B were convolved with the half-disk masks generated in Figure 14 and the resultants were

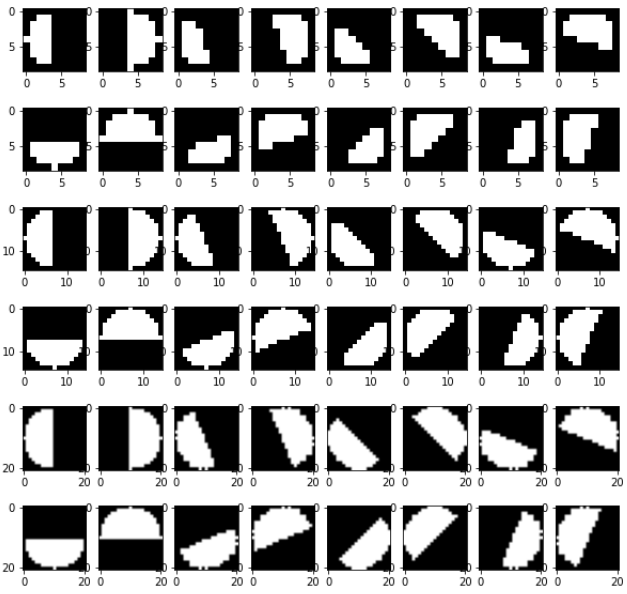

Fig. 14. Half Disk Mask pairs in 8 orientations and 3 scales

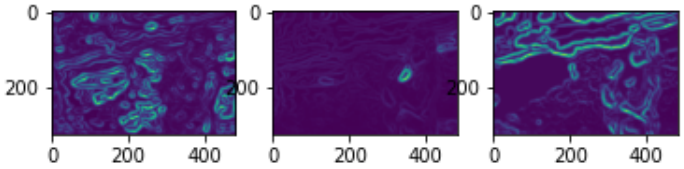

Fig. 15. T_g, B_g, C_g for Image 1

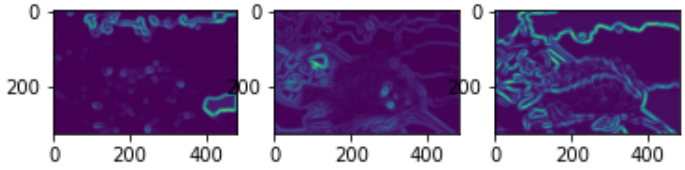

Fig. 16. T_g, B_g, C_g for Image 2

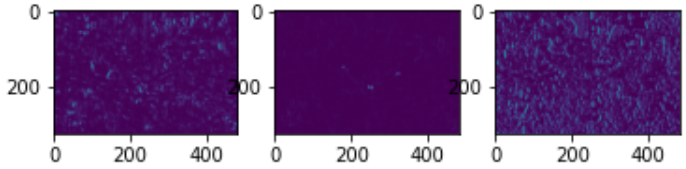

Fig. 17. T_g, B_g, C_g for Image 3

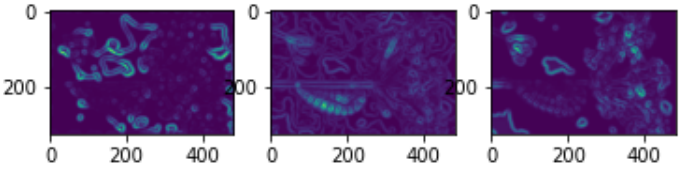

Fig. 18. T_g, B_g, C_g for Image 4

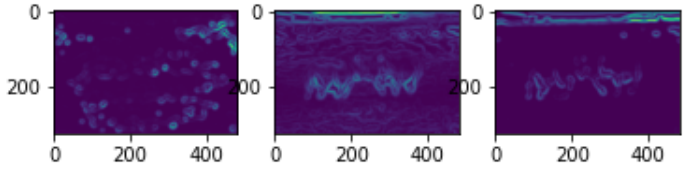

Fig. 19. T_g, B_g, C_g for Image 6

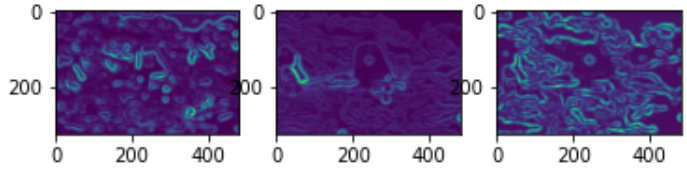

Fig. 20. T_g, B_g, C_g for Image 7

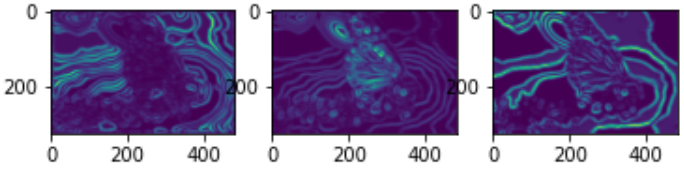

Fig. 21. T_g, B_g, C_g for Image 8

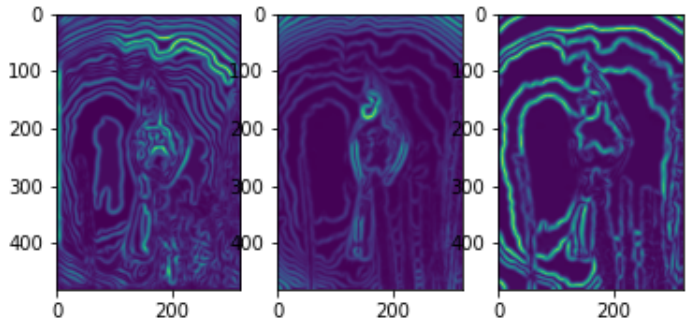

Fig. 22. T_g, B_g, C_g for Image 9

used to perform test and compute the corresponding gradients in texton, color and brightness domains which are denoted by T_g, C_g, B_g respectively. The gradient maps are shown from Figure 15 to 23

D. pb-lite Output

The gradients generated in previous subsection are used along with sobel and canny baselines to obtain a probabilistic boundary for each image. We aggregate the gradients across the *N* dimension vector for each gradient image. The obtained 2D T_g, C_g, B_g maps are aggregated and used to obtain Hadamard product with the sobel and canny results. The resultant is called pb-lite output and is presented for each image in Figures 24 to 33.

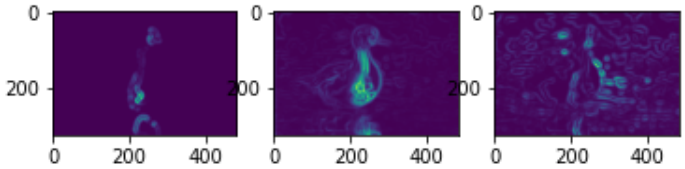

Fig. 23. T_g, B_g, C_g for Image 10

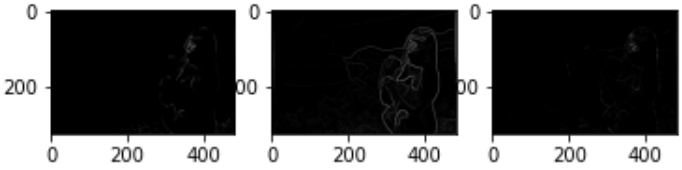

Fig. 24. Canny, Sobel, pb-lite Outputs for Image 1

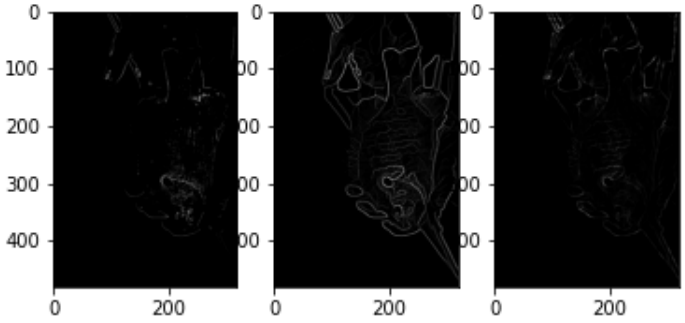

Fig. 25. Canny, Sobel, pb-lite Outputs for Image 2 (rotated 90 degrees clockwise)

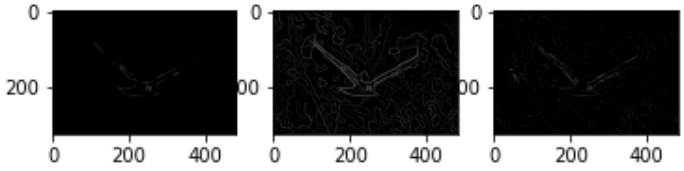

Fig. 26. Canny, Sobel, pb-lite Outputs for Image 3

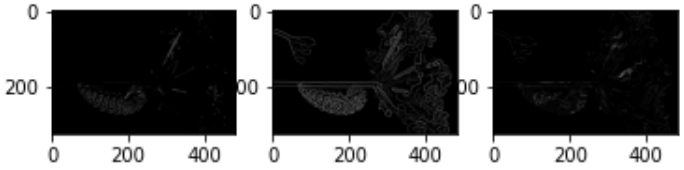

Fig. 27. Canny, Sobel, pb-lite Outputs for Image 4

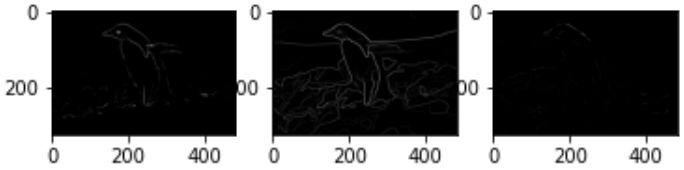

Fig. 28. Canny, Sobel, pb-lite Outputs for Image 5

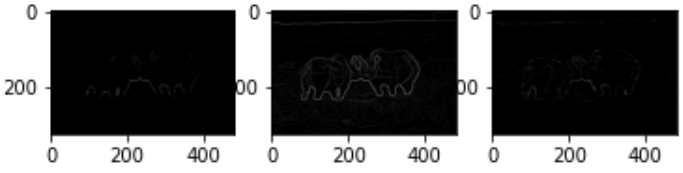

Fig. 29. Canny, Sobel, pb-lite Outputs for Image 6

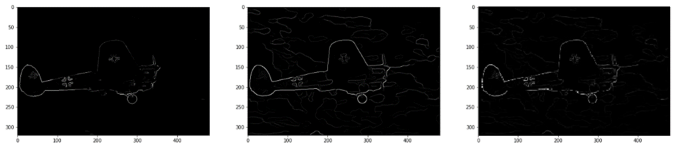

Fig. 30. Canny, Sobel, pb-lite Outputs for Image 7

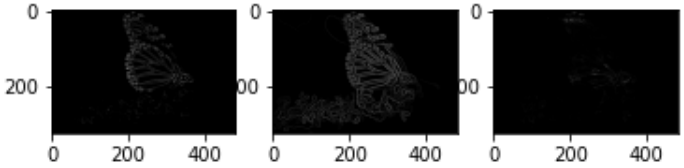

Fig. 31. Canny, Sobel, pb-lite Outputs for Image 8

E. Result

According to the results pb-lite outperforms canny edge detection on most images. Furthermore, it works better in reducing false positives when compared with sobel edge detection owing to it's computation of color and texture gradients along with the identification of intensity gradients which is the princple upon which the two traditional algorithms work. It must be noted that pb-lite does not much parameter tuning. It is more dependent upon filter maps rather than scale of the gaussian kernels. Although, the filter maps used here are generated using gaussian kernels and their first and second order derivatives, the need to tune the scale and kernel size is dispensed because of the array of filters that are generated. Hence, pb-lite is quite flexible to use as you can have a different set filter maps to obtain a new result. Hence, I feel pb-lite is better than canny and sobel.

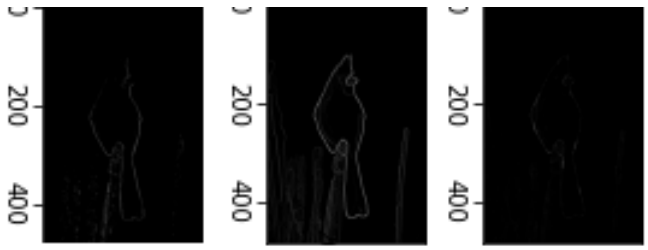

Fig. 32. Canny, Sobel, pb-lite Outputs for Image 9

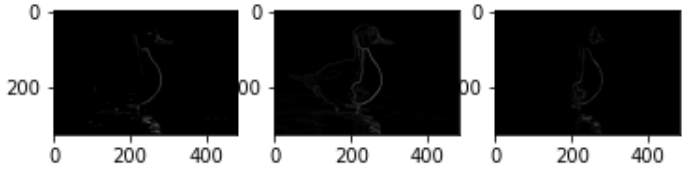

Fig. 33. Canny, Sobel, pb-lite Outputs for Image 10

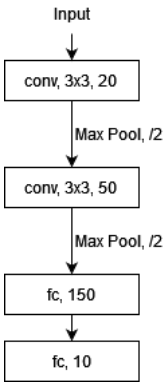

Fig. 34. Base Model Architecture

Fig. 35. Base model accuracy on train and test sets

II. Phase 2: Deep Dive on Deep Learning

In Phase 1 we implemented a traditional computer vision algorithm that required generation of filter maps to compute the edges in an image. However, contemporary approaches involve deep learning where these filter maps are created autonomously given the input and output datasets. Here we train deep learning networks to classify objects in CIFAR-10 dataset.

A. Base Model

I used a small 2 layer CNN to create a threshold of the results. The model architecture is present in Figure 34. It uses SGD loss function with an LR of 0.001 and a batch size of 64 images. The model acheived an accuracy of 25.36% and 25.56% on the train and test sets respectively.

The train accuracy and loss are shown in Figures 35 and 36 respectively.

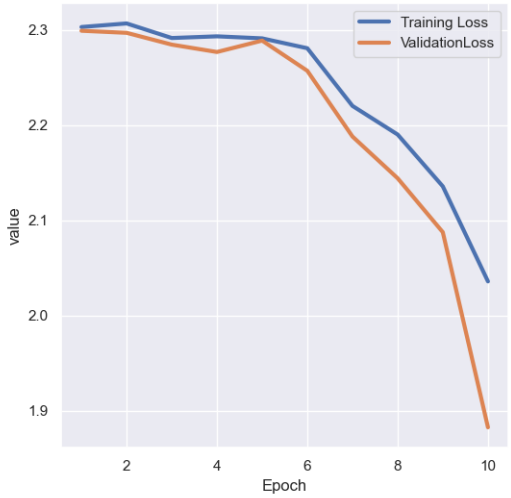

Fig. 36. Base model loss on train and test sets

TABLE I
Confusion Matrix for Base Model on Train Set

2321	120	27	41	6	315	213	74	1476	407
599	413	25	56	12	645	541	144	1735	830
1169	175	228	158	43	780	1529	218	358	342
708	116	129	371	19	1405	1252	251	356	393
536	80	143	185	54	832	2287	228	300	355
728	110	134	278	19	1668	1078	244	477	264
267	143	97	174	27	741	2802	301	134	314
489	232	122	274	33	746	1208	633	451	812
1097	83	15	27	1	586	75	27	2640	449
459	226	31	80	5	255	367	286	172	1549

TABLE II
Confusion Matrix for Base Model on Test Set

474	20	5	6	1	66	31	23	305	69
109	86	6	11	2	121	120	21	392	132
254	34	51	30	5	146	306	45	65	64
146	24	27	85	2	272	248	64	62	70
118	22	23	40	8	150	465	61	42	71
171	29	39	44	5	305	201	60	99	47
60	34	9	31	1	134	584	72	19	56
99	54	28	63	4	147	199	127	106	173
211	17	2	4	0	112	14	11	531	98
110	43	5	15	2	50	71	57	342	305

B. Base Model Update

To my initial model in Subsection II-A, I added batch normalization after each CNN layer. The model architecture is present in Figure 37. It uses SGD loss function with an LR of 0.001 and a batch size of 64 images. The inputs to the model were also normalized to a range of [-1,1]. The model achieved a maximum accuracy of 59.62% and 57.43% on the train and test sets respectively.

The train accuracy and loss are shown in Figures 38 and 39 respectively.

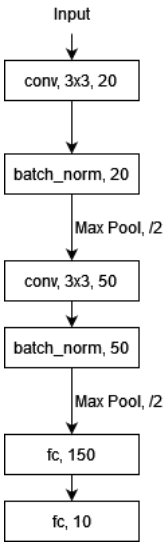

Fig. 37. Updated Base Model Architecture

Fig. 38. Updated Base model accuracy on train and test sets

TABLE III
CONFUSION MATRIX FOR UPDATED MODEL ON TEST SET

545	39	82	40	16	18	37	15	154	54
42	647	11	18	6	8	32	14	89	133
104	22	337	94	83	153	109	51	29	18
39	19	69	395	56	192	76	76	43	35
38	15	68	90	404	62	108	159	32	24
20	11	73	201	44	499	30	92	15	15
24	19	53	94	40	31	691	6	17	25
10	9	39	65	59	102	12	635	13	56
67	53	8	27	2	7	21	9	740	66
41	102	13	33	14	7	19	45	77	649

C. ResNet Model

The actual resnet model is quite big and difficult to train so, I used the residual network blocks from the ResNet paper in my implementation. The model architecture I used is present in Figure 40. The model was initially trained using Stochastic Gradient Descent as the Loss function with an LR of 0.001 and a batch size of 64 images. Howevevr, the convergence of the model parameters was quite slow. Hence, I had to replace it with AdamW loss function while keeping the parameters same. The inputs to the model were normalized here as well to attain a better accuracy. The model achieved a final accuracy of 75.32% and 60.68% on the train and test sets respectively.

Fig. 39. Updated Base model loss on train and test sets

TABLE IV
CONFUSION MATRIX FOR UPDATED MODEL ON TRAIN SET

2751	248	354	179	82	69	143	96	820	258
177	3257	60	93	76	24	158	87	427	641
535	92	1925	489	369	590	466	251	178	105
167	99	284	2124	211	1026	421	300	172	196
274	112	368	365	2170	246	458	701	128	178
82	42	340	1055	206	2489	188	435	64	99
121	124	257	434	220	128	3469	58	88	101
73	59	159	335	362	422	88	3212	39	251
380	245	67	147	48	34	97	32	3619	331
170	530	52	146	61	60	84	172	338	3387

TABLE V
CONFUSION MATRIX FOR RESNET ON TRAIN SET

3882	34	446	87	123	54	59	167	124	24
683	3239	64	101	140	47	107	161	122	336
296	5	2994	264	589	238	358	213	39	4
146	16	367	2082	566	1132	322	320	29	20
164	4	222	157	3564	166	205	495	10	13
44	10	238	664	458	3079	118	368	9	12
77	18	277	322	443	175	3588	78	15	7
67	4	112	117	597	299	25	3762	5	12
930	66	142	89	172	12	37	60	3437	55
552	335	54	124	169	58	50	436	85	3137

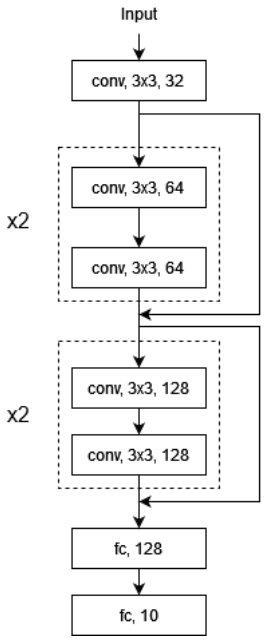

Fig. 40. ResNet Model Architecture

Fig. 41. ResNet accuracy on train and test sets

TABLE VI
CONFUSION MATRIX FOR RESNET ON TEST SET

724	10	115	14	34	8	14	35	40	6
164	582	16	30	32	11	23	35	22	85
59	2	539	46	136	77	81	51	7	2
38	3	75	391	123	224	67	69	4	6
24	3	62	28	641	35	80	119	4	4
17	5	58	155	94	565	29	68	5	4
14	6	77	65	107	39	675	12	3	2
20	2	23	22	124	66	15	723	1	4
200	21	26	20	36	4	12	16	647	18
116	88	10	28	42	14	12	85	24	581

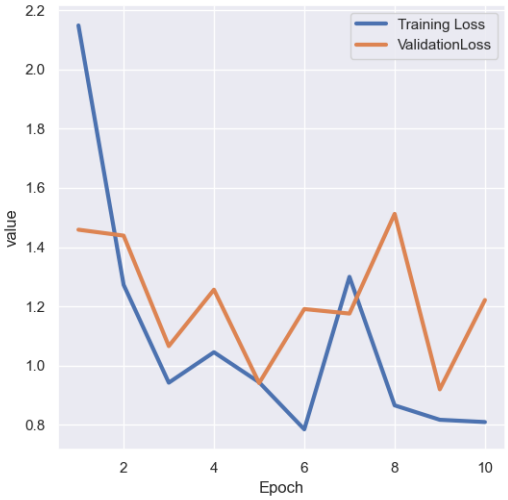

Fig. 42. ResNet loss on train and test sets

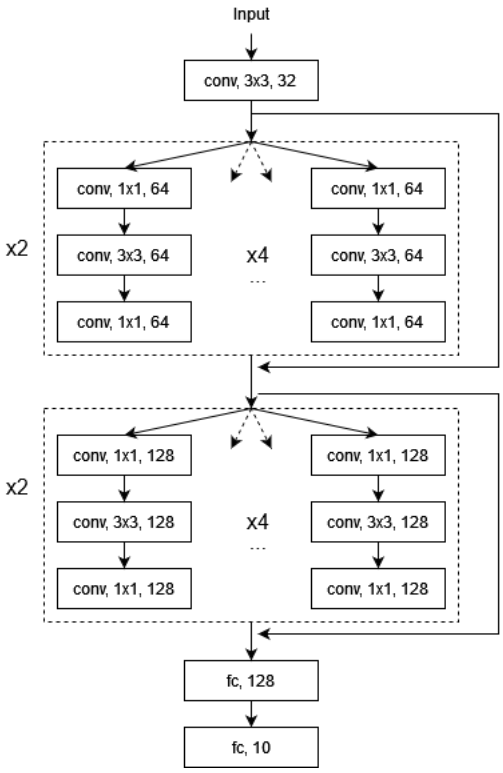

Fig. 43. ResNext Model Architecture

D. ResNext Model

The model architecture is present in Figure 43. It uses AdamW loss function with an LR of 0.001 and a batch size of 64 images. The inputs to the model were also normalized to a range of [-1,1]. The model achieved a maximum accuracy of 92.19% and 79.69% on the train and test sets respectively.

Fig. 44. ResNext accuracy on train and test sets

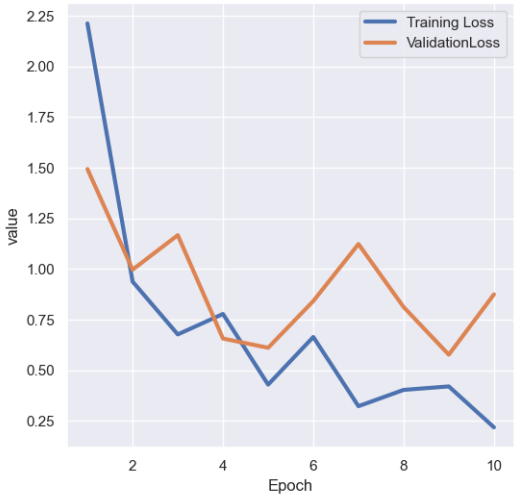

Fig. 45. ResNext loss on train and test sets

TABLE VII
CONFUSION MATRIX FOR RESNEXT ON TEST SET

815	17	55	6	22	5	6	15	45	14
28	878	4	12	2	5	11	7	22	31
63	3	672	36	62	50	64	31	12	7
27	8	63	500	56	155	89	55	21	26
22	4	77	35	699	31	39	77	8	8
11	4	52	103	42	674	25	63	11	15
13	8	41	39	26	20	817	10	17	9
33	1	30	20	38	46	9	806	7	10
60	31	16	9	11	2	9	3	843	16
45	111	12	7	4	4	5	20	29	763

E. Inference

According to the graphs and Tables in Subsection from II-A to II-D it is observed that ResNext model has the best accuracy.

TABLE VIII
CONFUSION MATRIX FOR RESNEXT ON TRAIN SET

4502	33	158	31	46	23	14	25	119	49
66	4676	19	25	18	13	21	10	58	94
205	19	4119	75	157	124	152	82	36	31
114	20	213	3375	154	550	250	188	68	68
107	11	200	97	4092	97	111	240	27	18
49	11	200	431	115	3816	97	214	27	40
43	20	111	102	53	86	4478	29	46	32
63	10	80	83	96	64	10	4560	7	27
160	74	44	19	23	10	14	16	4601	39
120	379	27	25	13	12	12	35	65	4312

Even though, the models were trained only for 10 epochs, ResNext model was able to classify the CIFAR10 dataset with better accuracy as compared to rest of the models.

Also, looking at Table IX, it is evident that ResNext and ResNet models have very low parameter count owing to how the two models the sizes of the feature maps thereby reducing the number of paramters. However, they took more time to process an input image as compared to the base models. But the similar counter parts of the base models having similar number of layers as ResNet and ResNext models used in my implemetation, the Iterations/Sec trend would have favoured ResNet and ResNext. Accoring to my results, I think ResNext performs the best among the tested models because of it's better accuracy and low paramter count.

TABLE IX
TABLE SHOWING MODEL PARAMETER AND INFERENCE TIME

	Parameter Count	Iterations/Sec
Base Model	657080	689.62
Updated Base Model	657220	459.33
Resnet	110378	216.07
ResNext	126890	113.71

REFERENCES

[1] Filter Banks, https://en.wikipedia.org/wiki/Filter_bank