RBE549: Homework 0 - Alohomora

Edwin Clement
MS RBE
Email: eclement@wpi.edu

***Abstract*—The Results and conclusions for Homework 0: Alohamora**

I. Phase 1: Shake my Boundary

This section contains the methodology and results of using a simplified Probablity of Boundary(Pb-lite) to detect boundaries of images.

A. Filter Generation

The following filters were used to detect characteristics of the image:

1) Oriented Derivative of Gaussians:

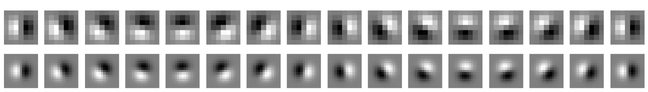


Fig. 1: DoG Filters

2) Leung-Malik Filters:

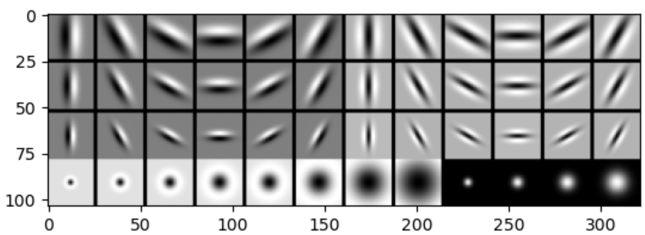


Fig. 2: LM Large

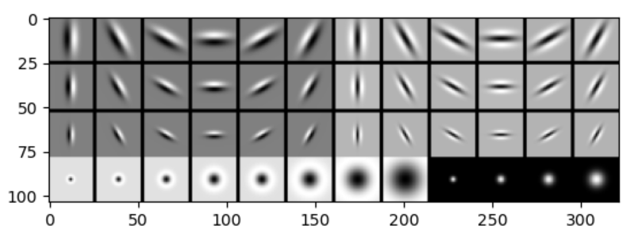


Fig. 3: LM Small

3) Gabor Filters:

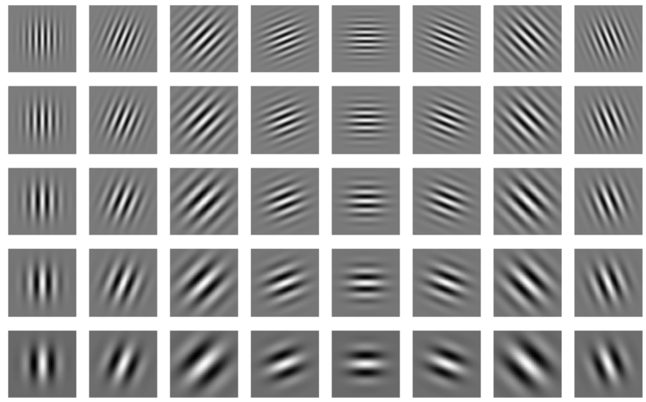
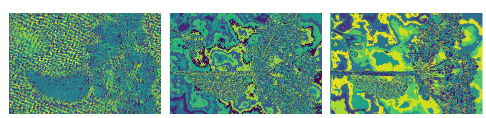


Fig. 4: LM Large

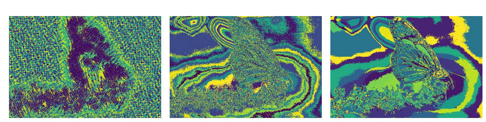
B. Calculating Texton, Brightness, Color Maps

All the images are subsequently run through all of these filters to extract features. The resultant data is of the shape $num\_filters \times width \times height$. K-means clustering is used to reduce the N-dimensional vector filter response at each pixel to a binned value. Here, I used 32 bins.



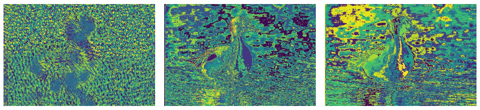
(a) Texton Map (b) Brightness Map (c) Color Map

Fig. 5: Maps for the 2st Image



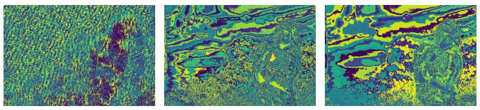
(a) Texton Map (b) Brightness Map (c) Color Map

Fig. 6: Maps for the 3st Image



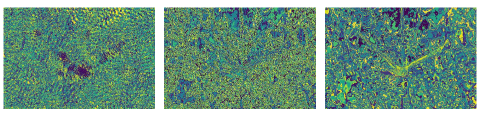
(a) Texton Map (b) Brightness Map (c) Color Map

Fig. 7: Maps for the 4st Image



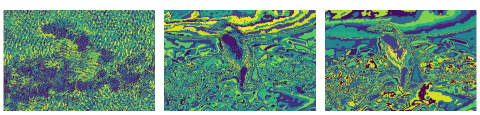
(a) Texton Map (b) Brightness Map (c) Color Map

Fig. 8: Maps for the 5st Image



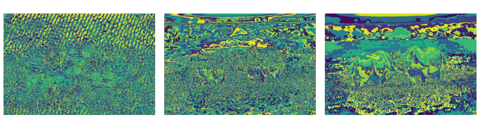
(a) Texton Map (b) Brightness Map (c) Color Map

Fig. 9: Maps for the 6st Image



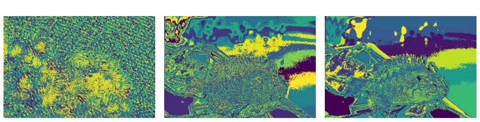
(a) Texton Map (b) Brightness Map (c) Color Map

Fig. 10: Maps for the 7st Image



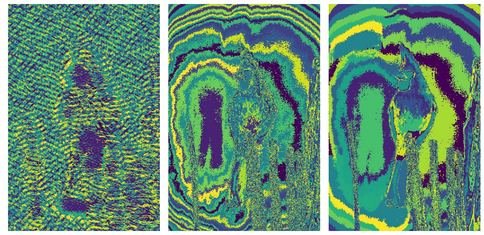
(a) Texton Map (b) Brightness Map (c) Color Map

Fig. 11: Maps for the 8st Image



(a) Texton Map (b) Brightness Map (c) Color Map

Fig. 12: Maps for the 9st Image



(a) Texton Map (b) Brightness Map (c) Color Map

Fig. 13: Maps for the 10st Image

C. Calculating Gradients

The 3 gradient maps: Color, Brightness, and Texture are obtained by calculating χ^2 distances using half-disk masks. The masks are generated using simple math and Numpy logical AND.

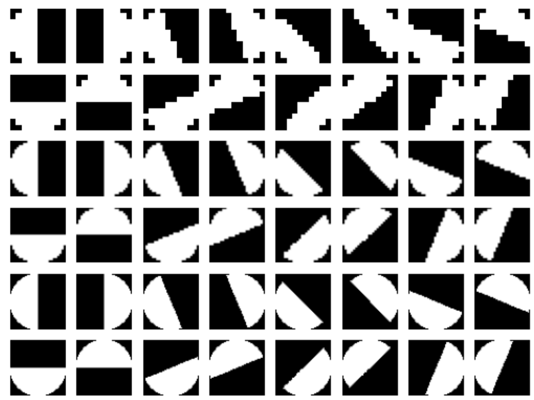
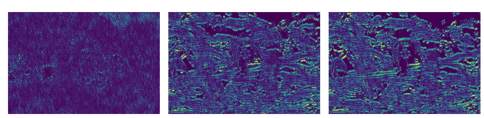


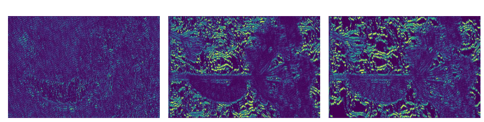
Fig. 14: Half Circle Masks



(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

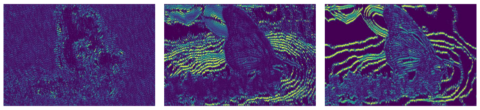
Fig. 15: Maps for the 1st Image

The Resulting gradients are as follows



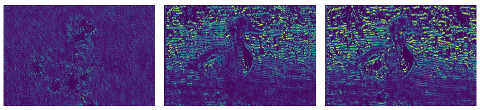
(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

Fig. 16: Maps for the 2st Image



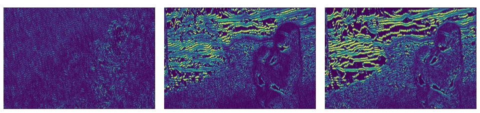
(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

Fig. 17: Maps for the 3st Image



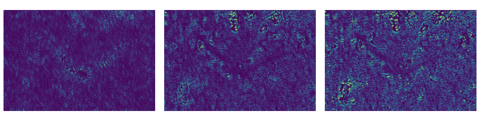
(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

Fig. 18: Maps for the 4st Image



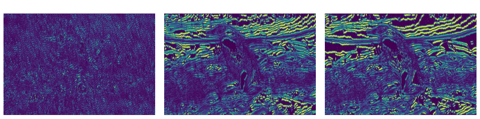
(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

Fig. 19: Maps for the 5st Image



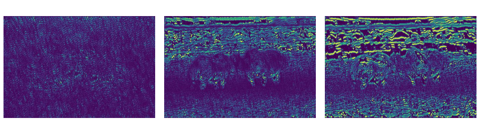
(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

Fig. 20: Maps for the 6st Image



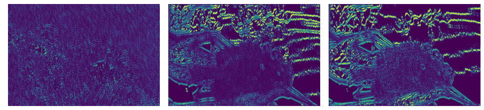
(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

Fig. 21: Maps for the 7st Image



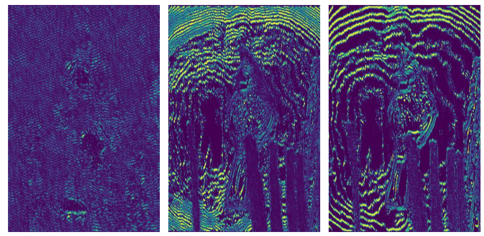
(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

Fig. 22: Maps for the 8st Image



(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

Fig. 23: Maps for the 9st Image

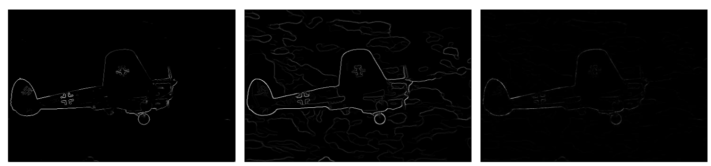


(a) Texton Gradient (b) Brightness Gradient (c) Color Gradient

Fig. 24: Maps for the 10st Image

D. *Results*

The following are the image results compared to Sobel and Canny.

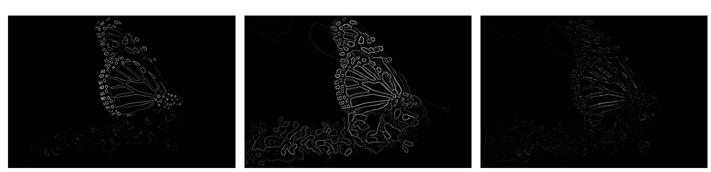


(a) Sobel (b) Canny (c) Pblite

Fig. 25: 1st Image

(a) Sobel (b) Canny (c) Pblite

Fig. 26: 2nd Image



(a) Sobel (b) Canny (c) Pblite

Fig. 27: 3rd Image

(a) Sobel (b) Canny (c) Pblite

Fig. 28: 4rth Image

(a) Sobel (b) Canny (c) Pblite

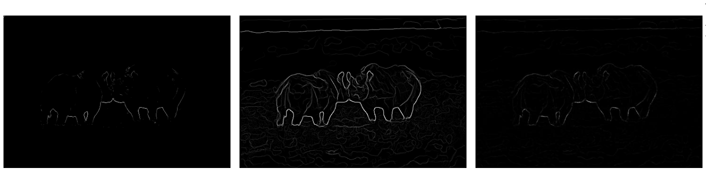
Fig. 29: 5th Image

(a) Sobel (b) Canny (c) Pblite

Fig. 30: 6th Image

(a) Sobel (b) Canny (c) Pblite

Fig. 31: 7th Image

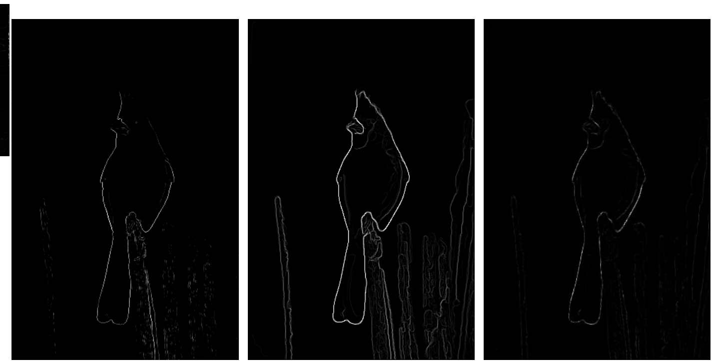


(a) Sobel (b) Canny (c) Pblite

Fig. 32: 8th Image

(a) Sobel (b) Canny (c) Pblite

Fig. 33: 9th Image



(a) Sobel (b) Canny (c) Pblite

Fig. 34: 10th Image

E. *Conclusion*

For any given image, the resultant edge detection with pb-lite is far less noisy.

II. Phase 2: Deep Dive on Deep Learning

A. *3.3 First Neural Network*

The first attempt I made after researching consists of 3 blocks of 2d Convolution layers separated by a transitional block. The NN is then terminated with a fully-connected Linear network that classifies the image into 10 labels. The network diagram is as follows.

Parameters	33
Optimizer	SGD(lr=0.01)
Batch Size	256
Num Epochs	10

TABLE I: Specific Details

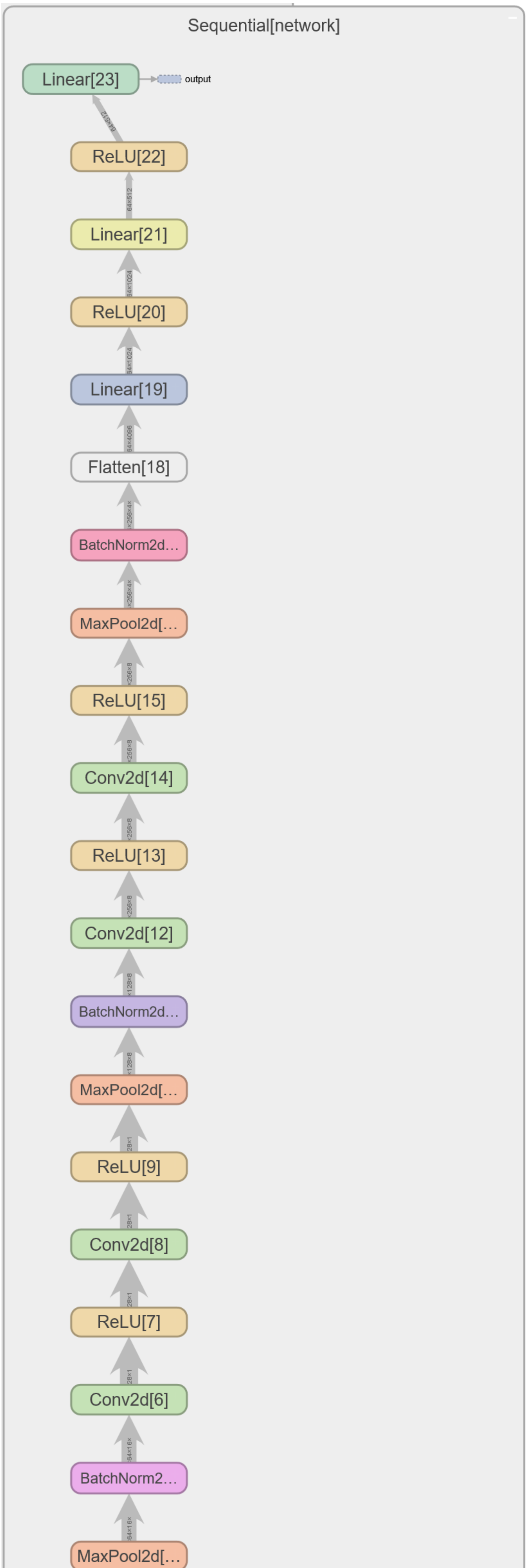


Fig. 35: Basic Neural Network

The Loss and Accuracy curves while training are as follows:

Fig. 36: CNN Accuracy Curve

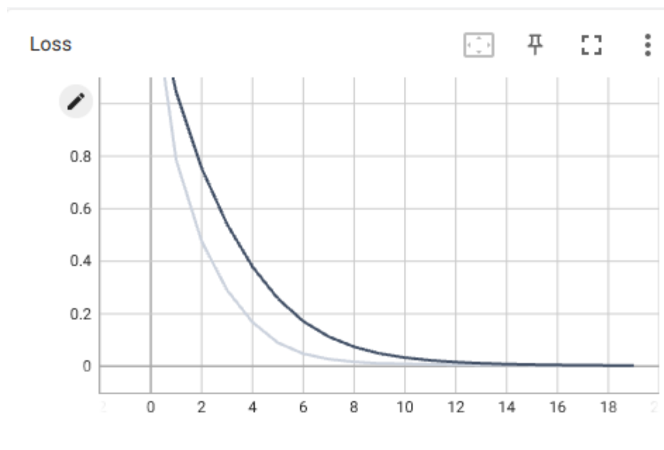


Fig. 37: CNN Loss/iteration Curve

I was able to obtain 80.03% accuracy on the testing data with this model. The confusion matrix for training and test cases are as follows:

	0	1	2	3	4	5	6	7	8	9
0	5000	0	0	0	0	0	0	0	0	0
1	0	5000	0	0	0	0	0	0	0	0
2	0	0	5000	0	0	0	0	0	0	0
3	0	0	0	5000	0	0	0	0	0	0
4	0	0	0	0	5000	0	0	0	0	0
5	0	0	0	0	0	5000	0	0	0	0
6	0	0	0	0	0	0	5000	0	0	0
7	0	0	0	0	0	0	0	5000	0	0
8	0	0	0	0	0	0	0	0	5000	0
9	0	0	0	0	0	0	0	0	0	5000

Fig. 38: CNN - Confusion on Training Data

	0	1	2	3	4	5	6	7	8	9
0	835	13	34	9	16	6	8	4	41	34
1	15	882	2	4	2	3	6	6	15	65
2	60	1	697	45	67	39	46	30	7	8
3	14	8	49	647	58	136	45	20	11	12
4	20	2	57	47	766	27	29	44	7	1
5	12	2	35	131	39	719	15	39	4	4
6	5	4	36	39	29	14	865	3	4	1
7	13	2	31	27	43	47	7	822	0	8
8	40	16	7	4	6	5	4	3	896	19
9	26	52	6	5	3	3	5	9	17	874

Fig. 39: CNN - Confusion on Testing Data

B. 3.4 Improving First Neural Network

The training process was improved via adding augmentation of the images before training. With these augmentations, there is no improvement in accuracy which stays at 79.9%. I can only conclude that the neural network understands the characteristics of the training data. The following steps were taken:

1) Randomly cropping after adding a padding of 4 on all sides
2) Randomly Flipping the Image
3) Normalizing the image

The following are the results of the experiment

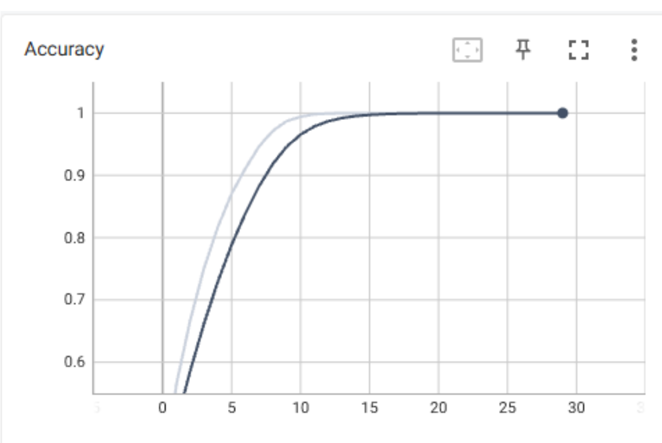


Fig. 40: CNN Accuracy Curve

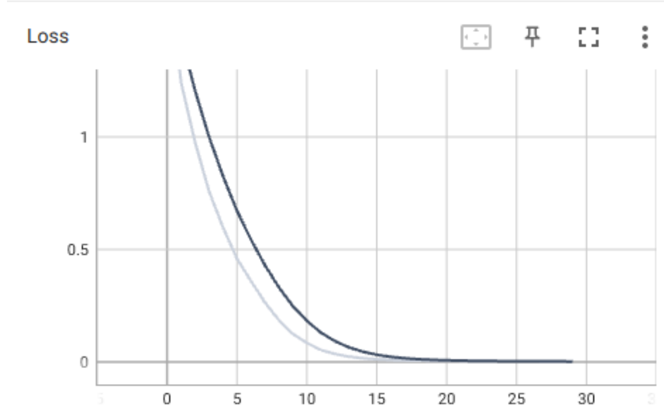


Fig. 41: CNN Loss/iteration Curve

	0	1	2	3	4	5	6	7	8	9
0	5000	0	0	0	0	0	0	0	0	0
1	0	5000	0	0	0	0	0	0	0	0
2	0	0	5000	0	0	0	0	0	0	0
3	0	0	0	5000	0	0	0	0	0	0
4	0	0	0	0	5000	0	0	0	0	0
5	0	0	0	0	0	5000	0	0	0	0
6	0	0	0	0	0	0	5000	0	0	0
7	0	0	0	0	0	0	0	5000	0	0
8	0	0	0	0	0	0	0	0	5000	0
9	0	0	0	0	0	0	0	0	0	5000

Fig. 42: CNN - Confusion on Training Data

	0	1	2	3	4	5	6	7	8	9
0	822	12	29	10	18	6	12	7	51	33
1	20	865	2	9	0	3	3	5	28	65
2	67	2	673	53	67	42	53	29	7	7
3	20	8	61	615	52	140	49	26	12	17
4	25	2	50	58	736	29	32	55	9	4
5	16	5	24	151	32	710	11	44	4	3
6	8	3	46	47	27	15	837	8	4	5
7	19	1	37	34	37	49	5	808	1	9
8	50	25	8	6	7	3	2	2	876	21
9	34	56	6	13	2	5	4	13	19	848

Fig. 43: CNN - Confusion on Testing Data

C. 3.5 Other Architectures

1) ResNet: The ResNET architecture was developed as a response to the issue of vanishing gradient. When a NN is deep enough, the calculated gradients used to adjust/optimize it become miniscule and training further leads to no accuracy improvements. ResNet thus introduces a connection between from the start of the block to the end without any computation so that the some of the features of initial layers reach later layers. Accuracy Obtained on Testing Data is 82.31 %, A minor improvement over CNN.

Parameters	218
Optimizer	SGD(lr=0.01, weight_decay=0.001, momentum=0.9)
Batch Size	128
Num Epochs	15

TABLE II: Specific Details

The following are the results of the experiment

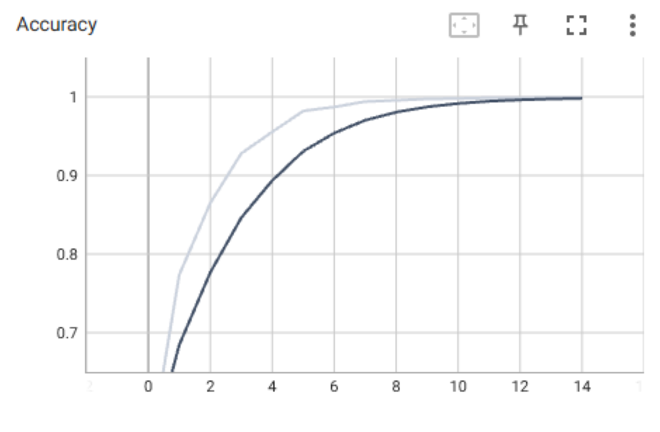


Fig. 44: ResNet Accuracy Curve

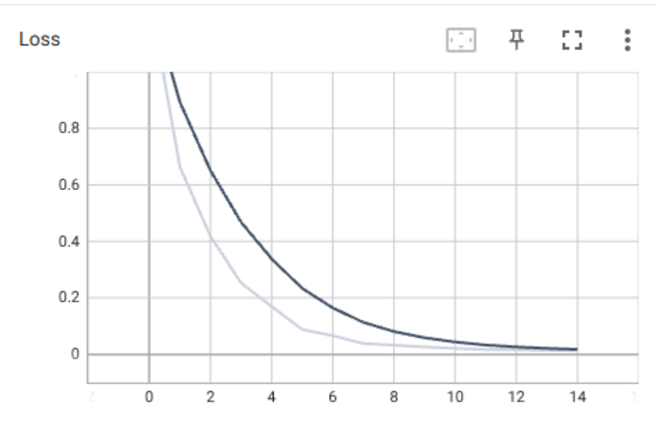


Fig. 45: ResNet Loss/iteration Curve

	0	1	2	3	4	5	6	7	8	9
0	4951	0	28	2	2	3	0	1	9	4
1	8	4964	1	3	0	1	9	1	3	10
2	8	0	4949	20	1	11	8	2	1	0
3	6	0	20	4886	17	57	6	7	0	1
4	4	1	119	27	4802	13	29	5	0	0
5	2	0	30	45	10	4898	5	10	0	0
6	0	0	32	83	0	7	4877	0	1	0
7	2	0	16	4	11	6	2	4959	0	0
8	28	2	3	2	0	2	5	0	4951	7
9	10	5	2	1	3	0	4	8	1	4966

Fig. 46: ResNet - Confusion on Training Data

	0	1	2	3	4	5	6	7	8	9
0	867	4	44	17	5	8	5	6	33	11
1	15	909	6	4	2	3	6	5	16	34
2	41	0	780	53	31	35	33	18	8	1
3	11	2	75	703	36	114	30	20	6	3
4	10	4	104	52	736	26	41	24	3	0
5	5	1	48	126	22	757	8	31	1	1
6	8	0	50	82	11	10	833	3	2	1
7	10	1	34	30	34	36	2	852	0	1
8	40	8	18	8	6	2	6	0	902	10
9	22	39	7	8	6	3	5	6	12	892

Fig. 47: ResNet - Confusion on Testing Data

2) ResNext: ResNext is a further evolved form of ResNet. Here instead of just sequentially appending layers, layers are stacked parallel as well. The number of stacked layers is defined by a new parameter: Cardinality.

With this approach, I was able to get 72.99 % accuracy on testing data

Parameters	188
Optimizer	SGD(lr=0.001, weight_decay=0.001, momentum=0.9)
Batch Size	32
Num Epochs	10

TABLE III: Specific Details

The following are the results of the experiment

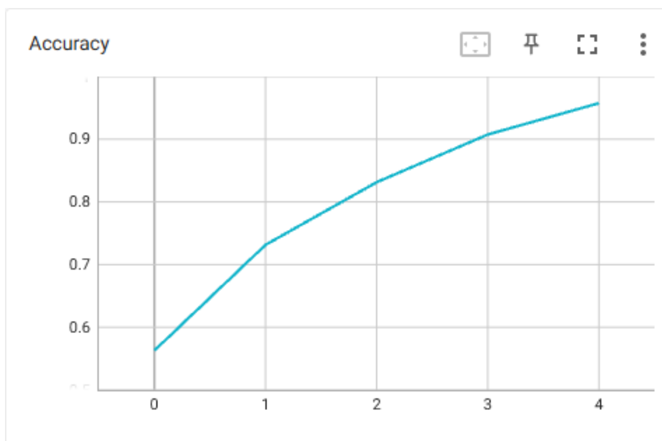


Fig. 48: ResNext Accuracy Curve

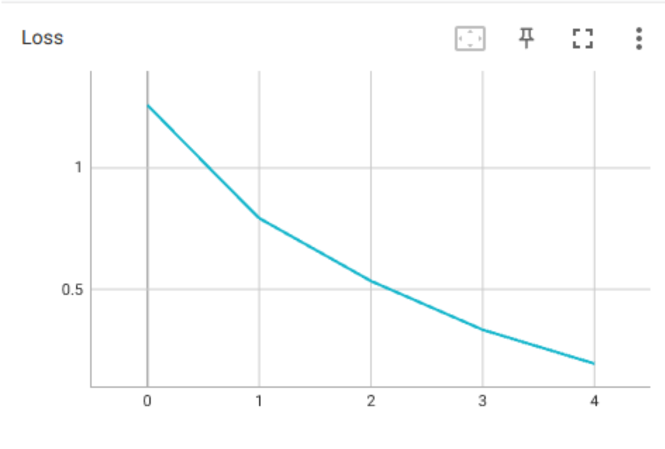


Fig. 49: ResNext Loss/iteration Curve

	0	1	2	3	4	5	6	7	8	9
0	4153	9	301	128	130	14	8	77	169	11
1	63	4613	58	48	23	27	9	68	32	59
2	48	0	4581	121	142	25	26	51	6	0
3	3	0	124	4520	68	199	27	52	6	1
4	3	0	113	126	4543	40	20	154	1	0
5	1	0	91	347	56	4355	16	130	4	0
6	4	0	178	211	97	53	4422	28	6	1
7	3	0	59	88	51	31	2	4765	1	0
8	31	12	57	90	40	8	5	19	4731	7
9	47	44	43	135	38	30	8	178	40	4437

Fig. 50: ResNext - Confusion on Training Data

	0	1	2	3	4	5	6	7	8	9
0	650	9	98	47	45	9	9	33	87	13
1	29	791	10	35	13	14	7	25	17	59
2	35	0	692	77	93	34	27	35	6	1
3	12	1	95	651	43	123	27	42	5	1
4	4	0	90	69	710	32	18	74	2	1
5	4	0	45	211	25	634	11	68	2	0
6	6	1	83	90	69	29	698	19	5	0
7	8	0	30	56	33	33	2	835	0	3
8	22	11	20	39	16	2	4	5	871	10
9	28	33	20	44	11	8	2	63	24	767

Fig. 51: ResNext - Confusion on Testing Data

3) DenseNet: Densenet is a further improvement in flow of information from previous layers. In this network design, a layer has access to all the feature extractions of all previous layers, hence the name dense referring to a lot of inter-layer connection

With this approach, I was able to get 83.87 % accuracy on testing data. Furthermore, densenet trained much quicker than resnext.

Parameters	596
Optimizer	SGD(lr=0.1, momentum=0.9)
Batch Size	64
Num Epochs	10

TABLE IV: Specific Details

The following are the results of the experiment

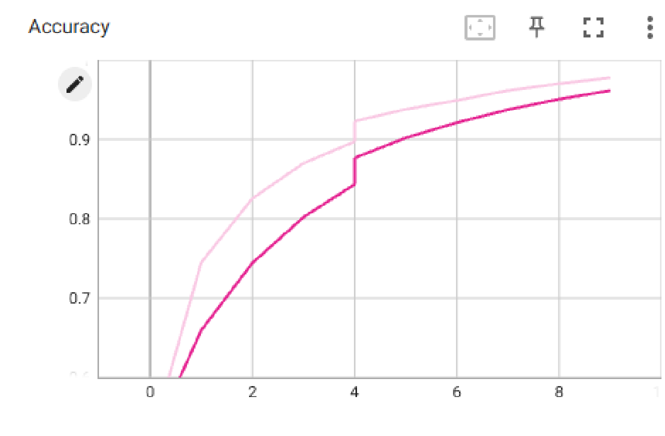


Fig. 52: DenseNet Accuracy Curve

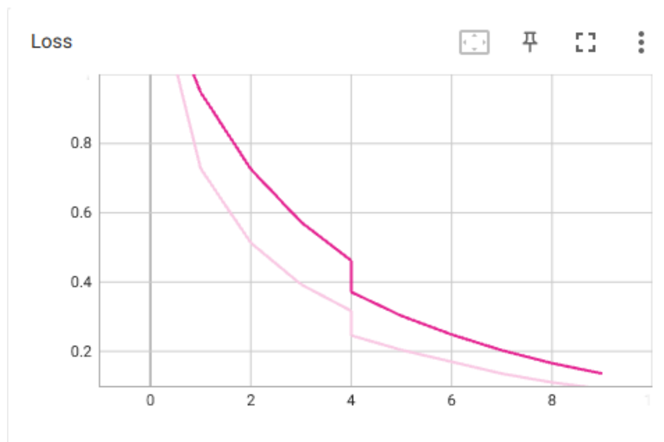


Fig. 53: DenseNet Loss/iteration Curve

	0	1	2	3	4	5	6	7	8	9
0	4845	48	36	15	1	5	3	3	26	18
1	0	4992	1	0	0	2	2	0	2	1
2	117	5	4694	43	11	47	65	11	5	2
3	39	14	69	4281	50	366	143	17	14	7
4	57	4	91	72	4501	70	70	131	2	2
5	12	5	48	147	24	4680	36	44	2	2
6	9	9	28	17	6	19	4910	0	1	1
7	29	18	28	42	12	111	15	4734	4	7
8	59	57	7	3	0	3	10	0	4846	15
9	45	299	3	5	0	2	6	1	23	4616

Fig. 54: DenseNet - Confusion on Training Data

	0	1	2	3	4	5	6	7	8	9
0	895	20	23	10	2	3	6	1	26	14
1	4	975	1	1	0	0	0	1	4	14
2	56	3	779	33	28	40	51	6	3	1
3	20	6	45	658	34	157	53	14	9	4
4	17	4	55	30	756	29	42	63	4	0
5	10	7	25	84	23	815	15	20	0	1
6	10	2	29	11	7	10	927	3	1	0
7	24	13	14	25	17	52	7	845	1	2
8	44	27	9	5	0	1	7	0	897	10
9	24	107	3	1	3	1	2	2	17	840

Fig. 55: DenseNet - Confusion on Testing Data

D. Analysis

Augmentation did not have as much impact as I expected. This is probably due to the high initial scoring of the initial CNN. Densenet trains much faster and considering that accuracy of all methods lie within 10% of each other, DenseNet leads the way in terms of practical utility.

	CNN	AugCNN	Resnet	ResNext	DenseNet
Parameters	33	33	218	188	596
Test Acc	80.03	79.9	82.31	72.99	83.87
Train Acc	100	100	98.406	90.24	94.198

REFERENCES

[1] https://medium.com/jun94-devpblog/cv-3-gradient-and-laplacian-filter-difference-of-gaussians-dog-7c22e4a9d6cc
[2] https://discourse.vtk.org/t/gaussian-filter-and-python/2529/3
[3] https://medium.com/@akumar5/computer-vision-gaussian-filter-from-scratch-b485837b6e09
[4] https://www.robots.ox.ac.uk/ vgg/research/texclass/filters.html
[5] https://medium.com/@rajilini/laplacian-of-gaussian-filter-log-for-image-processing-c2d1659d5d2
[6] https://sgugger.github.io/convolution-in-depth.html
[7] https://www.analyticsvidhya.com/blog/2021/09/convolutional-neural-network-pytorch-implementation-on-cifar10-dataset/
[8] https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
[9] https://shonit2096.medium.com/cnn-on-cifar10-data-set-using-pytorch-34be87e09844
[10] https://archive.is/F65Ul
[11] https://cs231n.github.io/
[12] https://github.com/facebookarchive/fb.resnet.torch/issues/180
[13] https://blog.paperspace.com/writing-resnet-from-scratch-in-pytorch/
[14] https://medium.com/dataseries/enhancing-resnet-to-resnext-for-image-classification-3449f62a774c
[15] https://drago1234.github.io/about_me/pdf/CS5194_ResNet_v2.0.pdf
[16] https://github.com/andreasveit/densenet-pytorch/blob/master/train.py
[17]
[18] https://medium.com/jun94-devpblog/cv-3-gradient-and-laplacian-filter-difference-of-gaussians-dog-7c22e4a9d6cc
[19] https://discourse.vtk.org/t/gaussian-filter-and-python/2529/3
[20] https://medium.com/@akumar5/computer-vision-gaussian-filter-from-scratch-b485837b6e09
[21] https://www.robots.ox.ac.uk/ vgg/research/texclass/filters.html
[22] https://medium.com/@rajilini/laplacian-of-gaussian-filter-log-for-image-processing-c2d1659d5d2
[23] https://sgugger.github.io/convolution-in-depth.html
[24] https://www.analyticsvidhya.com/blog/2021/09/convolutional-neural-network-pytorch-implementation-on-cifar10-dataset/
[25] https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
[26] https://shonit2096.medium.com/cnn-on-cifar10-data-set-using-pytorch-34be87e09844
[27] https://archive.is/F65Ul
[28] https://cs231n.github.io/
[29] https://github.com/facebookarchive/fb.resnet.torch/issues/180
[30] https://blog.paperspace.com/writing-resnet-from-scratch-in-pytorch/
[31] https://medium.com/dataseries/enhancing-resnet-to-resnext-for-image-classification-3449f62a774c
[32] https://drago1234.github.io/about_me/pdf/CS5194_ResNet_v2.0.pdf
[33] https://github.com/andreasveit/densenet-pytorch/blob/master/train.py