Homework 0: Alohamora!

Kartik Madhira
Masters of Engineering in Robotics
University of Maryland, College Park
Email: kmadhira@terpmail.umd.edu
USING 3 LATE DAYS

***Abstract*—Boundary detection, Image segmentation and object classification in an image are fundamental problems in computer vision. In this paper we present two separate novel techniques that try to answer the boundary detection and object classification problem and discuss the importance of having these novel techniques.**

I. PHASE 1: SHAKE MY BOUNDARY

We perform boundary detection using a novel method called the 'Pb-Lite' boundary detection. This algorithm takes in an input image and using banks of filter creates a boundaries visually better than traditional approaches of Canny and Sobel boundary detection algorithms.

Filtering is essentially response to convolutions on each pixel with a variety of filters and is used to access the low level features in the image. This helps us to measure and aggregate the regional texture, brightness and color properties. Different scales and orientations of a particular filter are used so that various types of textures can be addressed. Here, we have used three filter banks namely: Oriented DoG filters, Leung-Malik Filters and Gabor Filters.

A. Oriented DoG Filters

This filter is created by convolving a simple Sobel Filter and a Gaussian kernel. The figure below shows the gaussian filter bank used for generating the results shown in the future sections. The filter bank is generated by using a single scale value and 15 orientations. The total number of filters is 15.

B. Leung-Malik Filters

The Leung-Malik filter bank is a collection 48 filters with multiple scales and orientations. It consists of first and second order derivatives of Gaussians, Laplacian of Gaussian(LOG) filters and 4 Gaussian filters. All these filters account for different types of features in the image. The filter bank is shown in the figure below:

C. Gabor Filters

A Gabor filter is generated by modulating a gaussian kernel with a sinusoidal plane wave. This is a linear filter that basically analyses if there is any specific frequency (governed by λ) content in the image in specific directions around the

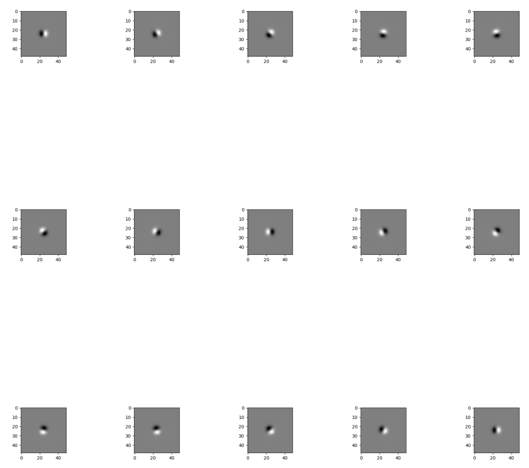


Figure 1: Oriented DoG filter bank for scale=2

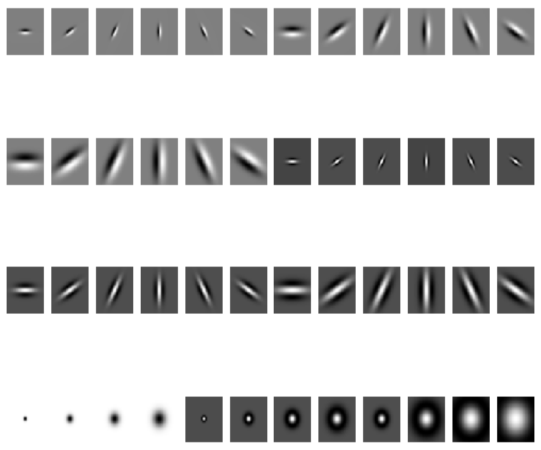


Figure 2: Leung Malik filter bank

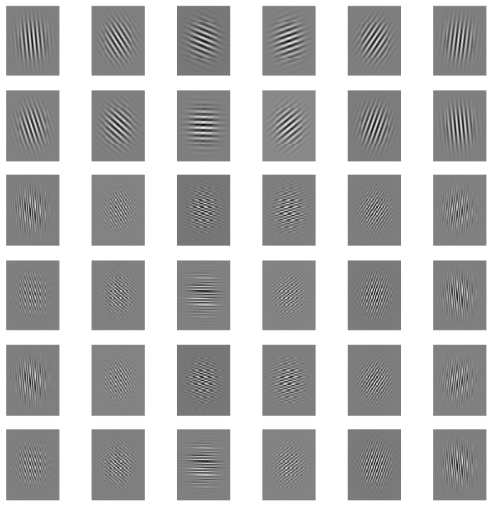


Figure 3: Gabor filter bank

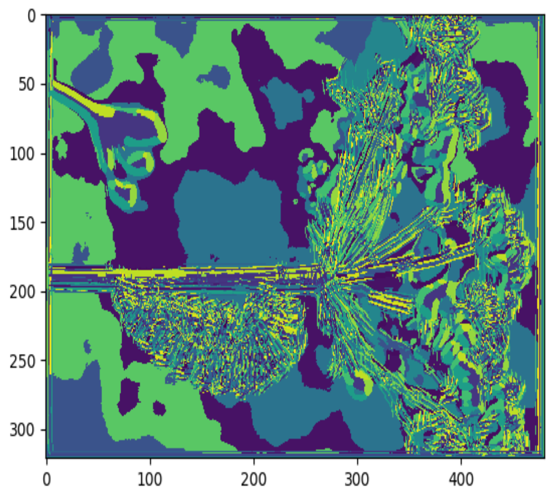


Figure 4: Texton map for image 2

point of interest. The Gabor filter bank used in this project is shown in the figure below: The filter bank with is generated for λ=1 with three scales: (9,16,25). Also, for each scale value, 12 filters with different orientations, uniformly spaced from 0 to 360 degrees are generated.

D. *Texton Map $\mathcal{T}$*

The first step in this process is to filter the image and find a Texton map which is essentially the texture map of the image. We stack the filter responses depth wise and then compute a K-means clustering into K number Texton ID clusters of the filter responses. We use K=64 as the number of clusters.

E. *Brightness Map $\mathcal{B}$*

The Brightness map captures the changes in intensity of light in the image. Similar to Texton map generation the K-Means clustering of the grayscale image is performed for K=16.

F. *Color Map $\mathcal{C}$*

The Color map captures the changes in color in the image. The color values were clustered using K-Means clustering into 16 clusters.

G. *Gradient Maps*

The Maps are used to again generate gradient maps T_g,B_g and C_g. These maps encode the texture, brightness and color distributions changing at each pixel. These are generated by comparing the values at each pixel by convolving the image with a left/right half-disc pair centered at the pixel.

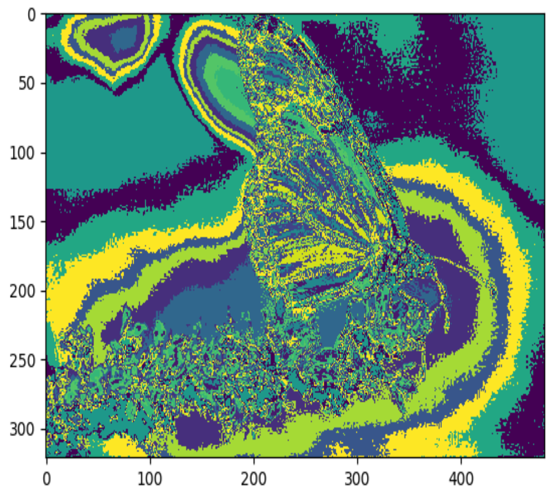


Figure 5: Brightness map for image 3

We generate half disks by first generating a full disk kernels and then applying a mask on top of the kernel generated. Next we rotate the mask for the different orientations we need for a particular scale. We masked this with the kernel using a logical OR.

Using the above generated Half-Disk masks we compute the T_g,B_g and C_g maps by comparing the distributions generated using each half-disk pair with a χ^2 measure. The

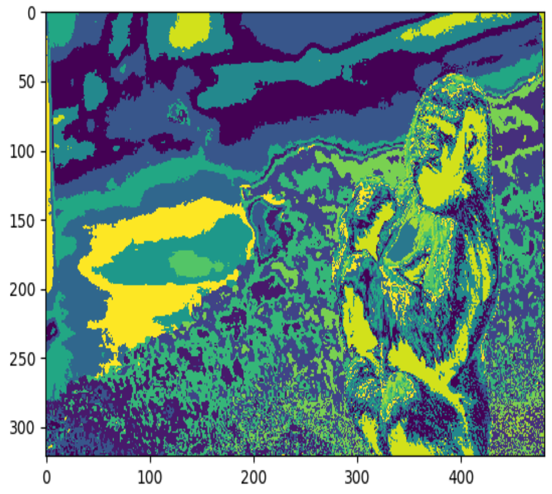


Figure 6: Color map for image 5

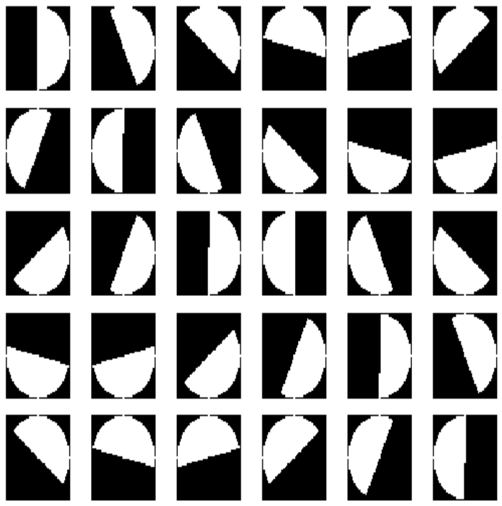


Figure 7: Half-Disk filters

binning scheme is defined for K indexes which is equal to the number of K-Means clusters for each T_g,B_g and C_g. This procedure is repeated for all the half-disk pairs to generate a 3D matrix of size $m \times n \times N$ where m, n are the dimensions of the image and N is the number of filters.

H. PB-Lite Output

Finally, the gradient maps generated are combined with Canny and Sobel baselines using the equation:

$$PbEdges = \frac{(T_g + B_g + C_g)}{3} \odot (w_1 * cannyPb + w_2 * sobelPb) \quad (1)$$

The $\odot$ is the Hadamard operator which is the element-wise multiplicatin of the arrays in the equation. The choice of the weights w_1 and w_2 is based on the canny and sobel baselines and the features we want from each baseline. The only constraint is that $w_1 + w_2$ = 1.

II. Phase 2: Deep Dive on Deep Learning

1) Initial neural network: The initial deep learning model used here is a simple neural net with 2 convolution layers followed by a fully connected network, followed by a softmax output. This architecture did not use any data augmentation while training and also no standardization technique was used. The network reached above 90% training accuracy pretty fast(less than 27 epochs as can be seen in the graphs). The important thing to note here is that even though the training accuracy is pretty high, the test accuracy reaches a maximum of 52.7% for the same number of epochs as the train set.

2) Improving accuracy: : In this network we try to improve the accuracy of the previous network by first standardizing the dataset within values [-1,1]. Next we also try to apply data augmentation techniques wherein we do random noise addition as well random left and right image rotation. To increase the complexity of the network we add more convolution layers followed by batch normalization. This is shown in Fig.8. We get considerable improvement in the test accuracy as the accuracy improves over the previous model with a maximum of 72.03%

3) ResNet: : We keep the standardization and data augmentation as it is and implement the ResNet architecture. A Residual Network, or ResNet is a neural network architecture which solves the problem of vanishing gradients in the simplest way possible that is by applying skip connections in a general residual block as shown in fig 11. This allows the network to accommodate deep layers without having the vanishing gradient problem. This network has a slight improvement from the previous network with an accuracy of 73.98% in the test set. We use 25 deep layers in the ResNet architecture. Without the skip connections, a 'plain' architecture with 25 deep layers would not allow us to have lower losses as epochs increase.

4) ResNeXt: :
ResNext architecture is an improvement over the ResNet architecture. This network utilizes the concept of parallelising layers and concatenating them from Google's Inception model as well as the skip connection used in the ResNet. This network utilizes the use of cardinality which is the size

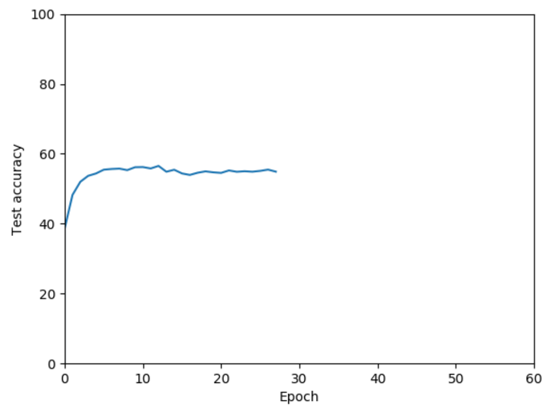


Figure 8: Training and Loss vs Epochs without for network in section I

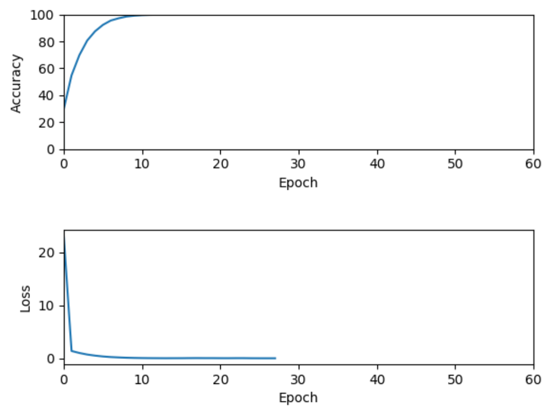


Figure 9: Test accuracy vs Epochs for network in section I

of the set of transformations . In ResNext architecture, the output, a[l], of layer, 'l' much deeper into the network before applying the non-linearity (Relu). Such multiple blocks are added in parallel and the number of parallel units is equal to cardinality. ResNeXt implemented in this homework has two different parallelly connected blocks with cardinality 8 . Both the blocks have three filters and have a convolution layer with 128 filters in between to connect them. We get a maximum accuracy of 68.7% by training it till 60 epochs.

5) DenseNet: Previous networks have shown that that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In DenseNet each layer connects to every other layer in a feed-forward fashion. The idea here is that if connecting a skip connection from the previous layer improves performance, we just keep skip connections between all layers. That way there is always a direct route for the information backwards through the network.

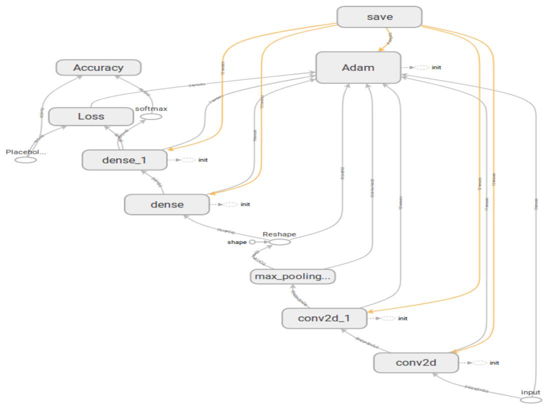


Figure 10: Model of the network in section 2

Parameter	Value
Batch Size	512
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	27
Max test Accuracy(last epoch)	52.7%
No. of Parameters	410090
Inference time(s)	0.0013

Table I: Parameters for the deep network used in section I

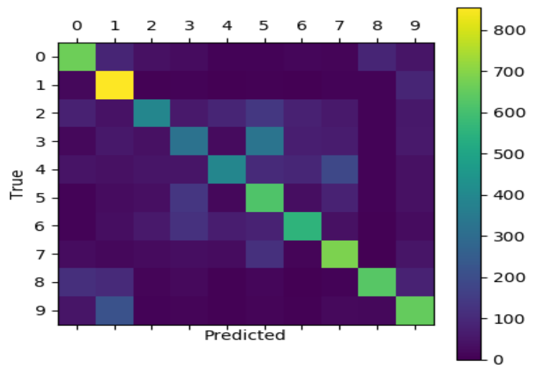


Figure 11: Confusion Matrix for network in I

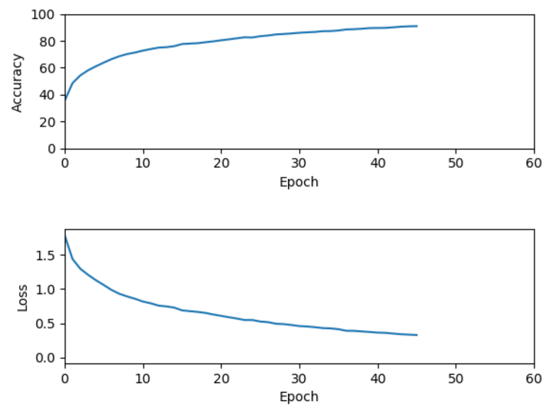


Figure 12: Training and Loss vs Epochs without for improved network

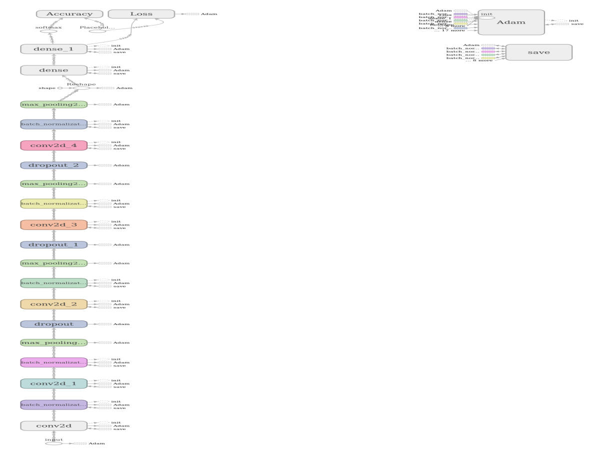


Figure 14: Model of the improved network

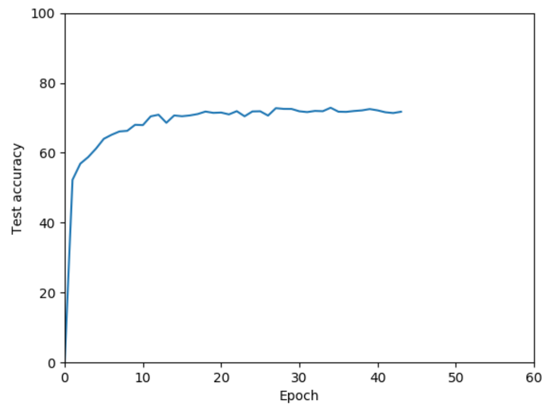


Figure 13: Test accuracy vs Epochs for improved network in section 2

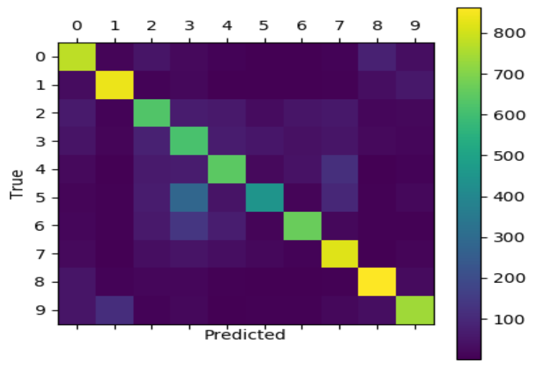


Figure 15: Confusion Matrix for network in II

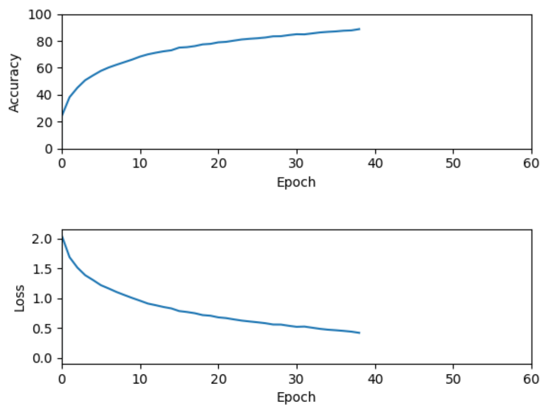


Figure 16: Training and Loss vs Epochs for ResNet

Parameter	**Value**
Batch Size	512
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	45
Max test Accuracy(last epoch)	72.03%
No. of Parameters	473386
Inference time(s)	0.0023

Table II: Parameters for the deep network used in section II

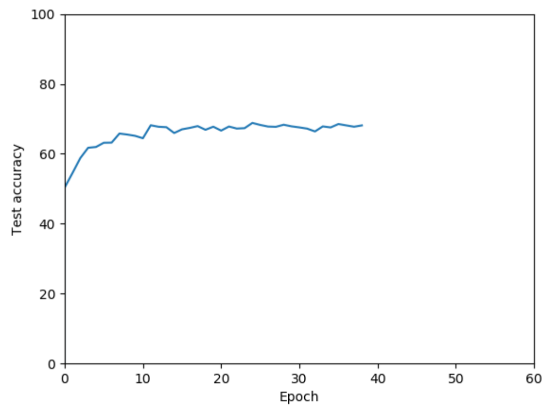


Figure 17: Test accuracy vs Epochs for ResNet

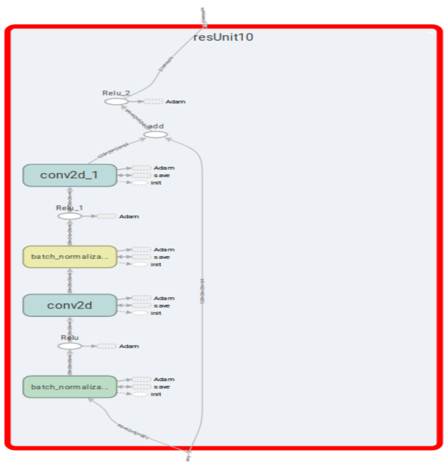


Figure 18: Skip connection in ResNet

Parameter	Value
Batch Size	512
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	38
Max test Accuracy(last epoch)	73.98%
No. of Parameters	1134670
Inference time(s)	0.0025

Table III: Parameters for ResNet

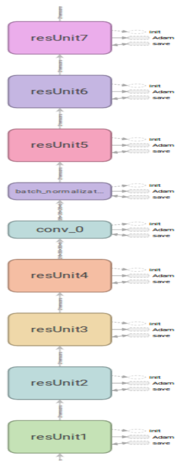


Figure 19: First 7 of the 25 layers deep residual network

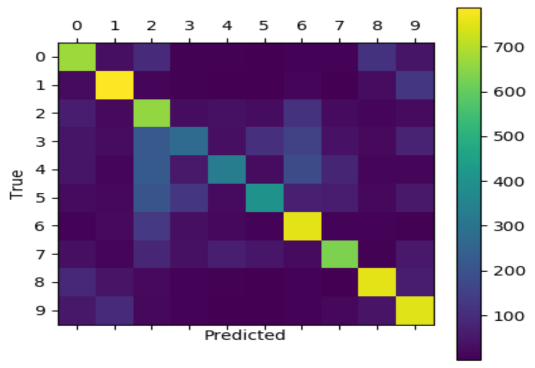


Figure 20: Confusion Matrix for ResNet

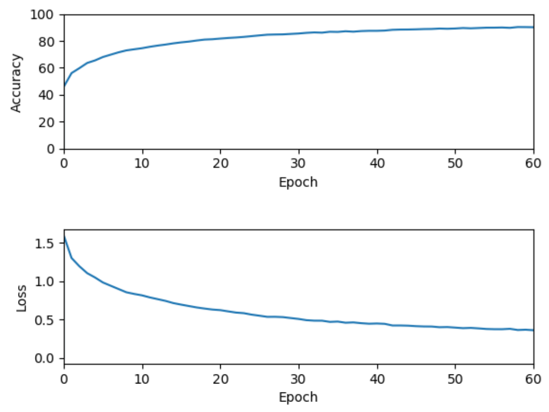


Figure 21: Training and Loss vs Epochs for ResNeXt

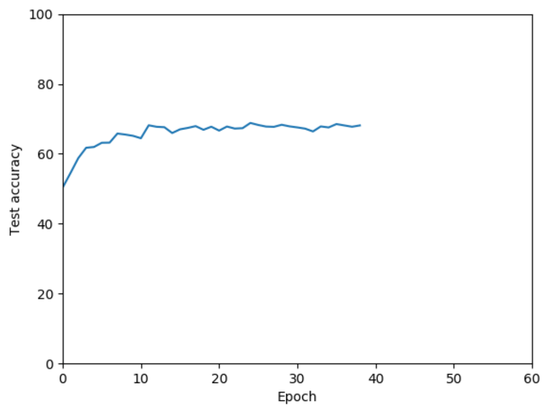


Figure 22: Test accuracy vs Epochs for ResNeXt

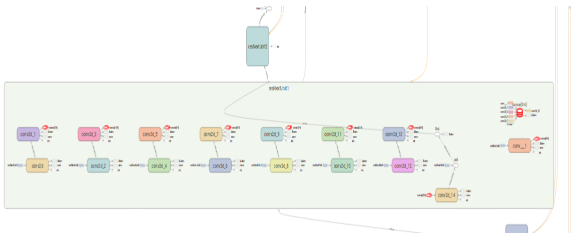


Figure 23: A ResNeXt block with cardinality=8

Parameter	Value
Batch Size	512
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	64
Max test Accuracy(last epoch)	68.7%
No. of Parameters	1304670
Inference time(s)	0.0033

Table IV: Parameters for the ResNeXt

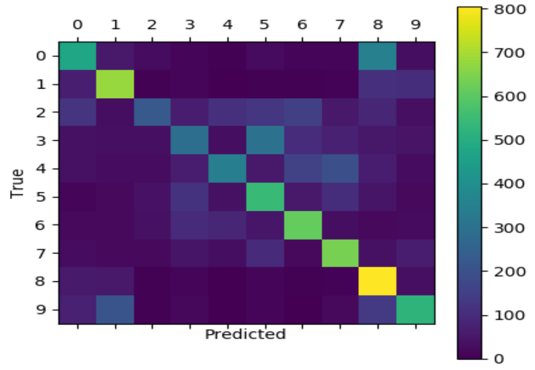


Figure 24: Confusion Matrix for ResNeXt

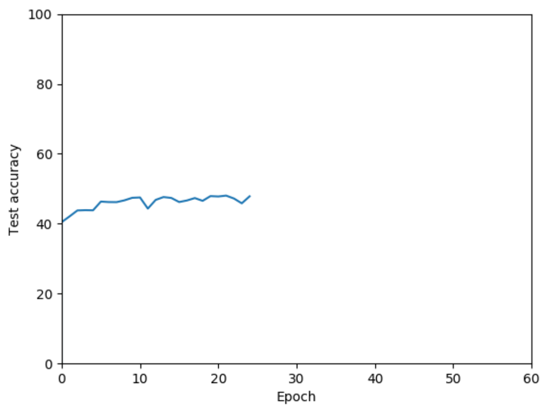


Figure 25: Training and Loss vs Epochs for Densenet

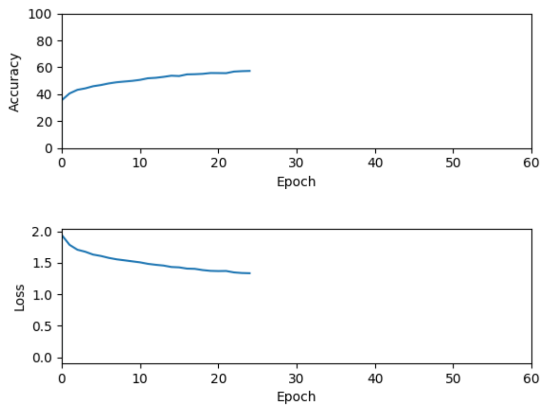


Figure 26: Test accuracy vs Epochs for Densenet

III. Conclusion

The conclusion goes here.

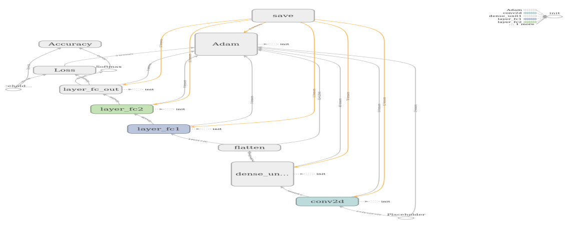


Figure 27: Densenet architecture

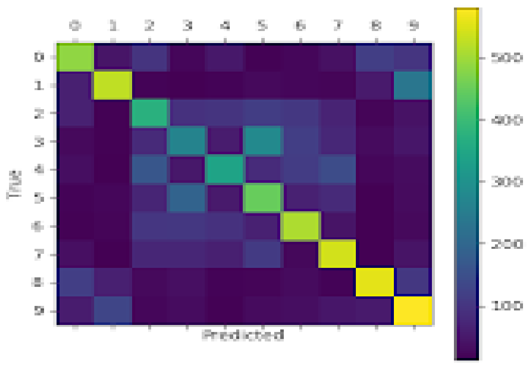


Figure 28: Confusion Matrix for Densenet

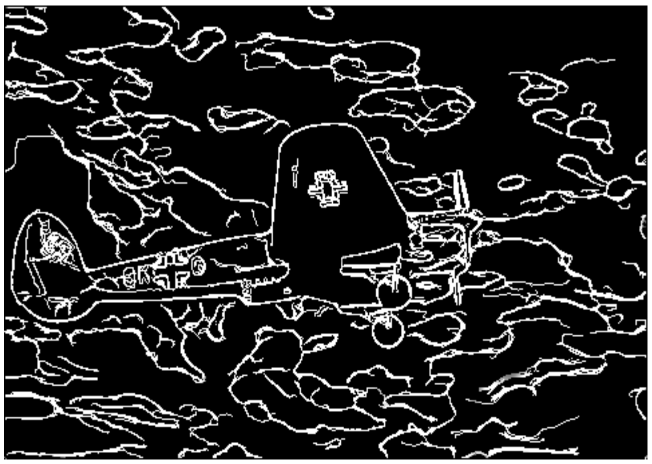


Figure 31: Pb-Lite output for image 1

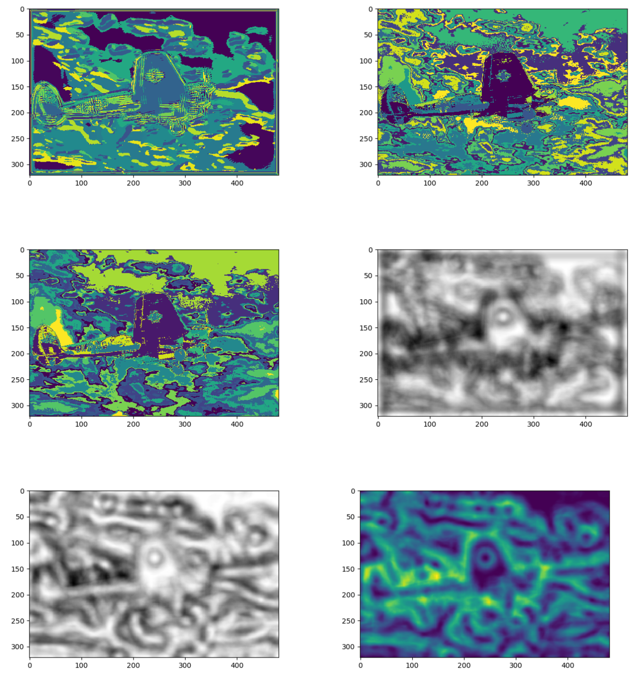
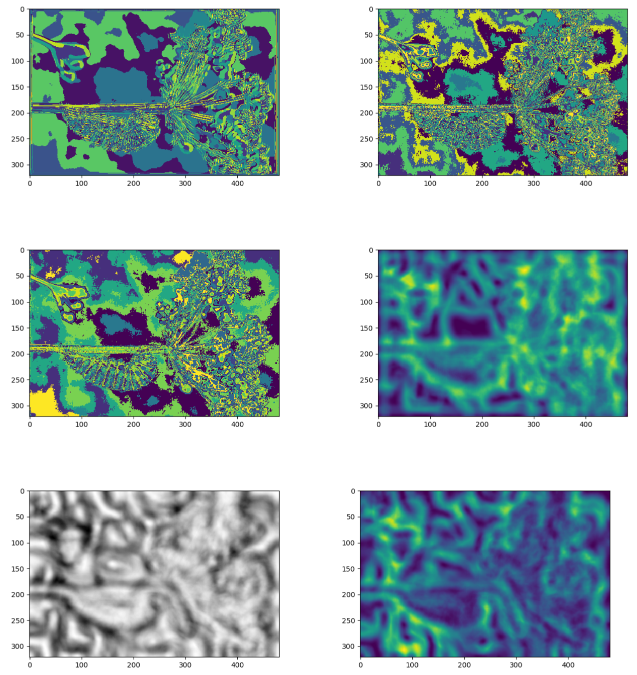


Figure 29: Clockwise (a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 1



(a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 2

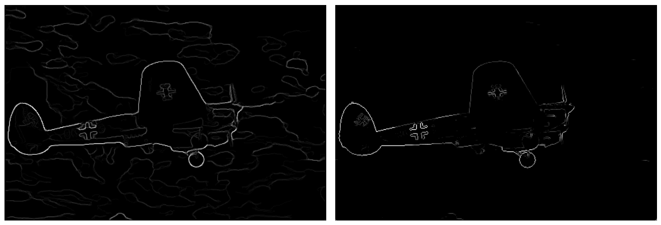


Figure 30: (f) Canny baseline and Sobel baseline for image 1

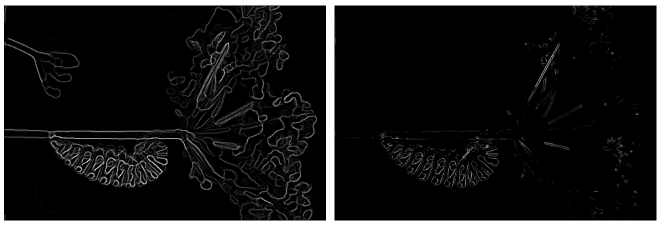


Figure 32: (f) Canny baseline and Sobel baseline for image 2

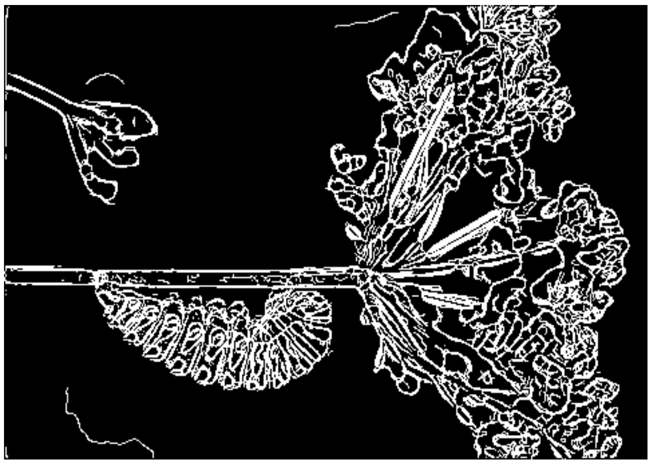


Figure 33: Pb-Lite output for image 2

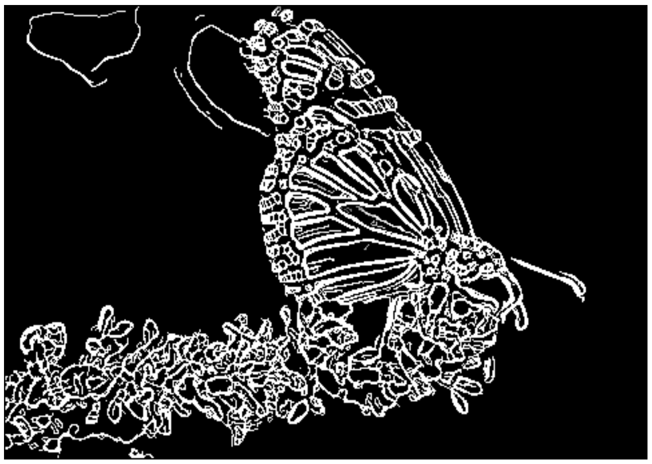


Figure 36: Pb-Lite output for image 3

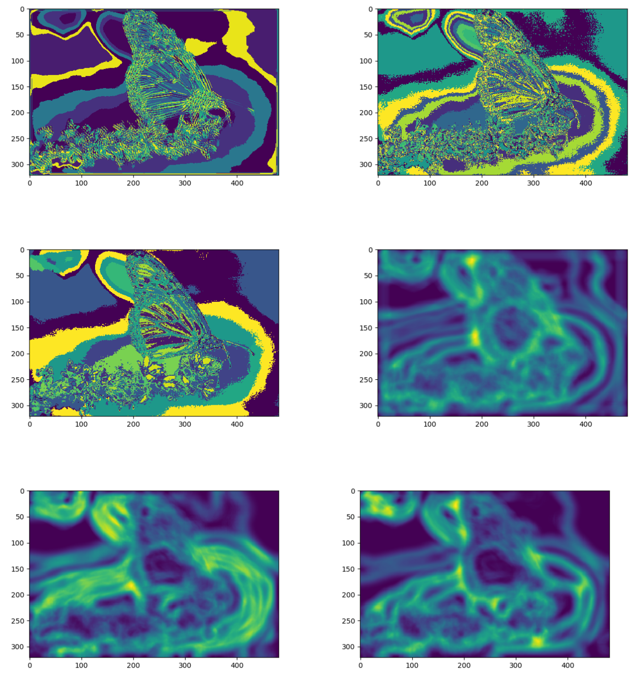


Figure 34: (a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 3

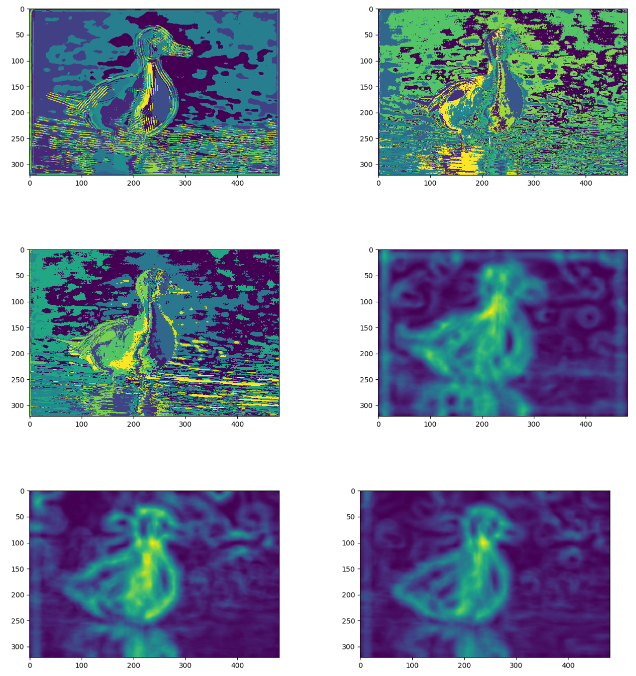


Figure 37: (a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 4

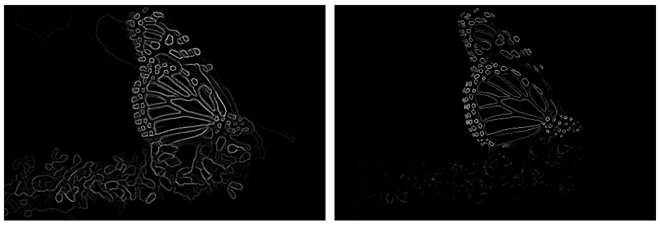


Figure 35: (f) Canny baseline and Sobel baseline for image 3

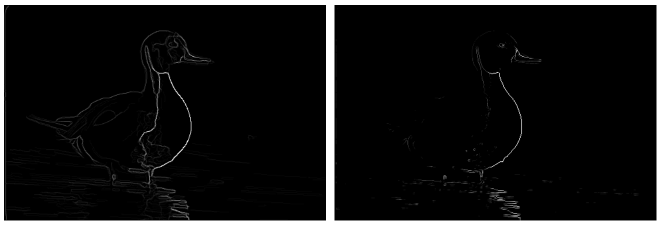


Figure 38: (f) Canny baseline and Sobel baseline for image 4

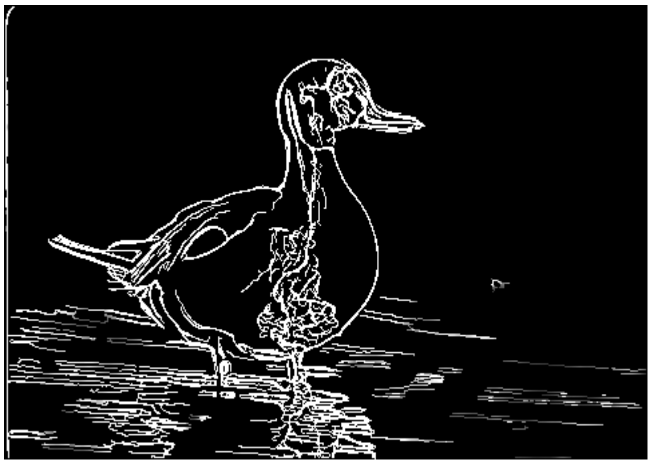


Figure 39: Pb-Lite output for image 4

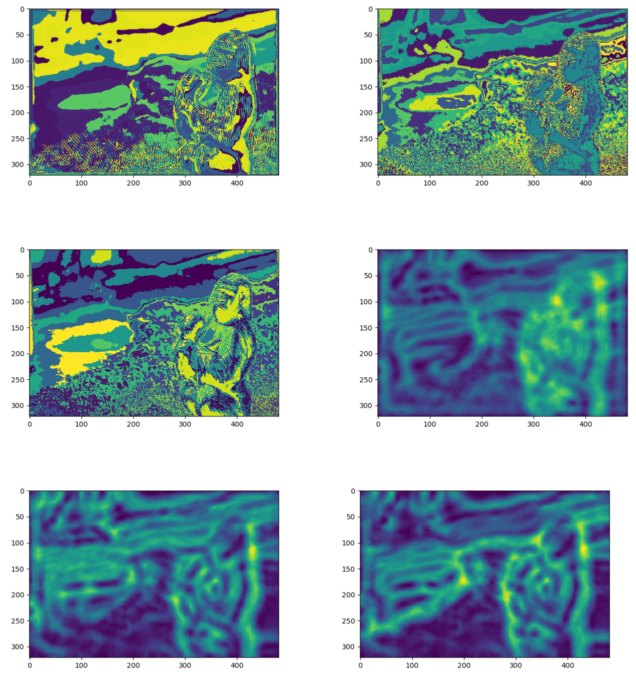


Figure 40: (a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 5

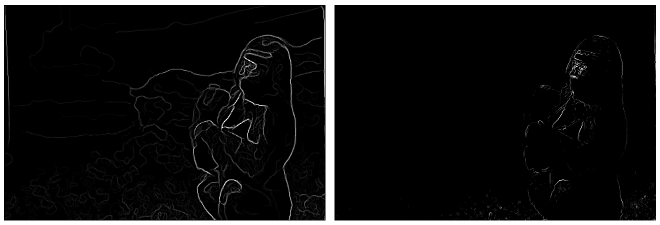


Figure 41: (f) Canny baseline and Sobel baseline for image 5

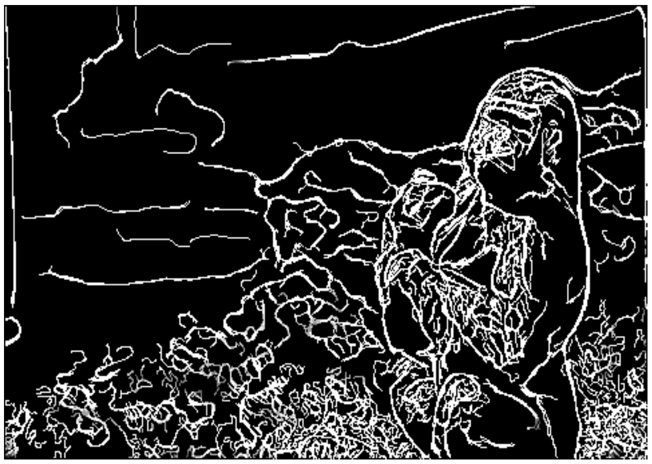


Figure 42: Pb-Lite output for image 5

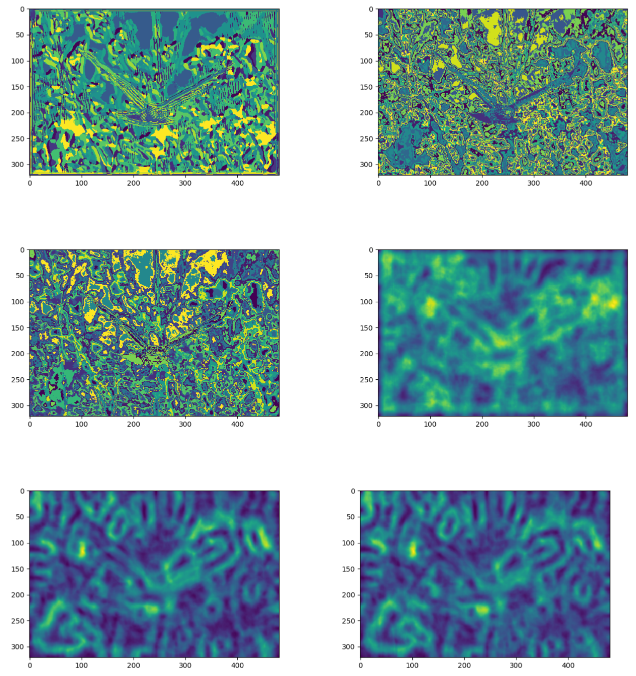


Figure 43: (a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 6

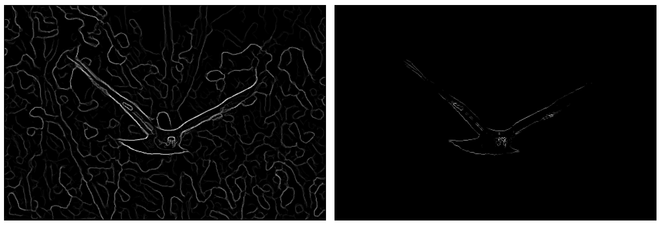


Figure 44: (f) Canny baseline and Sobel baseline for image 6

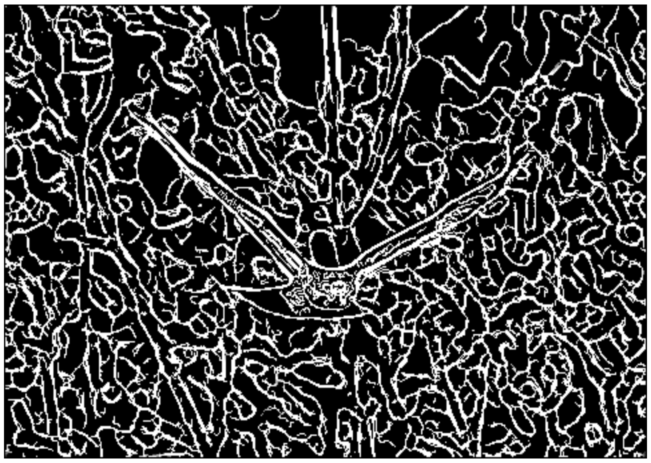


Figure 45: Pb-Lite output for image 6

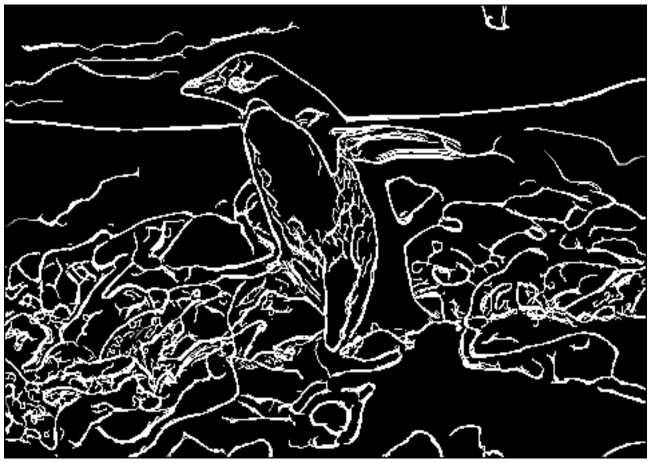


Figure 48: Pb-Lite output for image 7

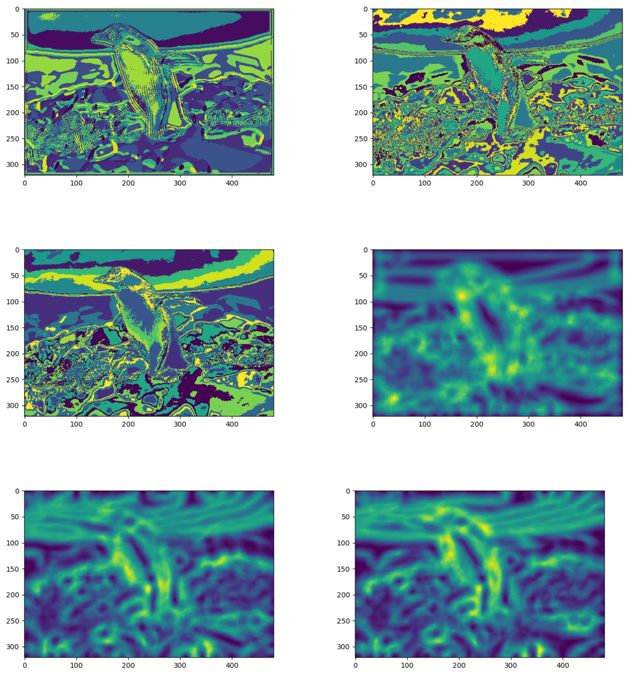


Figure 46: (a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 7

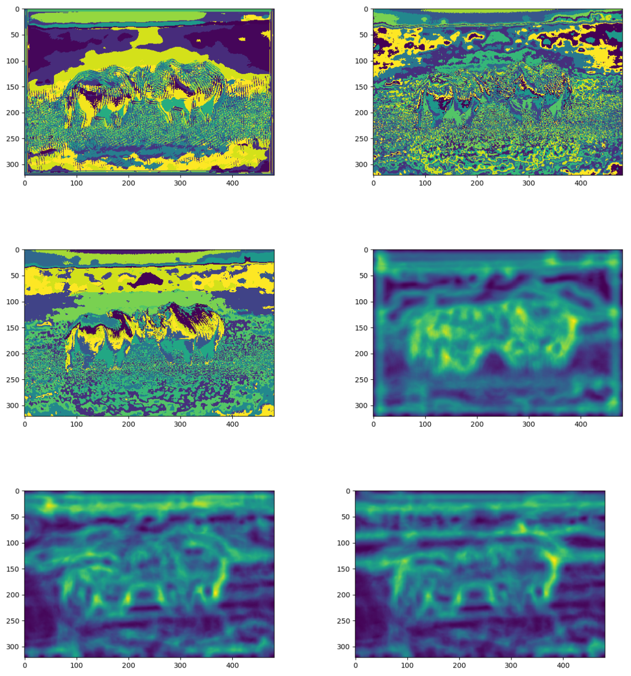


Figure 49: (a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 8

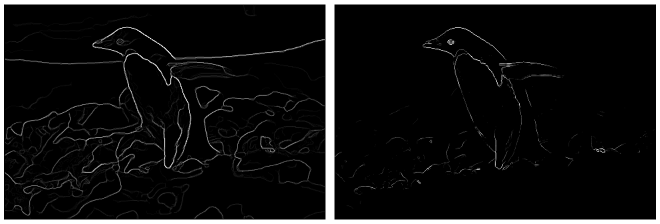


Figure 47: (f) Canny baseline and Sobel baseline for image 7

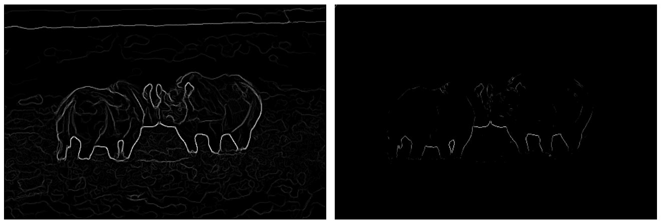


Figure 50: (f) Canny baseline and Sobel baseline for image 8

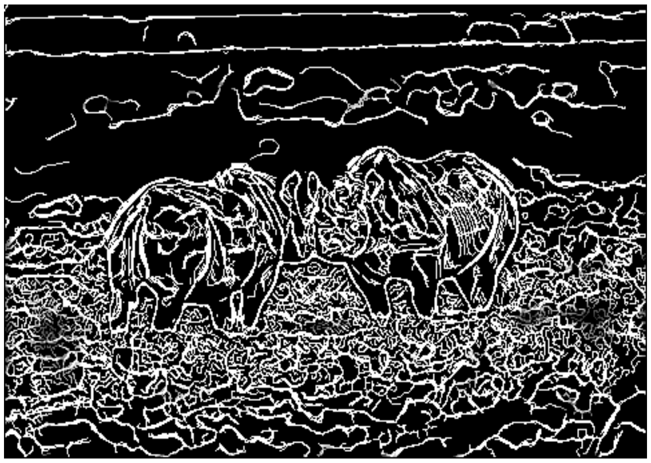


Figure 51: Pb-Lite output for image 8

Parameter	Value
Batch Size	512
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	25
Max test Accuracy(last epoch)	45.3%
No. of Parameters	1738714
Inference time(s)	0.0046

Table V: Parameters for the Densenet

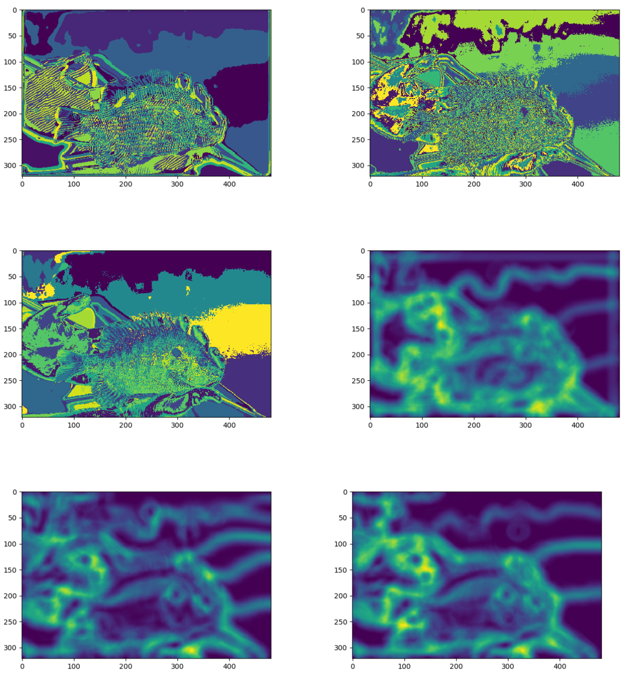


Figure 52: (a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 9

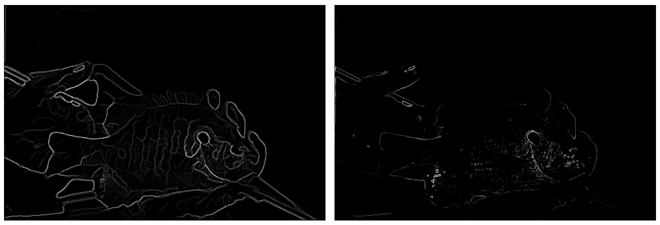


Figure 53: (f) Canny baseline and Sobel baseline for image 9

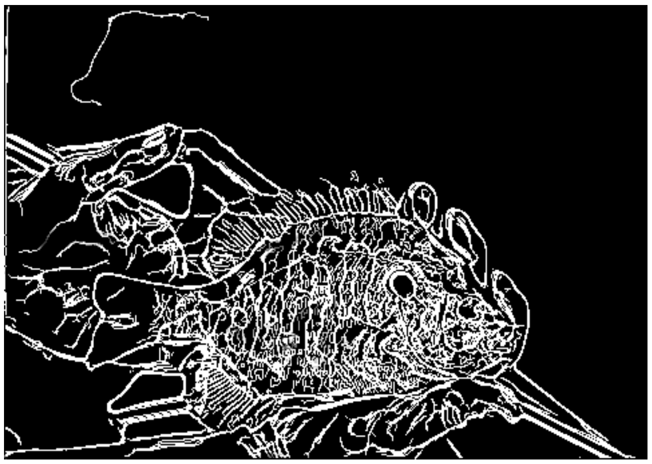


Figure 54: Pb-Lite output for image 9

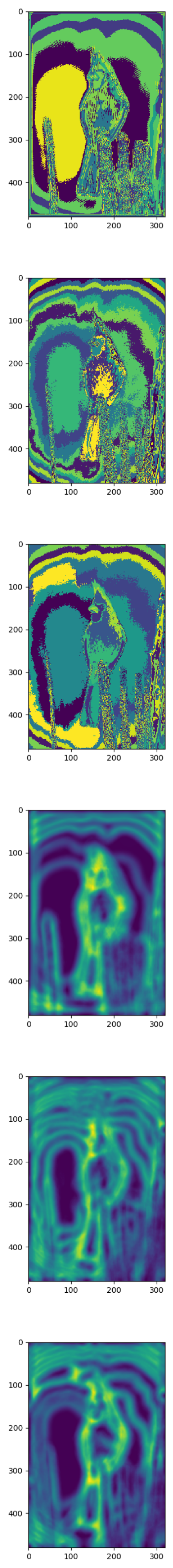


Figure 55: (a)Texton Map, (b)Brightness Map, (c)Color Map, (d)Texton gradient, (e)Brightness gradient and (e) Color gradient for image 10

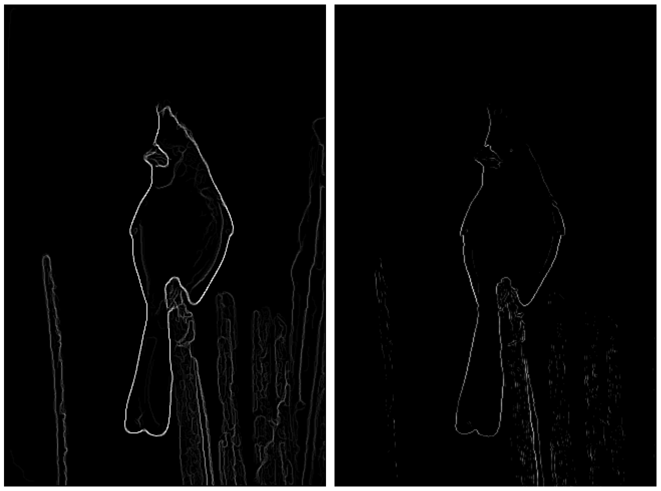


Figure 56: (f) Canny baseline and Sobel baseline for image 10

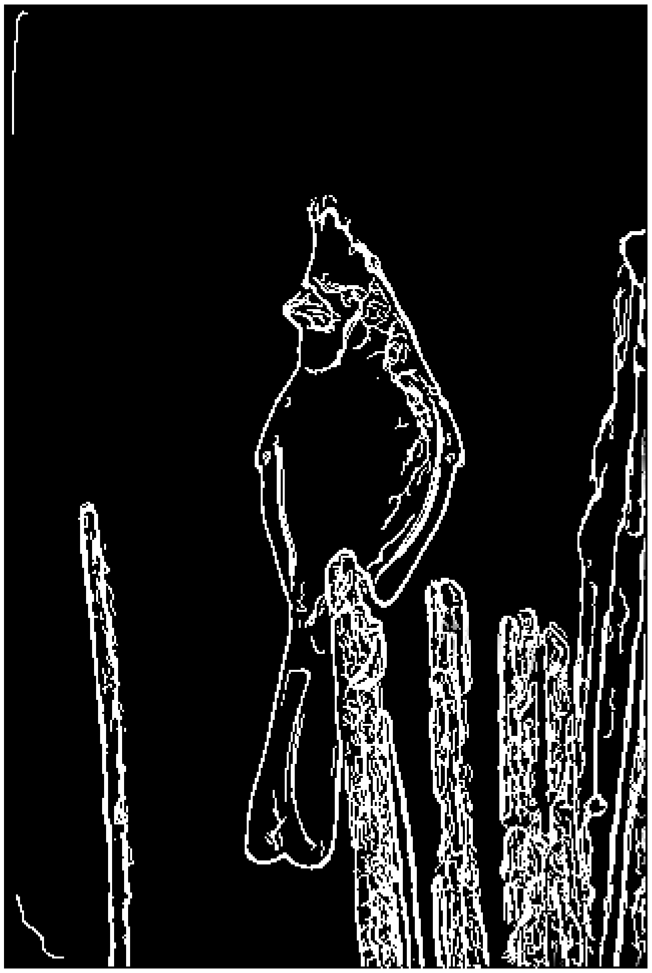


Figure 57: Pb-Lite output for image 10

References

[1] ResNets, HighwayNets, and DenseNets, Oh My! - A guide to Deep Nets