Homework 0 - Alohomora!

Prateek Arora
Robotics Graduate Student
University of Maryland
Email: pratique@terpmail.umd.edu
USING 3 LATE DAYS

***Abstract*—The purpose of the homework is to learn Classical and Deep Learning approaches in computer vision. The homework is divided into two phases, namely Phase 1 and Phase 2. Phase 1 focuses on Boundary detection using traditional techniques while Phase 2 is about implementing multiple neural and comparing them on various criterion like number of parameters, train and test set accuracy.**

I. Introduction

Edge detection is a classic problem in computer vision and a fundamental tool for many algorithms. The goal of such algorithms is to find all relevant discontinuities in an image. Because what constitutes a relevant discontinuity can vary highly among even humans, edge detection is considered to be a difficult problem.

The most naive method of solving this problem (Sobel) involves generating an intensity (i.e. grayscale) map from the original image, and computing differences in intensity across neighboring pixels. By cutting these values off at a predefined constant, this algorithm finds all edges sharper than that value. A slightly smarter version (Canny) attempts to decide which pixels comprise edges by checking whether the gradient at that pixel is similar to gradients at neighboring pixels. However, these two algorithms only consider changes in intensity, and fail to consider changes in texture.

In this project, we will attempt to beat these naive algorithms using pb-lite. In addition to considering intensity, we will also consider changes in texture and brightness in the local neighborhood. Using all of this information, we will assign to each pixel a probability that it lies on an edge.

II. PHASE 1: SHAKE MY BOUNDARY

The task in this phase is to perform boundary detection using an algorithm called "PB-Lite". The goal of Phase 1, as stated before, is to find all relevant discontinuities in an image. Because what constitutes a relevant discontinuity can vary highly among even humans, edge detection is considered to be a difficult problem. The block diagram of the algorithm in the figure 1. 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. This is computed by clustering the filter responses with K-Means clustering.

Filtering 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.

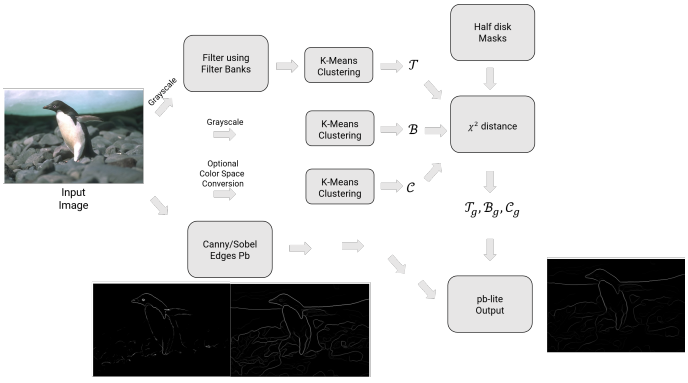

Figure 1. Block diagram of PB-Lite Algorithm

A. Oriented DoG Filters

The Oriented Difference of Gaussian Filter is generated by taking the difference of two normal Gaussian Filters with same variance but the centers are of each is shifted by an amount equal to the standard deviation. This filter can also be created by convolving a simple Sobel Filter and a Gaussian kernel. The figure 2 shows the gaussian filter bank used for generating the results shown in the future sections.

The filter bank is generated by using 5 different scale values and 15 orientations for each scale, linearly-spaced from $0\,\mathrm{deg}$ to $360\,\mathrm{deg}$. Hence, the total number of filters is $5 * 16 = 80$.

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 (3)

C. Gabor Filters

Gabor filters are inspired based on the way human visual system. A Gabor filter is generated by modulating a gaussian

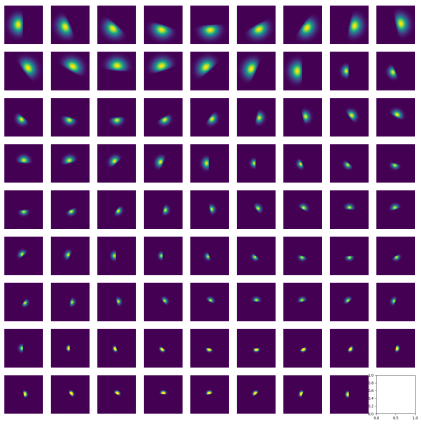

Figure 2. DoG Filter Bank

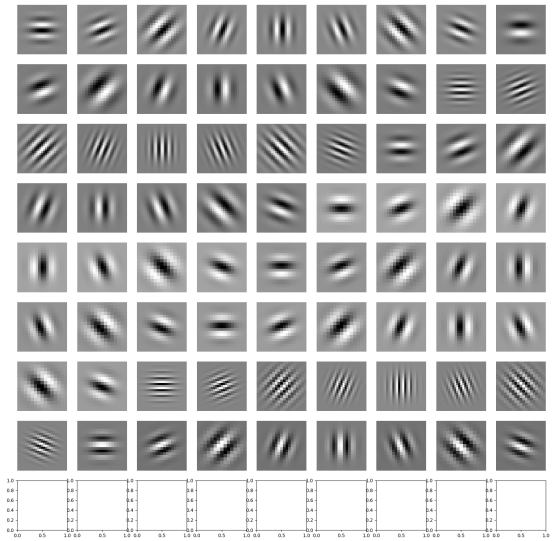

Figure 4. Gabor filter bank

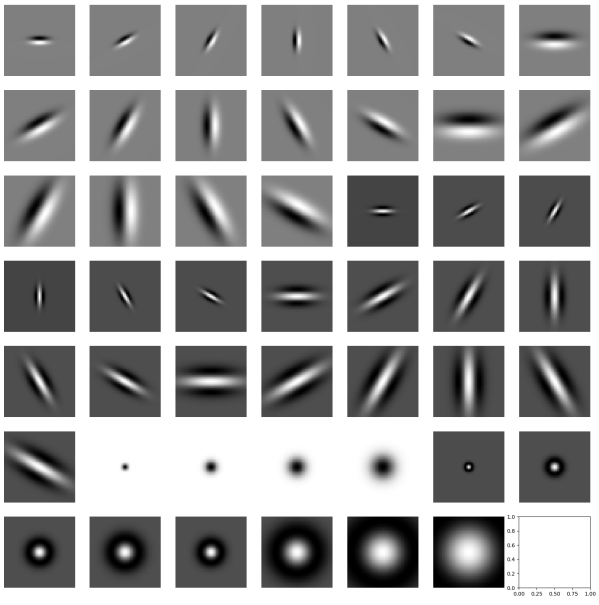

Figure 3. LM filter bank

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 point of interest. The Gabor filterbank used in this project is shown in the figure(4) The filter bank with is generated for λ=1 with three scales:[9,16,25]. Also, for each scale value, 15 filters with different orientations, uniformly spaced from 0 to 360 degrees are generated.

D. Texton Map $\mathcal{T}$

Once the filtering process of the image is complete, we end up with a stack of images of size $m \times n \times N$, where m, n are the dimensions of the image and N is the total number of filters used. Thus, each pixel value can now be represented as a distribution of these N values. Each distribution is then represented by a unique Texton-ID. These different distributions for all the pixels are then clustered into K textons using K-Means. This generates an image which captures the texture changes in the original image. Texton Maps for all the test set images be seen in the following figures:

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 and the output can be seen in the Phase-1 result section

F. Color Map $\mathcal{C}$

The Color map captures the changes in color/ chrominance in the image. The color values were clustered using K-Means clustering into 16 clusters. This generates an output image which can be seen Phase-1 result section

G. Gradient Maps

The Maps generated above are used to calculate gradient maps $\mathcal{T}_g$,$\mathcal{B}_g$ and $\mathcal{C}_g$. 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.

The basic concept behind this is that if the values are similar the gradient should be small and if the values are dissimilar, the gradient will be large.

The half-disks are generated by multiplying an array of size equal to the radius/scale of the circular disk with all values which lie inside this radius equal to 1 and rest 0, with an array of equal size but where one half of the array is 0s and the other half consists of 1s. This multiplication results in a half-disk which can be rotated to produce the desired half-disk mask.

Here if you rotate the disk after you've multiplied the two arrays will result in pixel voids. This can be avoided by rotating the rectangular block matrix of 0s and 1s and then by applying a "logical OR" operator on them. This gives the required half-disk masks which are shown Phase-1 result section

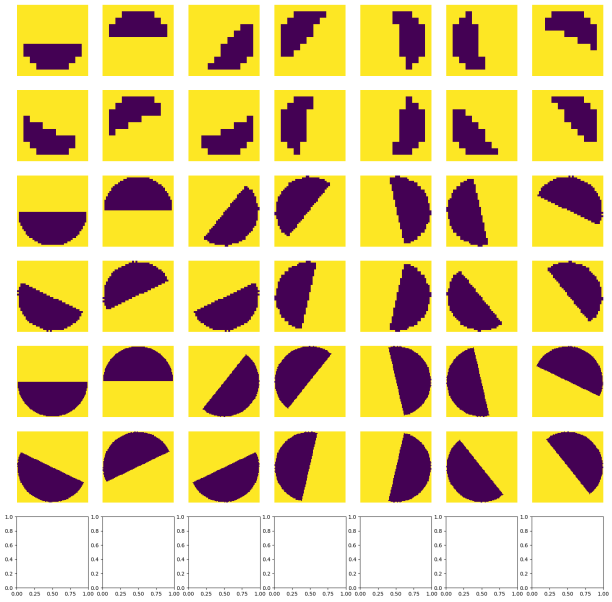

Figure 5. Half-Disk Masks for scales=[5,20,50]

Using the above generated Half-Disk masks we compute the $\mathcal{T}_g, \mathcal{B}_g$ and $\mathcal{C}_g$ maps by comparing the distributions generated using each half-disk pair with a χ^2 measure. The binning scheme is defined for $\mathcal{K}$ indexes which is equal to the number of K-Means clusters for each $\mathcal{T}_g$, $\mathcal{B}_g$,$\mathcal{C}_g$. 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.

The output of the mean of each $\mathcal{T}_g, \mathcal{B}_g$ and $\mathcal{C}_g$ is given as follows:

H. PB-Lite Output

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

$$PbEdges = \frac{(\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g)}{3} \odot (w_1 * cannyPb + w_2 * sobelPb) \quad (1)$$

The $\odot$ is the Hadamard operator which is the element-wise multiplication 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 $w_1 + w_2 = 1$. The Canny and Sobel outputs and the resulting Pb-lite outputs are shown in the figures**??** in the Phase-1 result section:

I. PHASE 2: Deep Dive on Deep Learning

1) Simple Convolution Neural Network: A Simple Convolution Neural Network comprises of convolution layers and fully connected layers. There are very deep layered architecture implemented for classification on CIFAR10 dataset but the architecture implemented in this section comprises of 4 convolution with different numbers of filters followed by two fully connected layers. While training this model no data augmentation or standardization was performed. On training the network 96.3% training accuracy is achieved in 25 epoch. Although the training accuracy is high, the test accuracy reached 64.23% only.
Inference:The reason for high training accuracy and low test accuracy is probably that the network is over-fitting the data and thus fails to classify previously unseen images. To get high test accuracy data augmentation and standardization needs to be performed which is explained in the next section.

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.

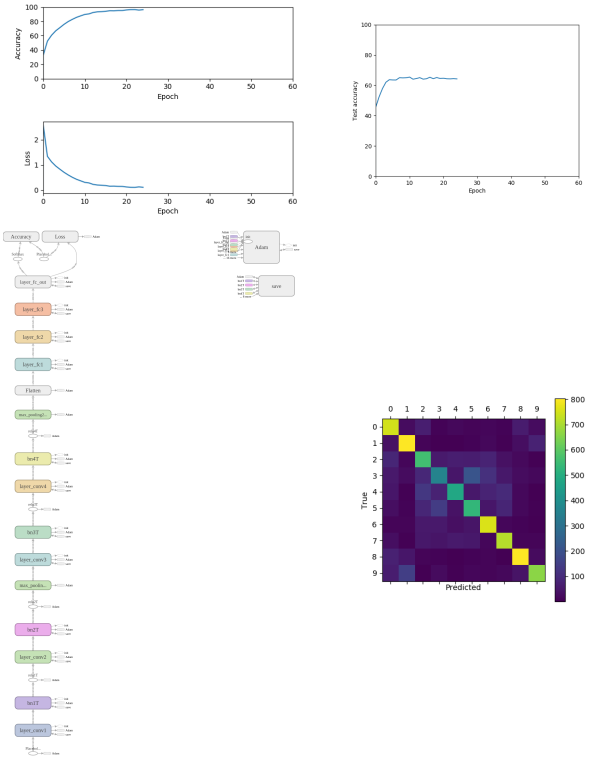

Figure 6. Serially from top left image: (a) Training set accuracy and loss, (b)Test set accuracy,(c) Network Architecture, (d) Confusion matrix for Simple CNN without data augmentation and standardization

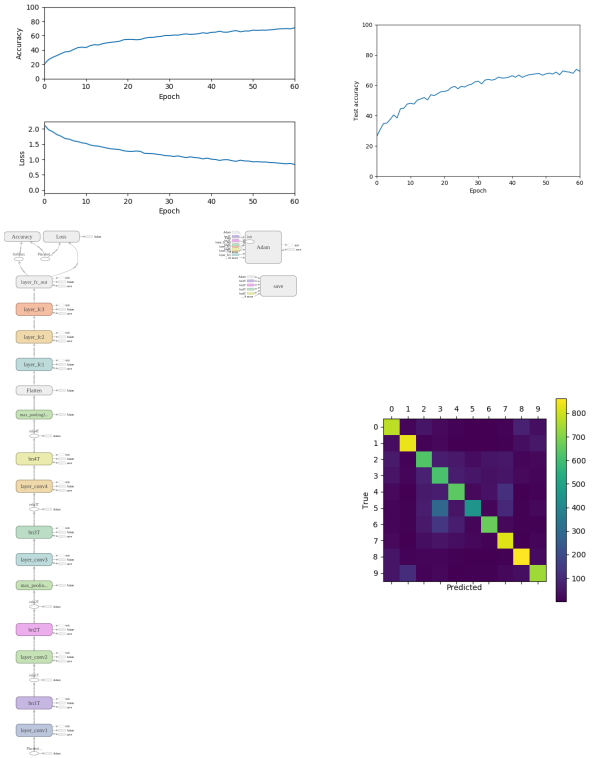

Figure 7. Serially from top left image: (a) Training set accuracy and loss, (b)Test set accuracy,(c) Network Architecture, (d) Confusion matrix for Simple CNN with data augementation

J. ResNeXt

ResNext architecture is an improvement over the ResNet architecture. In addition to utilizing the concept of residual learning framework from ResNet, the concept of cardinality is introduced in this network. Cardinality is the size of the set of transformations (as shown in II-J). 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. The Output of the parallel units is then summed up followed by application of non-linearity. It is empirically shown in the paper that even under the restricted condition of maintaining complexity, increasing cardinality is able to improve classification accuracy. Moreover, increasing cardinality is more effective than going deeper or wider when we increase the capacity. ResNeXt implemented in this homework has two different parallely connected blocks with cardinality 8 and 4 respectively. Both the blocks have three filters and have a convolution layer with 128 filters in between to connect them.

K. 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. DenseNets have several advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. In my implementation, an image is passed through a single convolution layer followed by a DenseNet block comprising of 5 convolution layers.

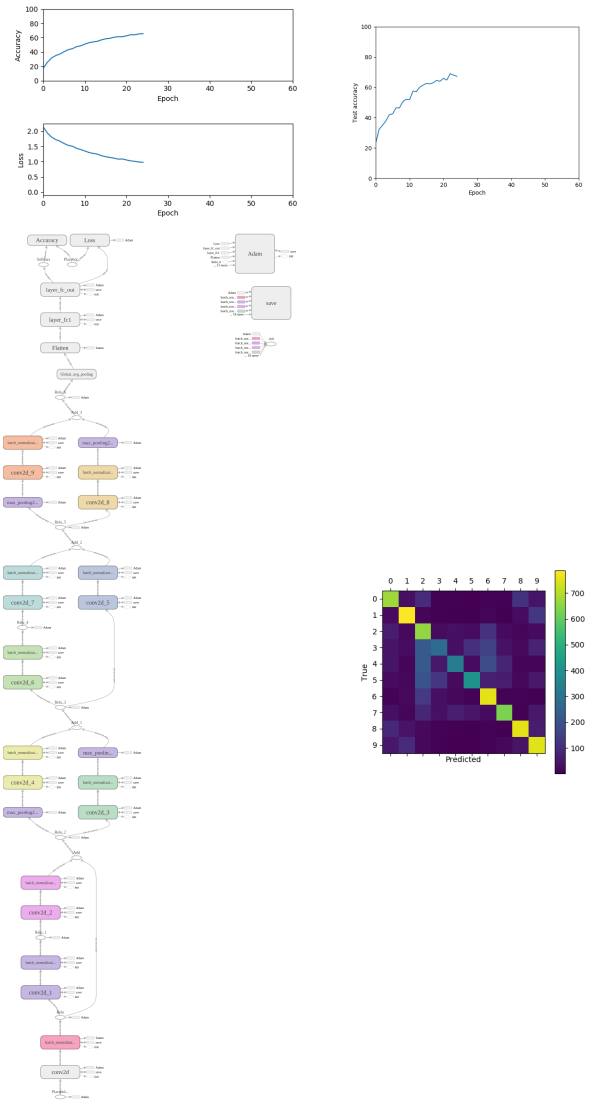

Figure 8. Serially from top left image: (a) Training set accuracy and loss, (b)Test set accuracy,(c) Network Architecture, (d) Confusion matrix for ResNet architecture

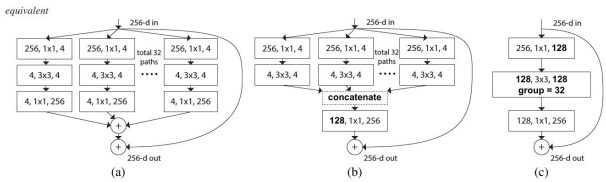

Figure 9. ResNeXt block diagram

III. Phase-2 results

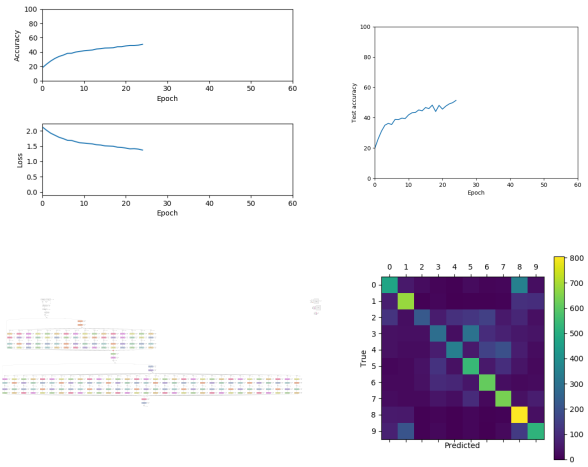

Figure 10. Serially from top left image: (a) Training set accuracy and loss, (b)Test set accuracy,(c) Network Architecture, (d) Confusion matrix for ResNeXt architecture

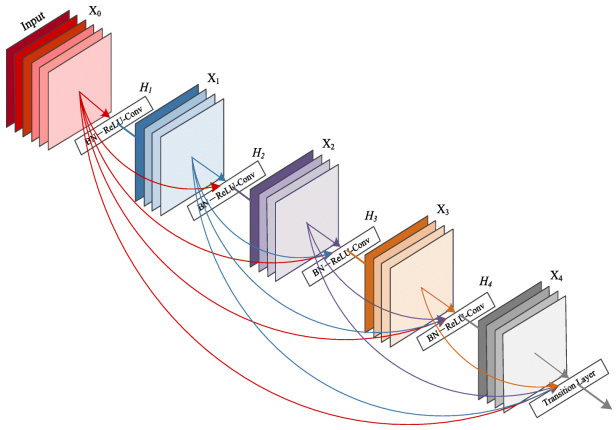

Figure 11. DenseNet block diagram

IV. Conclusion

In this homework we implemented Pb-lite algorithm to detect boundaries and different neural network architectures on CIFAR-10 dataset to develop an image classifier. For phase 1 the results are shown in the section titles ”Phase-1 results” . The networks were trained and the results have been shown in the table V provided in the corresponding subsection. A confusion matrix displaying the predictions vs true labels has been provided to analyse false positives. From the exercise of implementation of different types of Neural Nets it has been observed that each architecture has its own pros and cons. A simple CNN takes very small time to train but the test results are not very accurate. In theory it can be said that by making the CNN deeper and deeper the accuracy of the network will improve, but this is not the case. As per the results the simple CNN trained on standardized and augmented data performed the best and got an accuracy of 70.4%. Also ResNet performed better than ResNeXt and DenseNet. DenseNet and ResNeXt have more

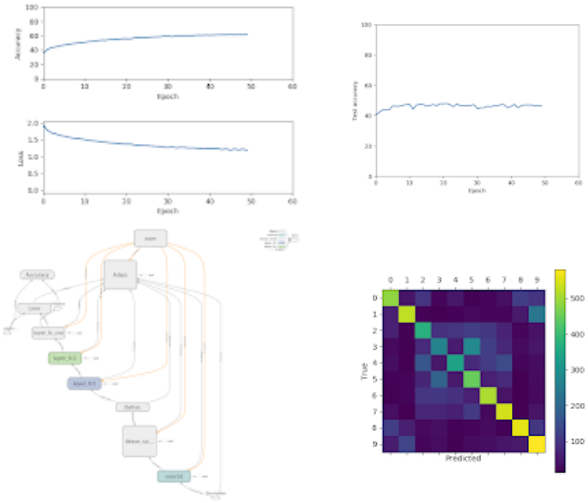

Figure 12. Serially from top left image: (a) Training set accuracy and loss, (b)Test set accuracy,(c) Network Architecture, (d) Confusion matrix for DenseNet architecture

features packed in reletively shallow network than ResNet and simple CNN. ResNeXt and DenseNet could provide more accuracy,according to the results show in respective paper, if they are trained for more epoch

References

[1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik, *Contour Detection and Hierarchical Image Segmentation*IEEE Trans. Pattern Anal. Mach. Intell. 33, 5 (May 2011), 898-916 Knuth: Computers and Typesetting, http://dx.doi.org/10.1109/TPAMI.2010.161

[2] Gabor Filters. https://en.wikipedia.org/wiki/Gabor$_f$ilterHvaas − $Labs/Tensorflow - Turorials$. https://github.com/Hvass-Labs/TensorFlow-Tutorials

[3] Official Tensor Flow Turorials. https://www.tensorflow.org/tutorials/images/deep$_c$nn

[4] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, Quoc V. Le, *Don't Decay the Learning Rate, Increase the Batch Size*

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition* https://dblp.org/rec/bib/journals/corr/HeZRS15

[6] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He *Aggregated Residual Transformations for Deep Neural Networks* https://arxiv.org/abs/1611.05431

[7] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger *Densely Connected Convolutional Networks* https://arxiv.org/abs/1608.06993

V. Phase-1 Results

Figure 13. From left to right: (a)Test image (b)Pb-lite Output

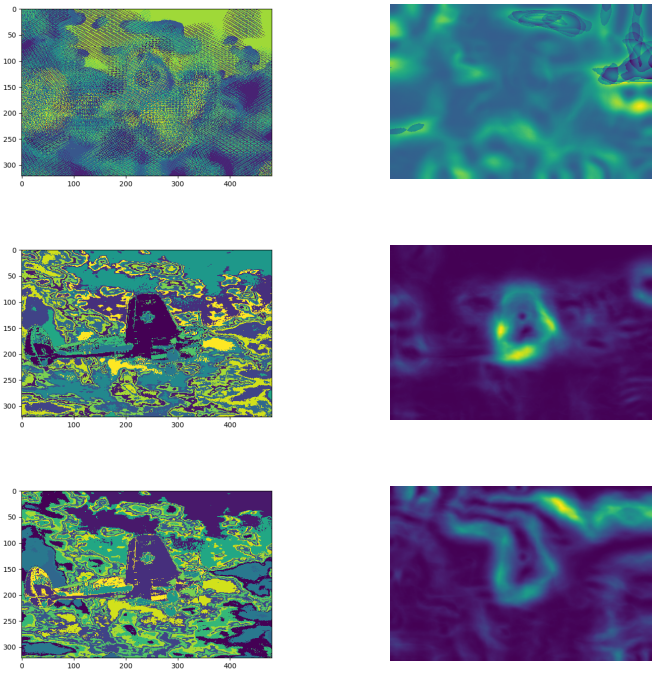

Figure 14. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 1

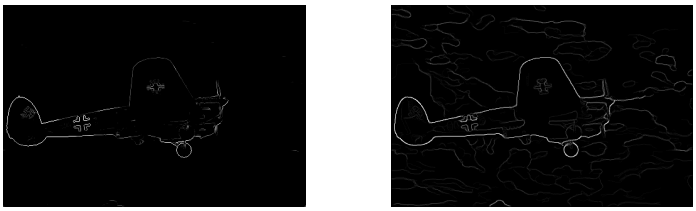

Figure 15. from left to right: (a)Sobel Baseline, (b)Canny Baseline

Figure 16. From left to right: (a)Test image (b)Pb-lite Output

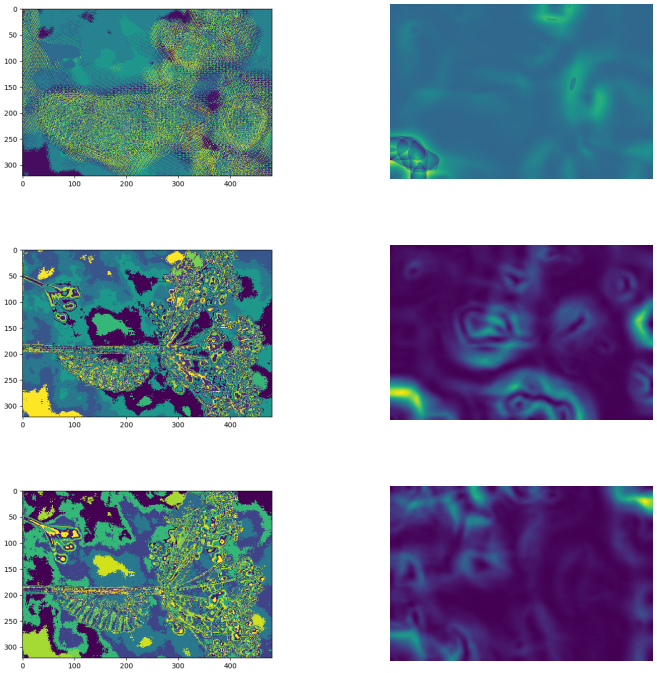

Figure 17. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 2

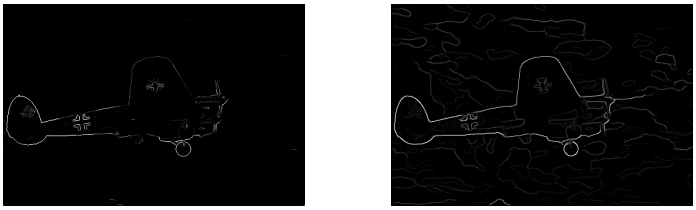

Figure 18. from left to right: (a)Sobel Baseline, (b)Canny Baseline

Figure 19. From left to right: (a)Test image (b)Pb-lite Output

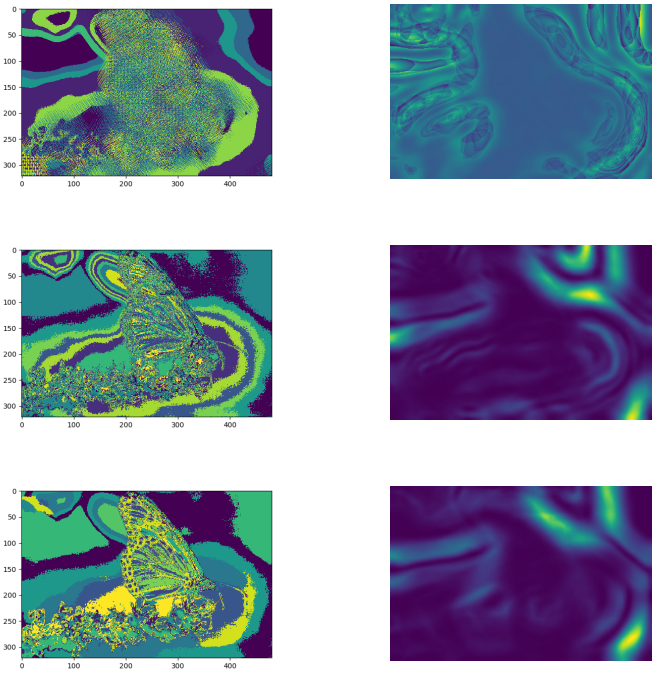

Figure 20. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 3

Figure 21. From left to right: (a)Test image (b)Pb-lite Output

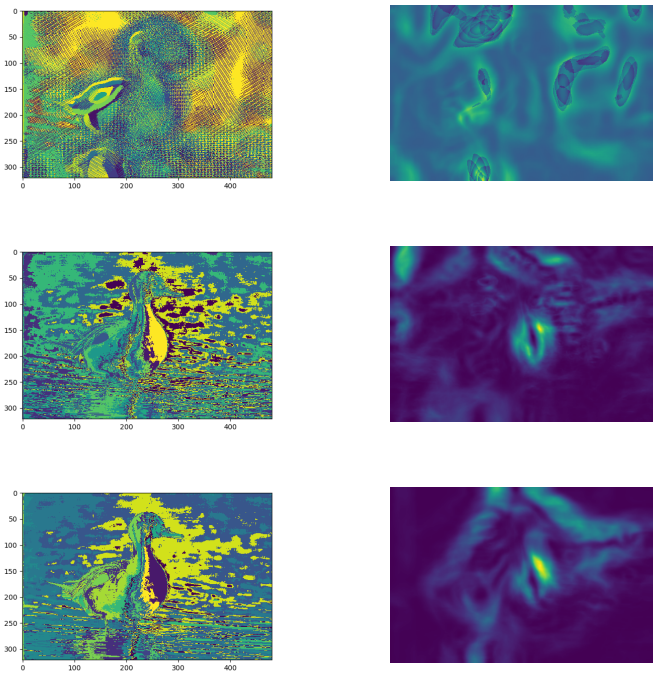

Figure 22. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 4

Figure 23. From left to right: (a)Test image (b)Pb-lite Output

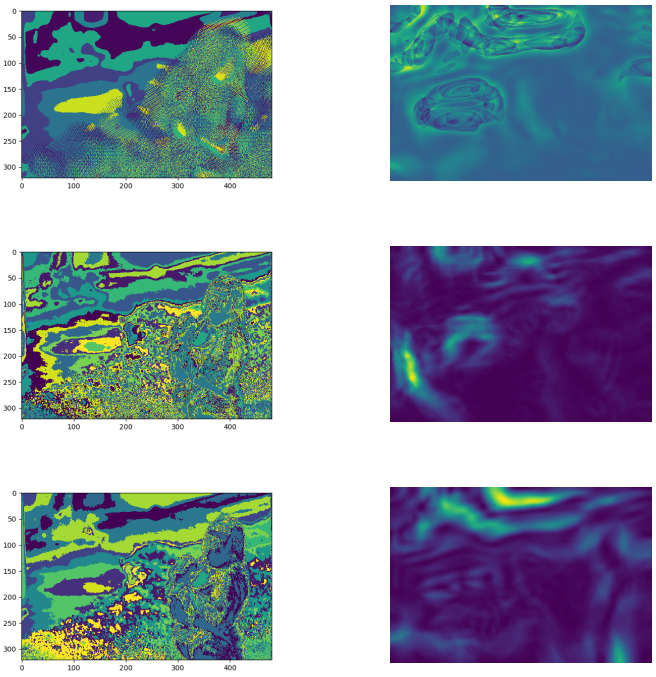

Figure 24. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 5

Figure 25. From left to right: (a)Test image (b)Pb-lite Output

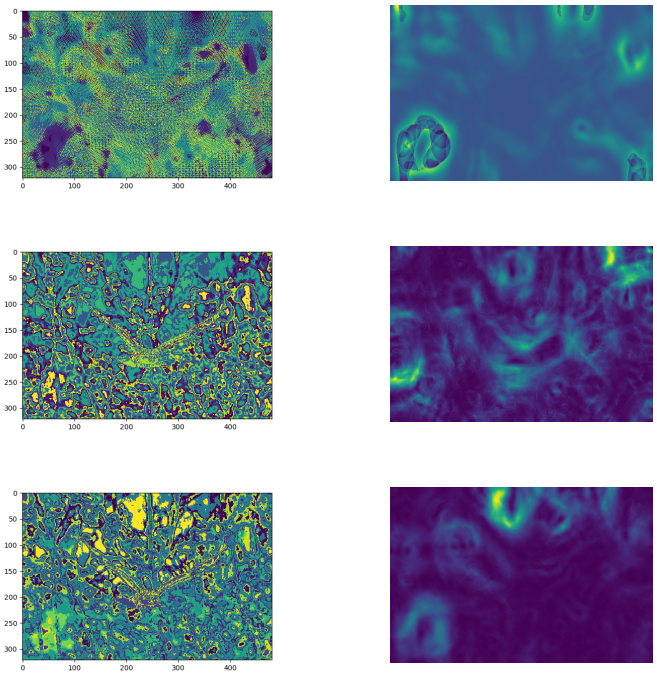

Figure 26. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 6

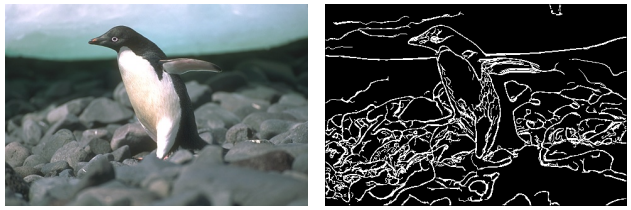

Figure 27. From left to right: (a)Test image (b)Pb-lite Output

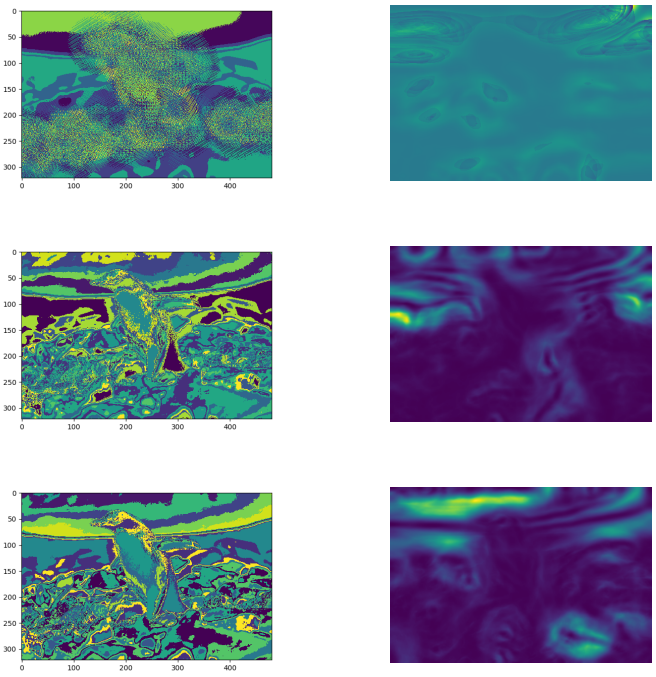

Figure 28. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 7

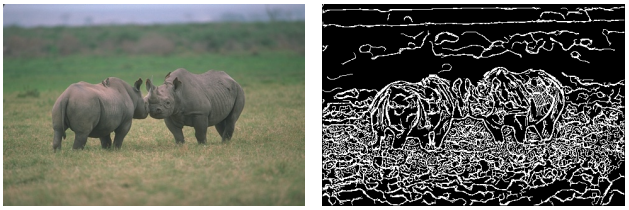

Figure 29. From left to right: (a)Test image (b)Pb-lite Output

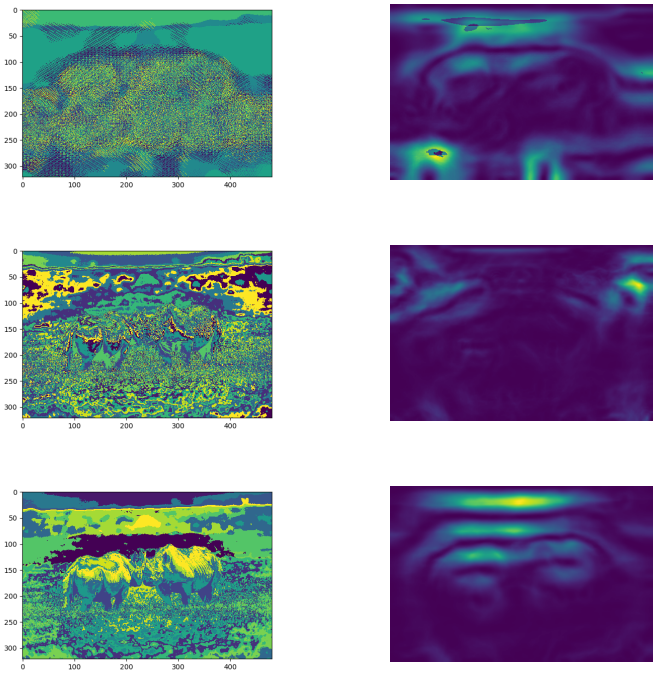

Figure 30. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 8

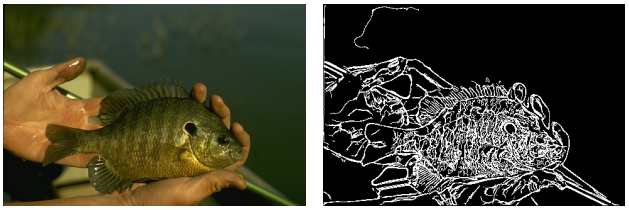

Figure 31. From left to right: (a)Test image (b)Pb-lite Output

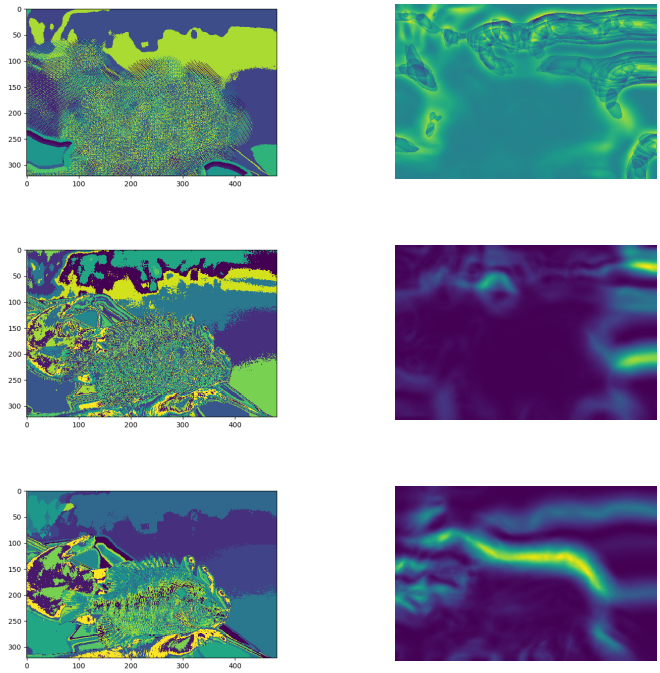

Figure 32. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 9

Figure 33. From left to right: (a)Test image (b)Pb-lite Output

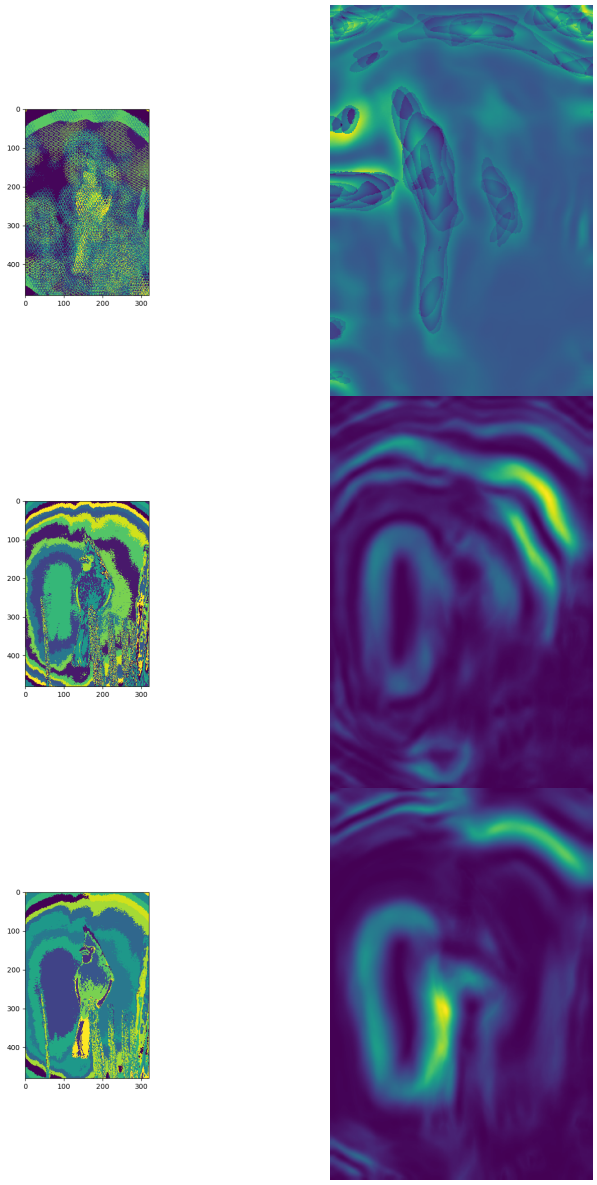

Figure 34. Serially from top left image: (a) Texton Map, (b) Texton gradient,(d) Brightness Map, (d) Brightness gradient, (e) Color Map and (f) Color gradient for test image 10

Architecture	Optimizer	Learning rate	Batchsize	Epochs	Train Accuracy	Test Accuracy	#Images	#Parameters	Avg. testin
Simple CNN (w/ data augmentation)	Adam	1e-3	128	25	96.3%	64.23%	50000	2500874	16.01 s
Simple CNN (w/ data augmentation)	Adam	1e-3	128	75	73.18%	70.4%	500000	2500874	18.08 s
ResNet	Adam	1e-3	128	25	65.62%	67.27%	500000	409066	23.01 s
ResNeXt	Adam	1e-3	128	25	50.76%	51.36%	500000	86314	57.92 s
DenseNet	Adam	1e-3	128	25	62.4%	43.32%	500000	1738714	73.3 s