Homework0: Alohomora

P Sai Ramana Kiran
Email: spinnamaraju@wpi.edu
Using 1 late day

***Abstract*—This report documents the results for Phase I: classical techniques for edge detection and Phase II: modern techniques for object classification. Phase I part of the assignment explores generation of variety of filter kernels using mathematical equations and related techniques. These kernels are further used to abstract the edges of an image. Phase II part of the assignment explores using deep Convolution Neural Networks (CNNs) in object classification problem. Different models are implemented and a detailed analysis is presented**

I. Shake My Boundary

This part of the assignment explores classical methods for edge detection. It implements a "lite" version of probability of boundary detection algorithm

A. Generating Filter Banks

One of the major challenge in Phase I part of the assignment was to generate 2D skewed gaussians $P(x, y)$ and their derivatives $P'(x, y)$, $P''(x, y)$ centered at 0 mean μ and varying variance σ. This was achieved by assuming that distributions of gaussian along each dimension is independent of each other as denoted in 1

$$P(x, y) = P(x)P(y) \tag{1}$$

$$P(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2} \tag{2}$$

Using this assumption, their nth partial derivatives can be easily computed as shown in 3 and 4

$$P_x^n(x, y) = P_x^n(x)P(y) \tag{3}$$

$$P_y^n(x, y) = P(x)P_y^n(y) \tag{4}$$

Moreover, to get probability distribution of a gaussian rotated by an angle of θ is achieved using rotation transformation as denoted in 5

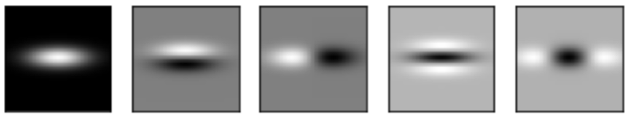
$$P(x_r, y_r) = P(xcos\theta + ysin\theta)P(ycos\theta - xsin\theta) \tag{5}$$

Combining above equations will simplify the job of generating gaussians of different variances (σ_x and σ_y) at different rotation (θ). A sample set of gaussians and it's derivatives are shown in 1

Now that a fundamental component of gaussians is created, following filter banks which are combinations of orientations and gaussian derivatives are generated.

1) Oriented Derivative Of Gaussians: As described in the problem statement, these filters are generated by convolving sobel kernels with gaussians of different sizes and rotating them. Filterbanks with 2 scales and 12 orientations is shown in figure 2

```
<Figure size 1600x1600 with 0 Axes>
```



```
<Figure size 1600x1600 with 0 Axes>
```

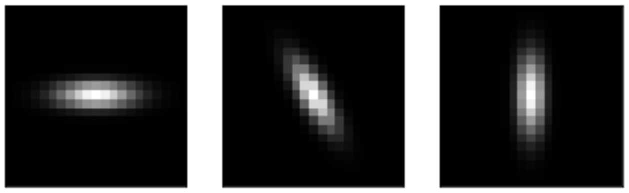


Fig. 1: 1st and 2nd derivative gaussians and rotated gaussians

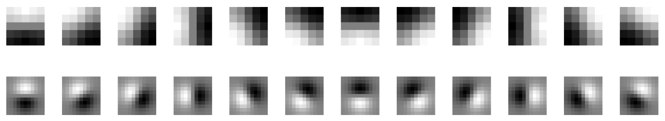


Fig. 2: Oriented Derivative of Gaussian

2) Leung-Malik Filters: Using the concepts mentioned above, generating Leung-Malik filters are straightforward. Figure 3 shows 96 LM filters, combining LM Small and LM Large filters

3) Gabor Filters: These filters which are approximated versions of how human visual system is generated with 4 scales and 7 orientations. Idea was to cover as many orientations and scales as possible while keeping computational feasibility in mind. Figure 4 show different gabor filters used

B. Clustering Feature maps

Now, we use the N filters generated in the above filter banks to convolve image of interest, which results in a series of N convoluted images representing various texture patterns. Along with these texture patterns, brightness and color images are clustered to create a texton map, brightness cluster, color cluster

1) Texton Map τ: Again, as mentioned in the problem statement, Texton maps τ are generated by applying K -

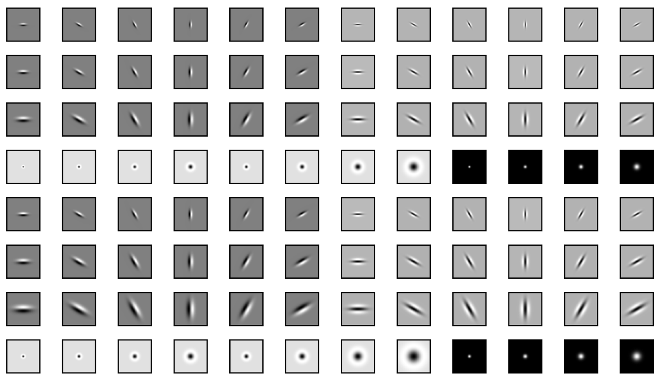


Fig. 3: Leung Malik Filters

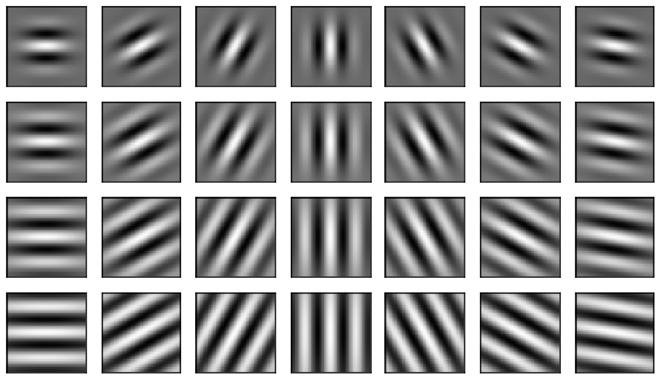


Fig. 4: Gabor Filters

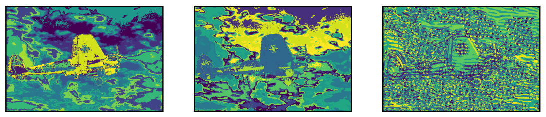
means clustering on the filtered maps. 80 bins were used in the clustering and right most image in figures 5a to 6e show different texton maps

2) Brightness Map B: B is created with similar methodology as τ map, except 16 bins were used instead of 80 bins. Left image in figures 5a to 6e represents results of various images with 16 bins

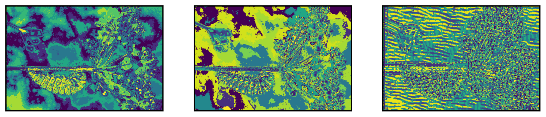
3) Color Map C: Center images in figures 5a to 6e represents results of various color maps. Clustering was done using 3 channels as feature maps and 16 bins

C. Calculating Gradients

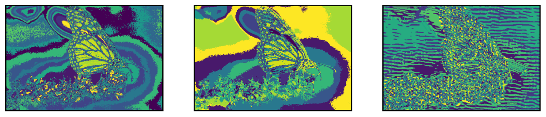
Gradients of different clustered feature maps was calculated using half disk mask pairs and filtering operation on feature matrices. Essentially, histograms of feature maps was calculated using half disk masks and a χ^2 distance is computed for each pair. So if we have N pairs of half disk masks, we would end up with N feature maps containing χ^2 distances at each pixel We use half disk masks to calculate gradients along these textured images



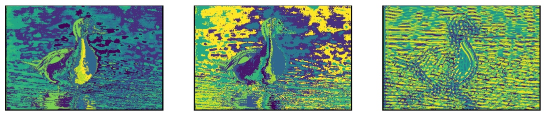
(a) image 1



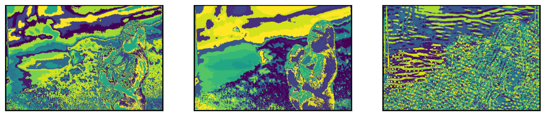
(b) image 2



(c) image 3



(d) image 4



(e) image 5

Fig. 5: Clustered (left to right) Brightness, Color and Texton maps of images 1 to 5

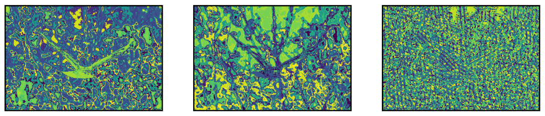
1) Half disk images: Half disk mask pairs with 3 scales $(10, 20, 30)$ and 7 orientations ranging from $0°$ to $145°$ are shown in 7

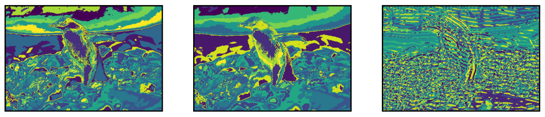
2) Gradients Brightness (B_g), Color (C_g), Texture (τ_g): Now, the 21 half disks masks generated is used to compute gradient of the brightness, color and Texton map. Results of which are shown in the figures 8a to 9e

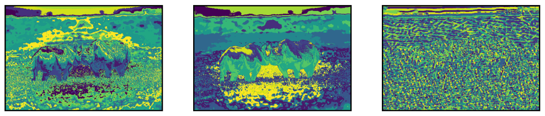
D. Pb-Lite Output and analysis

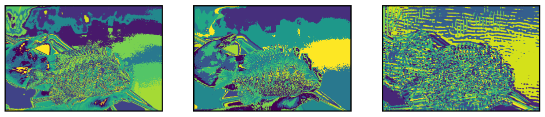
Combining above derived gradient maps along with Canny and Sobel baseline images gives the edges. as shown in 10. PbLite baseline clearly outperformed in many images and identified the boundaries successfully. For example, a comparison of pblite, sobel and canny can be seen in figure 11. PbLite in this case has accurately identified the aircraft alone, leaving the clouds unlike canny baseline. On the otherhand, sobel couldnt identify some aspects of the aircraft. However, there are few instances like figure 12 where pblite couldnt identify complete aspects of the animal.

That being said, most of the edge detection from pblite output is attributed to the canny and sobel baselines since Hadamard product is nullifying the pixels using 0s of canny and sobel baseline images. To make PbLite more standalone, more finer number of Gabor filters have to be used. All these filters have to be carefully selected since too many convolutions might create a noisy feature maps.


(a) image 6


(b) image 7


(c) image 8


(d) image 9

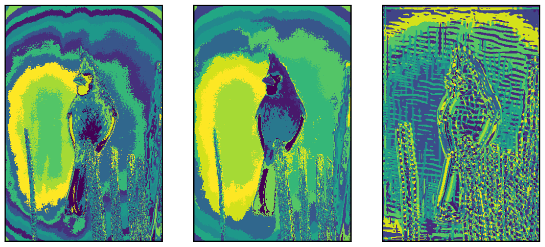

(e) image 10

Fig. 6: Clustered (left to right) Brightness, Color and Texton maps of images 6 to 10

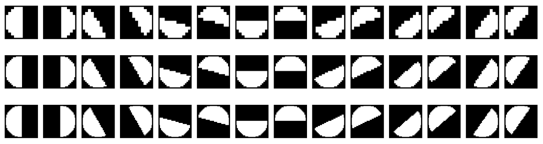
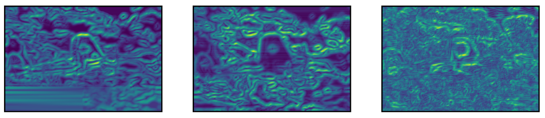
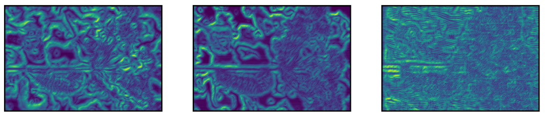
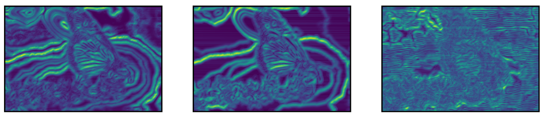
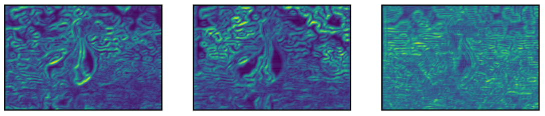


Fig. 7: Half Disk Masks


(a) image 1


(b) image 2


(c) image 3


(d) image 4

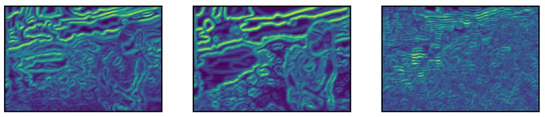

(e) image 5

Fig. 8: Gradients of (left to right) Brightness, Color and Texton maps of images 1 to 5

II. Deep Dive on Deep Learning

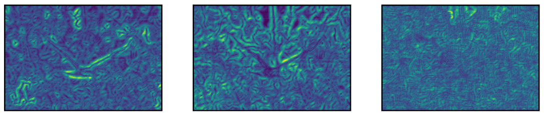
In this part of homework, modern techniques for image classification using Deep Convolutional Networks have been implemented.

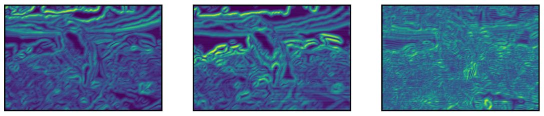
A. Base Model and Implementation details

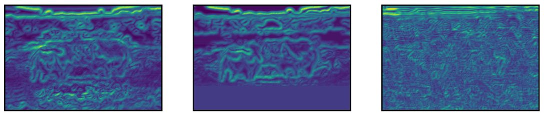
As a baseline model, an architecture inspired from VGG is implemented. Wherein, the number of convolution filters double ($\times 2$) as the network goes deep. Simultaneously, to capture more finer features of the images, the size of activation maps are halved (/2) using a Max pooling of size 2×2 in between the convolution filters. Each convolution filter is also passed to Rectified Linear Unit (ReLU). At the end the convolutions, a number of finer activation maps is flattened and fully connected to a linear layer. This in-turn is mapped to the classification layer. Architecture of the model is shown in figure 16. All the hyper-parameters and training results can be seen in figure 19. Comparison of training and testing accuracy over epochs can be found in figure 17, loss can be found in 18

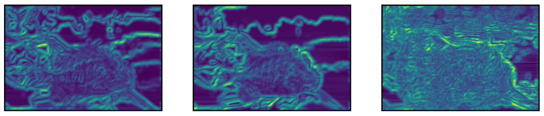
B. Improvements to Baseline Model

Few hyper parameters for improving the model prediction accuracy have been explored. Modified architecture can be seen in figure 15. Hyper-parameters and training results are summarized in the figure 19. Clearly with the following


(a) image 6


(b) image 7


(c) image 8


(d) image 9

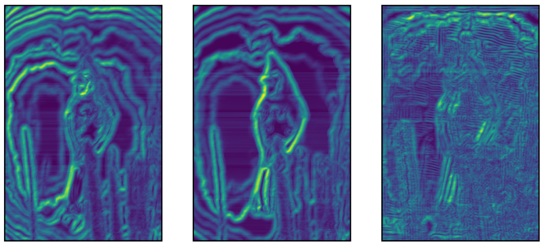

(e) image 10

Fig. 9: Gradients of (left to right) Brightness, Color and Texton maps of images 6 to 10

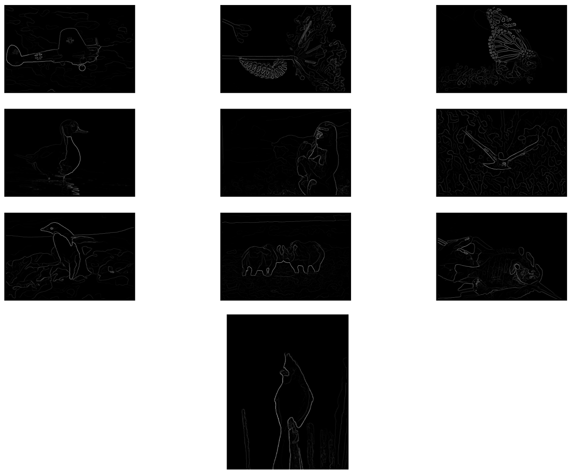


Fig. 10: Probability Lite Edge detection output

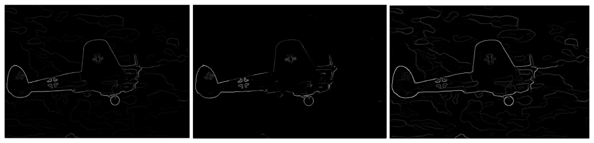


Fig. 11: Comparing pbLite, Sobel and Canny edges for image 1

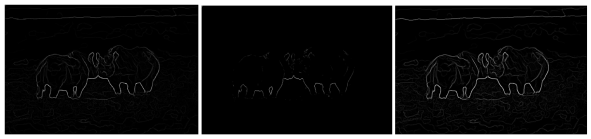


Fig. 12: Comparing pbLite, Sobel and Canny edges for image 8

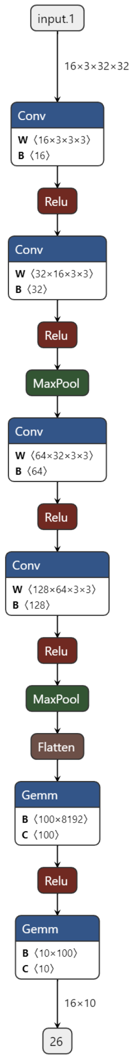
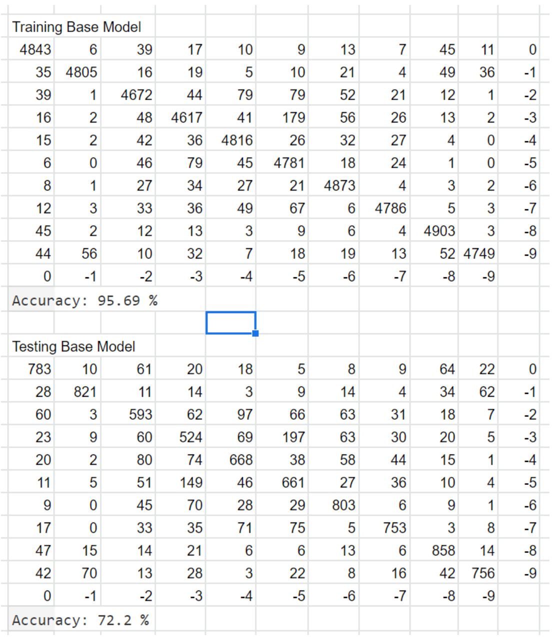


Fig. 13: Baseline model architecture

improvements, accuracy of the network improved by almost 10%!

1) learning rate decay: 2 variations, linear decay and step decay of the learning rate has been tried. However, both the decay algorithms didnt improve the base line model accuracy as shown in figure 17 with 'Learn Decay Model Accuracy' and 'Learn Decay Model Validation Accuracy' plots

2) Data Augmentation: Image data has been standardized at the beginning of the training (and testing) to a specific mean and variance. By standardizing the per pixel information, we are essentially making sure that weights are not chasing a moving target. Moreover, the images are resized to 64×64 from $32{\times}32$. Accordingly, the network parameters are changed



Training Base Model

4843	6	39	17	10	9	13	7	45	11	0
35	4805	16	19	5	10	21	4	49	36	-1
39	1	4672	44	79	79	52	21	12	1	-2
16	2	48	4617	41	179	56	26	13	2	-3
15	2	42	36	4816	26	32	27	4	0	-4
6	0	46	79	45	4781	18	24	1	0	-5
8	1	27	34	27	21	4873	4	3	2	-6
12	3	33	36	49	67	6	4786	5	3	-7
45	2	12	13	3	9	6	4	4903	3	-8
44	56	10	32	7	18	19	13	52	4749	-9
0	-1	-2	-3	-4	-5	-6	-7	-8	-9	

Accuracy: 95.69 %

Testing Base Model

783	10	61	20	18	5	8	9	64	22	0
28	821	11	14	3	9	14	4	34	62	-1
60	3	593	62	97	66	63	31	18	7	-2
23	9	60	524	69	197	63	30	20	5	-3
20	2	80	74	668	38	58	44	15	1	-4
11	5	51	149	46	661	27	36	10	4	-5
9	0	45	70	28	29	803	6	9	1	-6
17	0	33	35	71	75	5	753	3	8	-7
47	15	14	21	6	6	13	6	858	14	-8
42	70	13	28	3	22	8	16	42	756	-9
0	-1	-2	-3	-4	-5	-6	-7	-8	-9	

Accuracy: 72.2 %

Fig. 14: Baseline confusion matrix

to reflect the new image sizes

3) Batch normalization: Along with data augmentation, a batch normalization layer has been added in between convolutions. This also makes sure that layers are getting standardized inputs and weights dont move around too much from target. Both of these techniques have significantly improved the performance of the baseline model as can be seen in figure 17 with 'Batch Norm Model Accuracy' and 'Batch Norm Validation Accuracy' plots

C. ResNet,ResNeXt,DenseNet

A custom implementations of the architectures ResNet and ResNeXt are explored in this part of homework.

1) ResNet: In this variation of ResNet implementation, images are first resized from 32×32 to 64×64, similar to the previously improved baseline model. From here, the input is directly fed to the 'ResNetBlock' instead of first feeding to the convolution layer. This is because the first convolution layer was primarily used for downsampling the image and in this case images are small enough for them to act as input. Figure 20 is a screenshot from tensorboard, which reflects the 'ResNetBlock' implementation. Custom architecture is outlined in figure 21

Other hyper parameters and results are outlined in the figure 19. Along with the confusion matrix for test and training accuracy is given in figure 24. Training and testing accuracy per epoch can be found in 22 Clearly, this network outperforms the baseline model and it's variation by a considerable margin, despite being less number of training parameters. However, this comes at a cost of inference run time, where baseline model easily wins.

2) ResNeXt: This variation follows very similar design pattern from above, wherenin the first layer is the 'ResNeXt'

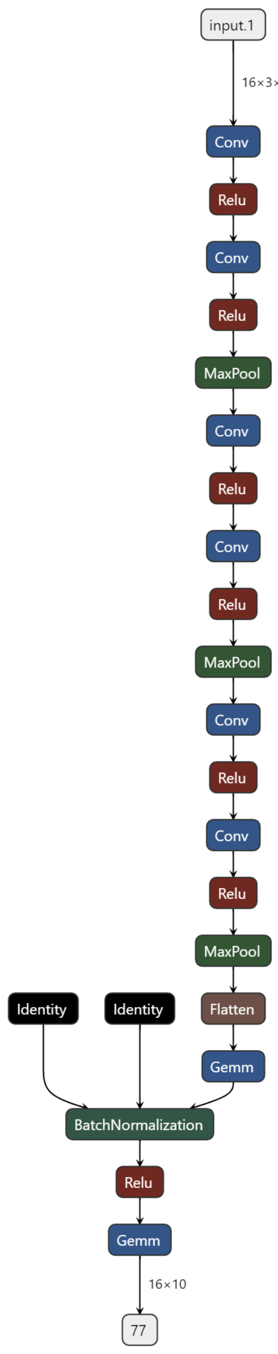
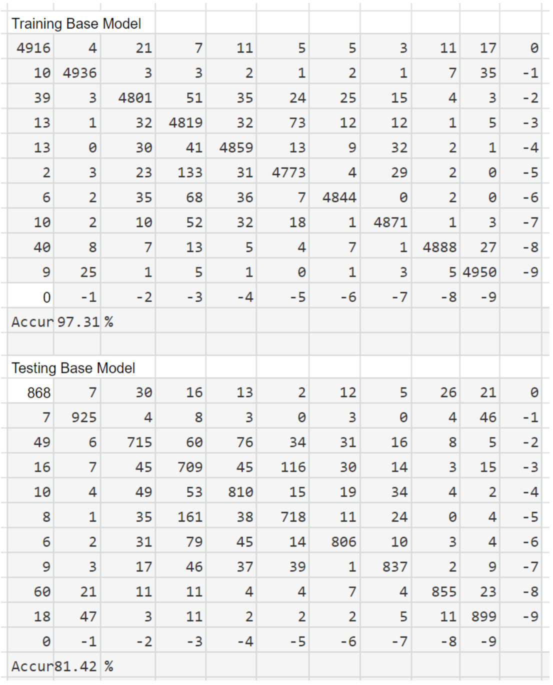


Fig. 15: Improved baseline model with batch normalization

block. This model uses the same input as above, where images are of size 64×64. Summary of architecture is outlined in figure 26. All the hyper parameters and results can be captured from the figure 19. From this figure, it can be inferred that despite the architecture having less number of the trainable parameters, it couldnt achieve the training and testing accuracy as good as custom ResNet implementation. This might be attributed to the lack of sufficient number of cardinal layers as indicated in the ResNeXt paper



Training Base Model										
4916	4	21	7	11	5	5	3	11	17	0
10	4936	3	3	2	1	2	1	7	35	-1
39	3	4801	51	35	24	25	15	4	3	-2
13	1	32	4819	32	73	12	12	1	5	-3
13	0	30	41	4859	13	9	32	2	1	-4
2	3	23	133	31	4773	4	29	2	0	-5
6	2	35	68	36	7	4844	0	2	0	-6
10	2	10	52	32	18	1	4871	1	3	-7
40	8	7	13	5	4	7	1	4888	27	-8
9	25	1	5	1	0	1	3	5	4950	-9
0	-1	-2	-3	-4	-5	-6	-7	-8	-9	
Accur	97.31	%								

Testing Base Model										
868	7	30	16	13	2	12	5	26	21	0
7	925	4	8	3	0	3	0	4	46	-1
49	6	715	60	76	34	31	16	8	5	-2
16	7	45	709	45	116	30	14	3	15	-3
10	4	49	53	810	15	19	34	4	2	-4
8	1	35	161	38	718	11	24	0	4	-5
6	2	31	79	45	14	806	10	3	4	-6
9	3	17	46	37	39	1	837	2	9	-7
60	21	11	11	4	4	7	4	855	23	-8
18	47	3	11	2	2	2	5	11	899	-9
0	-1	-2	-3	-4	-5	-6	-7	-8	-9	
Accur	81.42	%								

Fig. 16: Improved Baseline confusion matrix

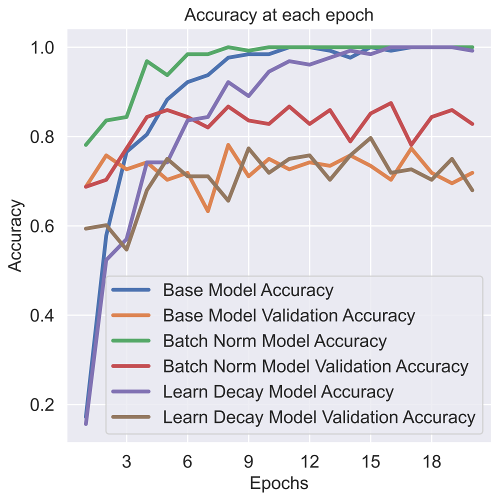


Fig. 17: Accuracy over different models

3) DenseNet: This variation connects every 'DenseNet-Block' with downstream blocks. However, unlike previous networks, it is not adding to the input but instead concatenating to the channel dimension. Thereby increasing the input channels as the number of blocks within the network go deep. These 'DenseNetBlock's together form a 'DenseNetLayer' as shown in figure 29. Architecture can be seen in the figure 30. There were multiple attempts at improving the densenet layer by modifying the image sizes, increasing the growth factor, also increasing the depth of a particular DenseNetLayer. However, not only the amount of time it took to train was massive, but training and testing results were not optimal. For example,

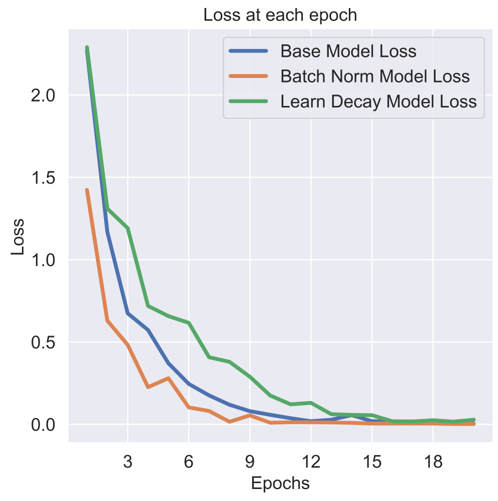
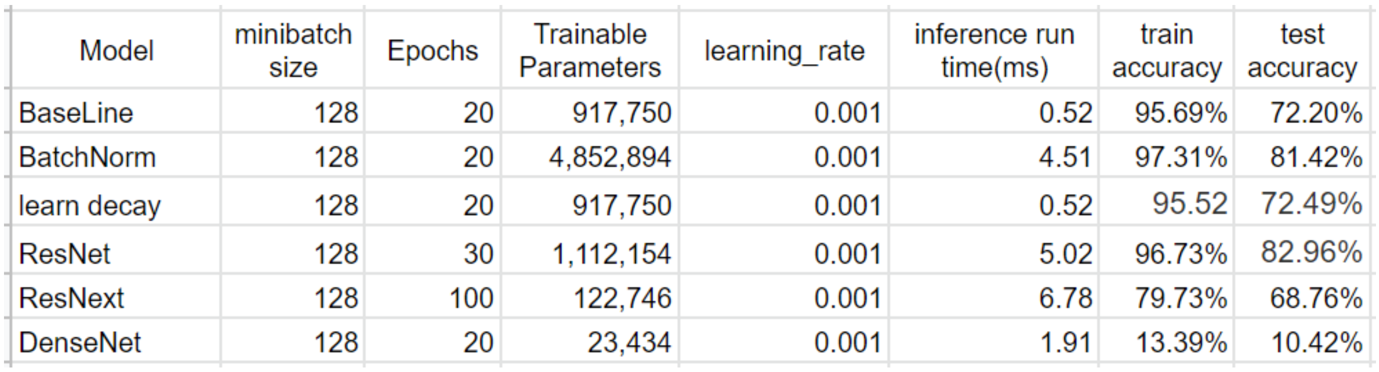


Fig. 18: Loss comparison over different models

in the initial attempt, it was trained with 4 layers of depth $[6, 12, 24, 8]$, with epochs as 100. Even with these parameters, network still gave a testing accuracy $< 10\%$. In subsequent runs, a modified and simplified layer have been tried with weights initialized using 'Kaiming Normal' Model. Even with this initialization, training couldnt be completed in time and results were suboptimal.

4) Phase II concluding remarks: Overall, the ResNet and improved baseline models worked really well. Still need to explore and tune many hyperparameters, optimizers and different layers to identify best combination to predict images from CIFAR-10. A better weight initialization for ResNeXt and DenseNet based off ImageNet models is very much desired. More methods to improve the inference time needs to explored since 4ms for a good prediction is not sustainable in high speed and latency sensitive environments



Model	minibatch size	Epochs	Trainable Parameters	learning_rate	inference run time(ms)	train accuracy	test accuracy
BaseLine	128	20	917,750	0.001	0.52	95.69%	72.20%
BatchNorm	128	20	4,852,894	0.001	4.51	97.31%	81.42%
learn decay	128	20	917,750	0.001	0.52	95.52	72.49%
ResNet	128	30	1,112,154	0.001	5.02	96.73%	82.96%
ResNext	128	100	122,746	0.001	6.78	79.73%	68.76%
DenseNet	128	20	23,434	0.001	1.91	13.39%	10.42%

Fig. 19: Comparison of models

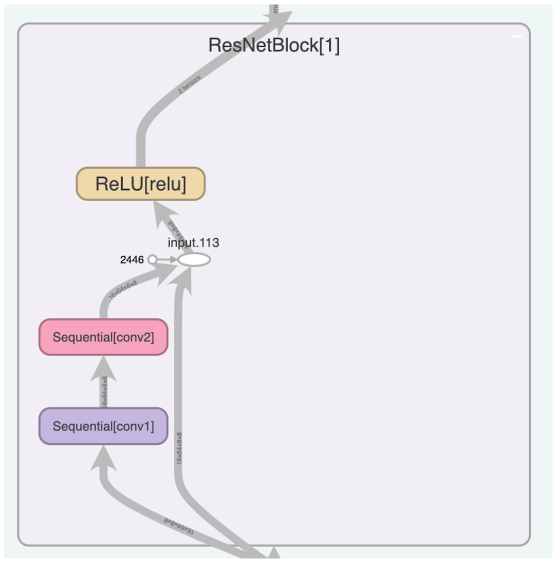


Fig. 20: ResNetBlock

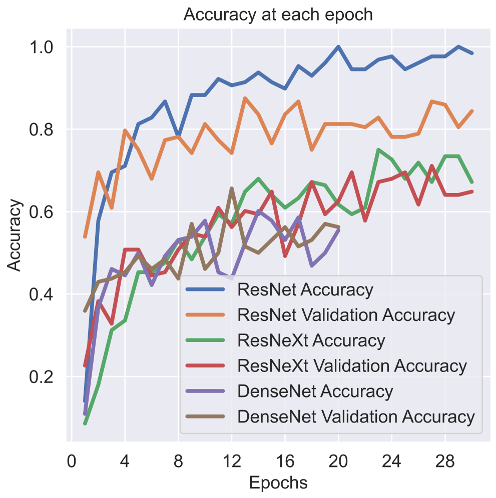


Fig. 22: Accuracy comparison of ResNet andResNeXt

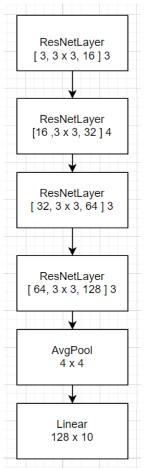

ResNetLayer [3, 3 x 3, 16] 3
ResNetLayer [16 ,3 x 3, 32] 4
ResNetLayer [32, 3 x 3, 64] 3
ResNetLayer [64, 3 x 3, 128] 3
AvgPool 4 x 4
Linear 128 x 10

Fig. 21: ResNet Architecture implementation

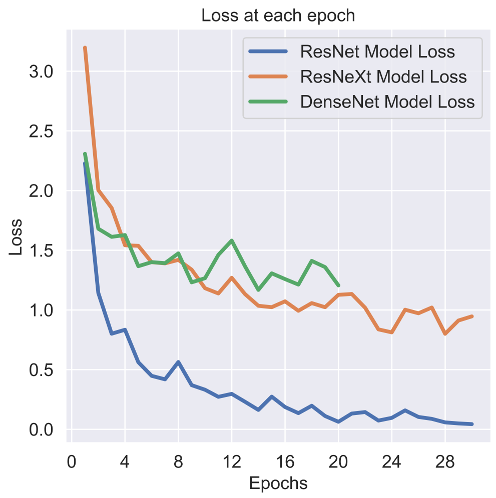
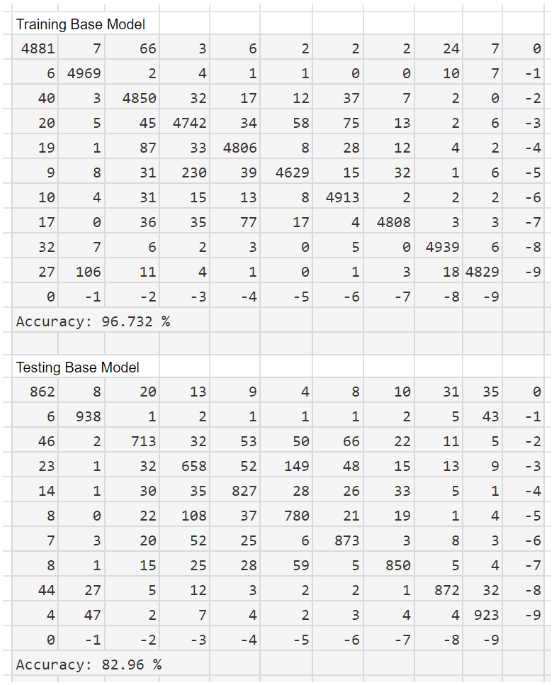


Fig. 23: Loss comparison of ResNet,ResNeXt



Training Base Model										
4881	7	66	3	6	2	2	2	24	7	0
6	4969	2	4	1	1	0	0	10	7	-1
40	3	4850	32	17	12	37	7	2	0	-2
20	5	45	4742	34	58	75	13	2	6	-3
19	1	87	33	4806	8	28	12	4	2	-4
9	8	31	230	39	4629	15	32	1	6	-5
10	4	31	15	13	8	4913	2	2	2	-6
17	0	36	35	77	17	4	4808	3	3	-7
32	7	6	2	3	0	5	0	4939	6	-8
27	106	11	4	1	0	1	3	18	4829	-9
0	-1	-2	-3	-4	-5	-6	-7	-8	-9	
Accuracy: 96.732 %										
Testing Base Model										
862	8	20	13	9	4	8	10	31	35	0
6	938	1	2	1	1	1	2	5	43	-1
46	2	713	32	53	50	66	22	11	5	-2
23	1	32	658	52	149	48	15	13	9	-3
14	1	30	35	827	28	26	33	5	1	-4
8	0	22	108	37	780	21	19	1	4	-5
7	3	20	52	25	6	873	3	8	3	-6
8	1	15	25	28	59	5	850	5	4	-7
44	27	5	12	3	2	2	1	872	32	-8
4	47	2	7	4	2	3	4	4	923	-9
0	-1	-2	-3	-4	-5	-6	-7	-8	-9	
Accuracy: 82.96 %										

Fig. 24: ResNet Confusion Matrix

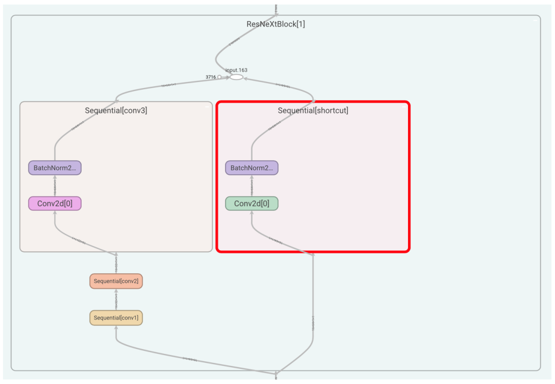


Fig. 25: ResNeXtBlock

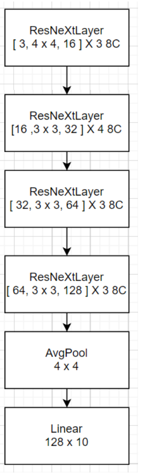
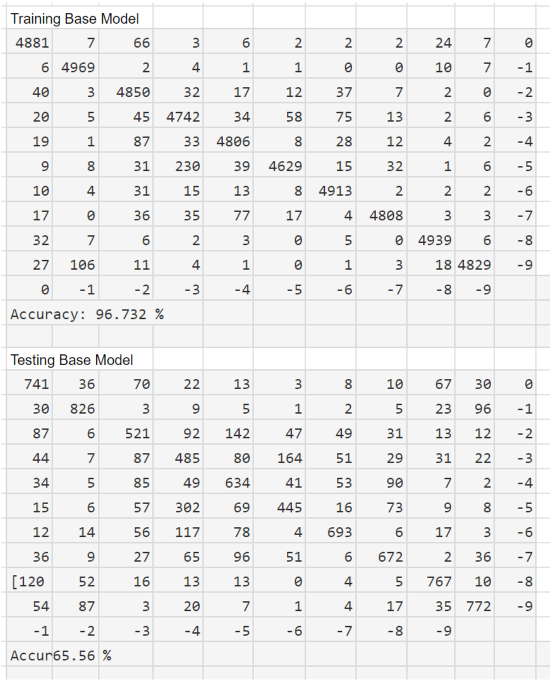
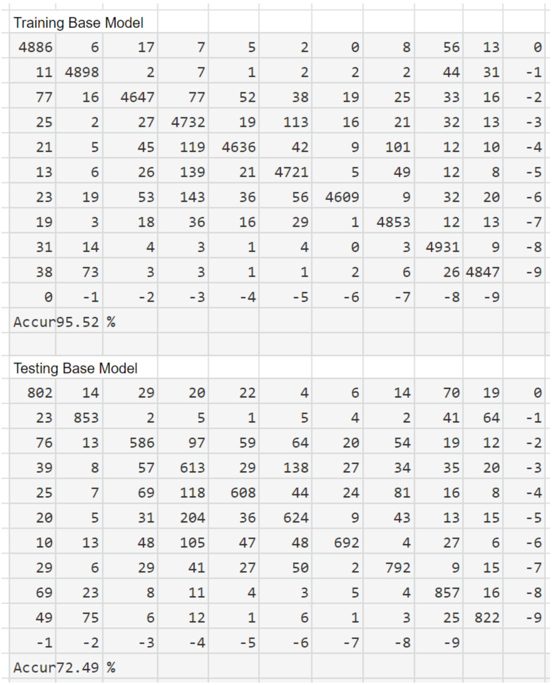


Fig. 26: ResNeXt Architecture implementation



Training Base Model										
4881	7	66	3	6	2	2	2	24	7	0
6	4969	2	4	1	1	0	0	10	7	-1
40	3	4850	32	17	12	37	7	2	0	-2
20	5	45	4742	34	58	75	13	2	6	-3
19	1	87	33	4806	8	28	12	4	2	-4
9	8	31	230	39	4629	15	32	1	6	-5
10	4	31	15	13	8	4913	2	2	2	-6
17	0	36	35	77	17	4	4808	3	3	-7
32	7	6	2	3	0	5	0	4939	6	-8
27	106	11	4	1	0	1	3	18	4829	-9
0	-1	-2	-3	-4	-5	-6	-7	-8	-9	
Accuracy: 96.732 %										
Testing Base Model										
741	36	70	22	13	3	8	10	67	30	0
30	826	3	9	5	1	2	5	23	96	-1
87	6	521	92	142	47	49	31	13	12	-2
44	7	87	485	80	164	51	29	31	22	-3
34	5	85	49	634	41	53	90	7	2	-4
15	6	57	302	69	445	16	73	9	8	-5
12	14	56	117	78	4	693	6	17	3	-6
36	9	27	65	96	51	6	672	2	36	-7
[120	52	16	13	13	0	4	5	767	10	-8
54	87	3	20	7	1	4	17	35	772	-9
-1	-2	-3	-4	-5	-6	-7	-8	-9		
Accur	65.56	%								

Fig. 27: ResNeXt Confusion Matrix



Training Base Model										
4886	6	17	7	5	2	0	8	56	13	0
11	4898	2	7	1	2	2	2	44	31	-1
77	16	4647	77	52	38	19	25	33	16	-2
25	2	27	4732	19	113	16	21	32	13	-3
21	5	45	119	4636	42	9	101	12	10	-4
13	6	26	139	21	4721	5	49	12	8	-5
23	19	53	143	36	56	4609	9	32	20	-6
19	3	18	36	16	29	1	4853	12	13	-7
31	14	4	3	1	4	0	3	4931	9	-8
38	73	3	3	1	1	2	6	26	4847	-9
0	-1	-2	-3	-4	-5	-6	-7	-8	-9	
Accur	95.52	%								
Testing Base Model										
802	14	29	20	22	4	6	14	70	19	0
23	853	2	5	1	5	4	2	41	64	-1
76	13	586	97	59	64	20	54	19	12	-2
39	8	57	613	29	138	27	34	35	20	-3
25	7	69	118	608	44	24	81	16	8	-4
20	5	31	204	36	624	9	43	13	15	-5
10	13	48	105	47	48	692	4	27	6	-6
29	6	29	41	27	50	2	792	9	15	-7
69	23	8	11	4	3	5	4	857	16	-8
49	75	6	12	1	6	1	3	25	822	-9
-1	-2	-3	-4	-5	-6	-7	-8	-9		
Accur	72.49	%								

Fig. 28: DenseNet Confusion Matrix

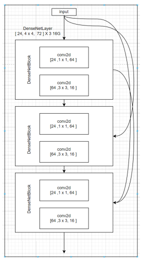


Fig. 29: DenseNet Block and Layer

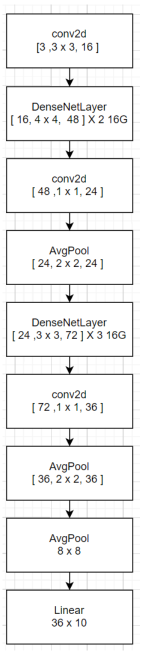


Fig. 30: DenseNet Block and Layer