Homework 0 - Alohomora

Zhentian Qian
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts
Email: zqian@wpi.edu

I. Phase 1: Shake My Boundary

A. Filter Banks

1) Oriented DoG filters: The Gaussian kernel has the following form:

$$g(x,y) = \eta e^{-\frac{1}{2}\cdot((x^2+y^2)/\sigma^2)} \tag{1}$$

where η is a normalizing constant, σ^2 is the variance.

To calculate the derivative of Gaussian kernel, we use the Soble operator [1]:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2}$$

The derivative of the Gaussian kernel in x and y direction are subsequently:

$$\frac{\delta g}{\delta x} = G_x * g(x,y), \qquad \frac{\delta g}{\delta y} = G_y * g(x,y) \tag{3}$$

Where $*$ represents the convolution operation.

The oriented derivative of Gaussian is the directional derivative of the Gaussian kernel. Suppose the directional vector $\mathbf{s} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$. The directional derivative $\Delta_{\mathbf{s}} g$ can be calculated as:

$$\Delta_{\mathbf{s}} g = \Delta g \cdot s = \cos\theta \frac{\delta g}{\delta x} + \sin\theta \frac{\delta g}{\delta y} \tag{4}$$

The final calculated oriented DoG filters are visulaized in Fig. 1.

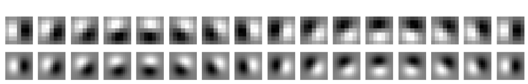

Fig. 1: Oriented DoG filter bank.

2) Leung-Malik Filters [2]: The Gaussian kernel with different variance in x and y direction can be written as:

$$g(x,y) = \eta e^{-\frac{1}{2}\cdot(x^2/\sigma_x^2 + y^2/\sigma_y^2)}$$

The first and second order derivatives of Gaussian at orientation $\theta = 0$ can be calculated analytically:

$$DoG(x,y) = \frac{\delta g}{\delta x} = -\eta \cdot x \cdot e^{(-\frac{1}{2}x^2/\sigma_x^2)} \cdot e^{(-\frac{1}{2}y^2/\sigma_y^2)} \tag{5}$$

$$D2oG(x,y) = \frac{\delta^2 g}{\delta x^2} = -\eta \cdot (x^2 - \sigma^2) \cdot e^{(-\frac{1}{2}x^2/\sigma_x^2)} \cdot e^{(-\frac{1}{2}y^2/\sigma_y^2)} \tag{6}$$

To calculate the first and second order derivatives of Gaussian at other orientations, we simply need to rotate the x, y coordinate by angle θ and substitute the rotated coordinates into (5) and (6). The rotated coordinates are:

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{7}$$

The Laplacian of Gaussian (LOG) filter kernel can also be calculated analytically:

$$LoG(x,y) = \frac{\delta^2 g}{\delta x^2} + \frac{\delta^2 g}{\delta y^2} = -\eta\Big(1 - \frac{x^2+y^2}{2\sigma^2}\Big) e^{-\frac{1}{2}\cdot((x^2+y^2)/\sigma^2)} \tag{8}$$

The final calculated LM filters are visulaized in Fig. 2.

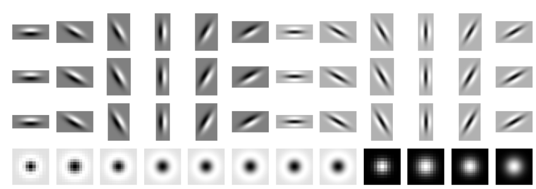

Fig. 2: Leung-Malik filter bank.

3) Gabor Filters: In the discrete domain, two-dimensional Gabor filters are given by [3]:

$$G_c[i,j] = Be^{-\frac{(i^2+j^2)}{2\sigma^2}} \cos(2\pi f(i\cos\theta + j\sin\theta)) \tag{9}$$

$$G_s[i,j] = Ce^{-\frac{(i^2+j^2)}{2\sigma^2}} \sin(2\pi f(i\cos\theta + j\sin\theta)) \tag{10}$$

where B and C are normalizing factors. The final calculated Gabor filters are visualized in Fig. 3.

B. Texton Map

The input image is then filtered with each element of the filter banks. The filter responses produced by Oriented DoG, LMS, LML and Gabor filter banks of input image 1 are visualized in Figs. 4–7. The filter responses are then concatenated into a $N \times W \times H$ array, where $N = 168$ is the total number of filters, and W and H are the dimensions of the image. The filter responses at all pixels in the image are then clustered into the $K = 64$ textons using kmeans algorithm. The generated texton maps for all images are visualized in Fig. 8.

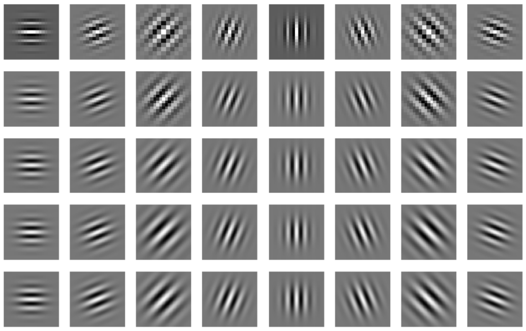

Fig. 3: Gabor filter bank.

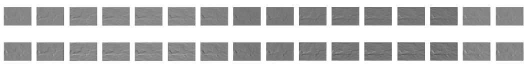

Fig. 4: Oriented DoG result of image 1.

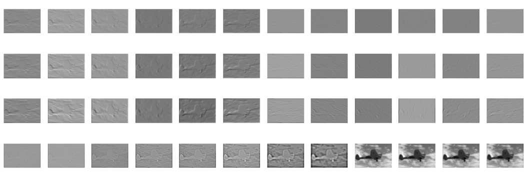

Fig. 5: LMS filter responses of image 1.

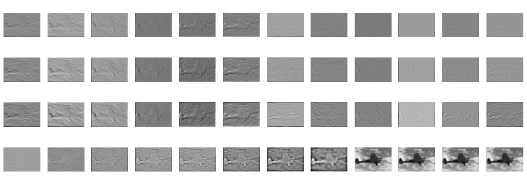

Fig. 6: LML filter responses of image 1.

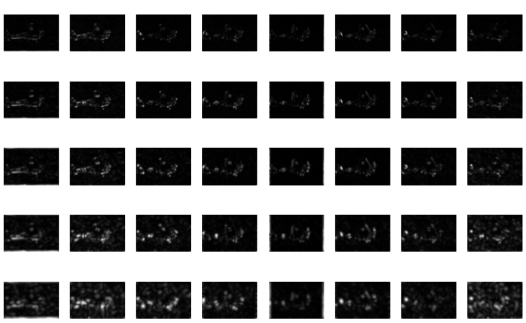

Fig. 7: Gabor filter responses of image 1.

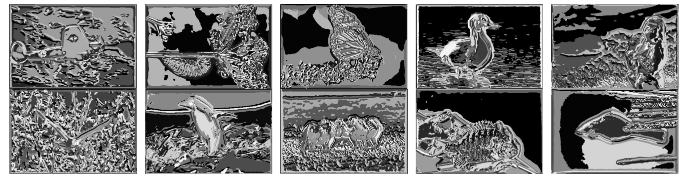

Fig. 8: Texton map $\mathcal{T}$ for all images

C. Brightness Map

To generate the brightness map, we first transform the image into the Lab [4] color space. The L channel for perceputal lightness is considered as the brightness of the image. K-means algorithm with K = 16 clusters are subsequently performed on image brightness data. The generated brightness maps for all images are visualized in Fig. 9.

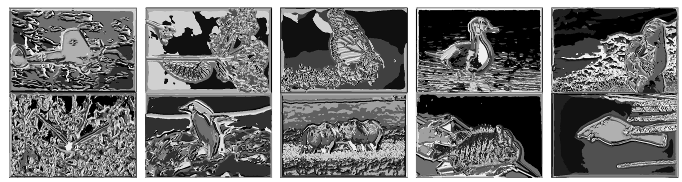

Fig. 9: Brightness map $\mathcal{B}$ for all images

D. Color Map

K-means algorithm with $k = 16$ clusters are run on the RGB channels of the image to produce to color map C. The generated color maps for all images are visualized in Fig. 9.

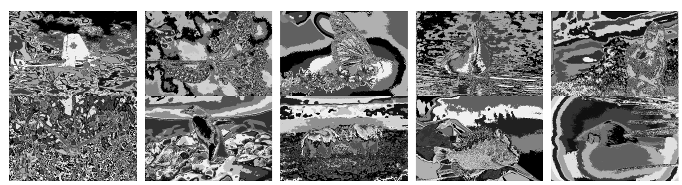

Fig. 10: Color map $\mathcal{C}$ for all images

E. Texture, Brightness and Color Gradients

The implemented half disks are visualized in Fig. 11. The half-disc masks are pairs of binary images of half-discs at different orientation and scale.

The algorithm we use to calculate the map gradients are described in Algorithm 1. Here we present the calculated gradient for Texton, brightnesss and Color maps of image 1 at orientation $\theta = 0°$ and $\theta = 90°$, visualized in Figs. 12, 13, 14.

F. Pb-lite Output

The original images are visualized in Fig. 15. The canny and sobel baselines are visualized in Figs. 16 and 17. The ground truth is visualized in Figs. 18. The PB-lite outputs are calculated based on the formula:

$$PBEdges = \frac{\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g}{3} \odot (0.5 * cannyPb + 0.5 * sobelPb) \tag{11}$$

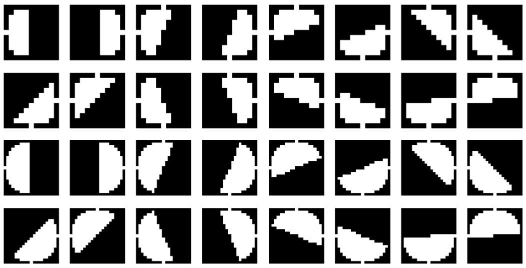

Fig. 11: Half disc masks at different scales and orientations.

Algorithm 1: Chi-square distance calculation procedure

Data: img
Result: chi_sqr_dist
chi_sqr_dist = img$*$0;
for *i* = *1:num_bins* **do**
    tmp = 1 where img is in bin i and 0 elsewhere;
    g_i = convolve tmp with left_mask;
    h_i = convolve tmp with right_mask;
    chi_sqr_dist += $\frac{1}{2} \cdot \frac{(g_i - h_i)^2}{g_i + h_i}$
end

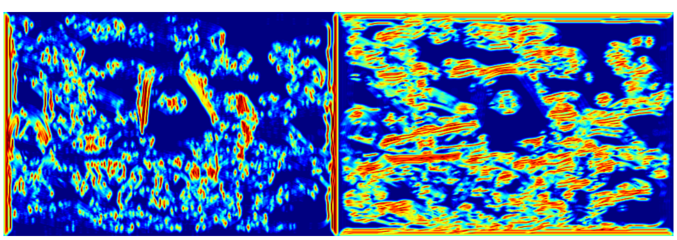

Fig. 12: Texton map gradient $\mathcal{T}_g$ of image 1 at $\theta = 0°$ and $\theta = 90°$.

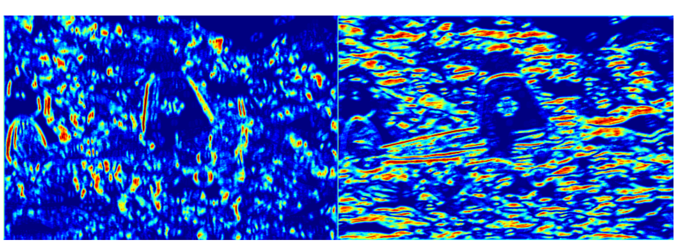

Fig. 13: Brightness map gradient $\mathcal{B}_g$ of image 1 at $\theta = 0°$ and $\theta = 90°$.

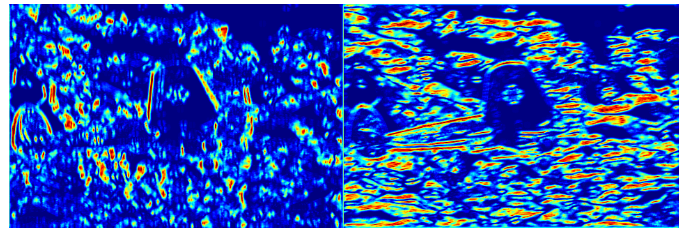

Fig. 14: Color map gradient $\mathcal{C}_g$ of image 1 at $\theta = 0°$ and $\theta = 90°$.

and are visualized in Fig. 19. Comparing the Pb-lite output with the sobel and canny baselines, we can see that false positive edges of the canny and soble baselines are suppressed in the Pb-lite output while true edges still remain. It is due to that fact that Pb-output is able to uses the global information of the image and also combine multiscale cues [5].

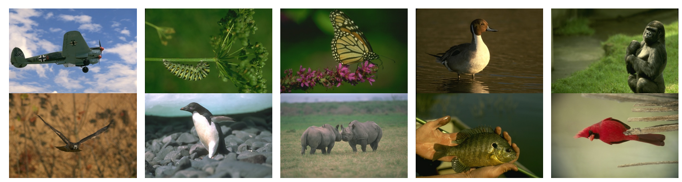

Fig. 15: Input images from the BSDS500 dataset.

Fig. 16: Canny baseline.

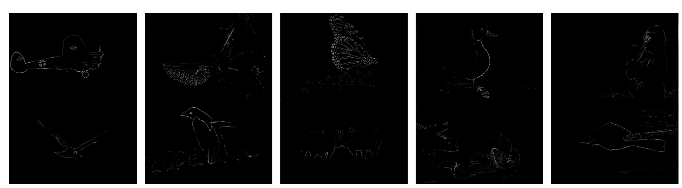

Fig. 17: Soble baseline.

II. Phase 2:Deep Dive on Deep Learning

A. *Train your first neural network*

The first neural network designed is a simple convolutional neural network, as visualized in Fig. 20. There are 30166 parameters in this model. We use a stochastic gradient decent optimizer for learning, with a learning rate $l_r = 0.001$ and a batch size of 32. The train and test accuracy over epochs are visualized in Figs. 21 and 22. Loss over epochs is visualized in Fig. 23. The confusion matrix of the trained model on

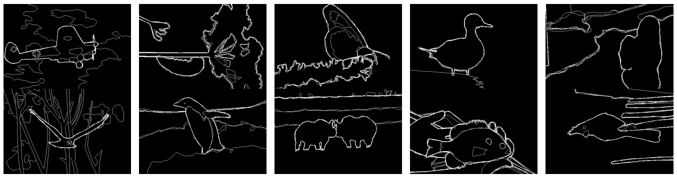

Fig. 18: Ground Truth.

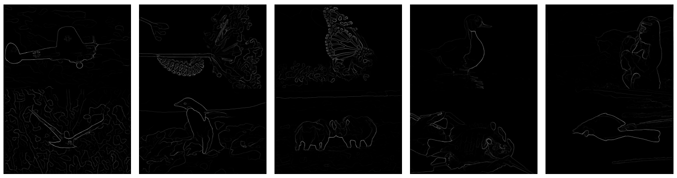

Fig. 19: Pb-lite output for all images

training data is:

$$\begin{bmatrix} 3852 & 62 & 99 & 119 & 71 & 65 & 30 & 85 & 356 & 261 \\ 53 & 4337 & 7 & 26 & 7 & 9 & 29 & 16 & 63 & 453 \\ 215 & 19 & 2897 & 478 & 425 & 326 & 286 & 224 & 57 & 73 \\ 63 & 19 & 102 & 3570 & 174 & 585 & 146 & 218 & 33 & 90 \\ 91 & 15 & 106 & 308 & 3714 & 138 & 131 & 395 & 45 & 57 \\ 22 & 15 & 88 & 812 & 139 & 3511 & 78 & 275 & 12 & 48 \\ 16 & 41 & 93 & 383 & 137 & 114 & 4105 & 35 & 12 & 64 \\ 23 & 5 & 40 & 179 & 118 & 124 & 11 & 4405 & 15 & 80 \\ 126 & 95 & 21 & 76 & 12 & 21 & 21 & 18 & 4383 & 227 \\ 49 & 99 & 7 & 32 & 7 & 19 & 17 & 38 & 43 & 4689 \end{bmatrix} \tag{12}$$

The confusion matrix of the trained model on testing data is:

$$\begin{bmatrix} 658 & 23 & 36 & 36 & 19 & 13 & 14 & 22 & 108 & 71 \\ 28 & 758 & 3 & 15 & 7 & 7 & 10 & 6 & 23 & 143 \\ 71 & 8 & 429 & 118 & 118 & 94 & 69 & 51 & 14 & 28 \\ 20 & 9 & 32 & 519 & 66 & 191 & 47 & 72 & 9 & 35 \\ 32 & 6 & 48 & 89 & 583 & 43 & 63 & 108 & 14 & 14 \\ 11 & 2 & 37 & 247 & 44 & 529 & 26 & 83 & 7 & 14 \\ 8 & 10 & 31 & 114 & 42 & 40 & 729 & 13 & 4 & 9 \\ 18 & 5 & 15 & 62 & 49 & 68 & 5 & 739 & 5 & 34 \\ 61 & 39 & 5 & 31 & 6 & 9 & 5 & 4 & 781 & 59 \\ 25 & 72 & 3 & 18 & 4 & 13 & 7 & 12 & 23 & 823 \end{bmatrix} \tag{13}$$

B. *Improving Accuracy of your neural network*

Multiple approaches are implemented to improve the accuracy of the neural network:

1) Standardize the data input. The data is scaled from [0,255] to [-1,1].
2) Decay the learning rate exponentially.
3) Increase the batch size 5 times every 10 epochs.
4) Data augmentation with random crop and flip.
5) Batch Normalization.

The test results suggest that method 1, 3, 5 are effective in improving accuracy. The network architecture is still as defined in Fig. 20 with 30166 parameters. The stochastic gradient decent optimizer is used for learning, with a learning rate $l_r = 0.001$. The initial batch size is 32. The train and test accuracy over epochs are visualized in Figs. 24 and 25. Loss over epochs is visualized in Fig. 26. The confusion matrix of the trained model on training data is:

$$\begin{bmatrix} 4131 & 64 & 235 & 71 & 69 & 28 & 23 & 55 & 236 & 88 \\ 68 & 4521 & 20 & 20 & 8 & 12 & 30 & 16 & 83 & 222 \\ 244 & 16 & 3578 & 210 & 362 & 202 & 199 & 119 & 52 & 18 \\ 69 & 18 & 235 & 3229 & 249 & 726 & 239 & 146 & 52 & 37 \\ 86 & 10 & 237 & 190 & 3944 & 143 & 149 & 203 & 26 & 12 \\ 29 & 7 & 160 & 661 & 192 & 3619 & 94 & 212 & 11 & 15 \\ 21 & 20 & 169 & 239 & 128 & 107 & 4255 & 21 & 22 & 18 \\ 38 & 9 & 120 & 129 & 191 & 186 & 24 & 4271 & 11 & 21 \\ 217 & 80 & 46 & 37 & 26 & 15 & 25 & 11 & 4465 & 78 \\ 97 & 212 & 27 & 40 & 14 & 15 & 25 & 55 & 103 & 4412 \end{bmatrix} \tag{14}$$

The confusion matrix of the trained model on testing data is:

$$\begin{bmatrix} 728 & 19 & 62 & 21 & 26 & 12 & 12 & 15 & 72 & 33 \\ 26 & 810 & 7 & 12 & 9 & 2 & 9 & 6 & 31 & 88 \\ 65 & 7 & 596 & 54 & 87 & 60 & 63 & 38 & 17 & 13 \\ 23 & 10 & 68 & 520 & 60 & 184 & 71 & 31 & 18 & 15 \\ 14 & 3 & 63 & 62 & 680 & 43 & 44 & 77 & 12 & 2 \\ 10 & 2 & 51 & 203 & 50 & 588 & 19 & 68 & 5 & 4 \\ 8 & 6 & 58 & 66 & 36 & 33 & 775 & 6 & 6 & 6 \\ 12 & 5 & 36 & 37 & 55 & 66 & 5 & 772 & 1 & 11 \\ 78 & 32 & 17 & 13 & 10 & 8 & 7 & 4 & 796 & 35 \\ 23 & 79 & 8 & 12 & 8 & 6 & 5 & 26 & 29 & 804 \end{bmatrix} \tag{15}$$

C. *ResNet, ResNeXt, DenseNet*

1) ResNet: The Resnet architecture is visualized in Fig. 27. There are 98858 parameters in this model. We use a stochastic gradient decent optimizer for learning, with a learning rate $l_r = 0.001$ and a batch size of 32. The train and test accuracy over epochs are visualized in Figs. 28 and 29. Loss over epochs is visualized in Fig. 30. The confusion matrix of the Resnet on training data is:

$$\begin{bmatrix} 4325 & 54 & 192 & 55 & 24 & 16 & 39 & 47 & 172 & 76 \\ 55 & 4732 & 10 & 12 & 3 & 2 & 9 & 1 & 34 & 142 \\ 228 & 10 & 3684 & 218 & 378 & 104 & 231 & 106 & 32 & 9 \\ 70 & 6 & 232 & 3290 & 157 & 858 & 230 & 88 & 40 & 29 \\ 56 & 2 & 268 & 186 & 3908 & 120 & 145 & 292 & 18 & 5 \\ 22 & 4 & 126 & 662 & 167 & 3773 & 47 & 183 & 8 & 8 \\ 35 & 13 & 199 & 281 & 127 & 63 & 4250 & 11 & 10 & 11 \\ 49 & 4 & 91 & 79 & 182 & 213 & 4 & 4357 & 5 & 16 \\ 144 & 44 & 19 & 31 & 13 & 4 & 11 & 6 & 4674 & 54 \\ 86 & 134 & 10 & 27 & 2 & 6 & 8 & 23 & 45 & 4659 \end{bmatrix} \tag{16}$$

The confusion matrix of the Resnet on testing data is:

$$\begin{bmatrix} 771 & 18 & 59 & 16 & 10 & 5 & 10 & 14 & 67 & 30 \\ 23 & 888 & 3 & 2 & 1 & 3 & 4 & 1 & 15 & 60 \\ 70 & 1 & 639 & 58 & 77 & 49 & 66 & 27 & 8 & 5 \\ 25 & 5 & 67 & 573 & 40 & 199 & 54 & 16 & 9 & 12 \\ 15 & 2 & 70 & 58 & 683 & 36 & 53 & 73 & 7 & 3 \\ 9 & 5 & 25 & 173 & 53 & 675 & 15 & 38 & 3 & 4 \\ 14 & 2 & 47 & 66 & 44 & 19 & 798 & 4 & 5 & 1 \\ 14 & 0 & 34 & 26 & 68 & 56 & 3 & 783 & 1 & 15 \\ 61 & 19 & 0 & 9 & 4 & 2 & 5 & 5 & 874 & 21 \\ 23 & 48 & 5 & 10 & 3 & 3 & 4 & 9 & 13 & 882 \end{bmatrix} \tag{17}$$

2) ResNeXt: The ResneXt architecture is visualized in Fig. 31. There are 1212266 parameters in this model. We use a stochastic gradient decent optimizer for learning, with a learning rate $l_r = 0.001$ and a batch size of 32. The train and test accuracy over epochs are visualized in Figs. 32 and 33. Loss over epochs is visualized in Fig. 34. The confusion matrix of the ResneXt on training data is:

$$\begin{bmatrix} 5000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4999 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 4987 & 0 & 1 & 1 & 7 & 0 & 1 & 0 \\ 0 & 0 & 0 & 4992 & 0 & 4 & 1 & 0 & 1 & 2 \\ 1 & 0 & 0 & 2 & 4994 & 1 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 4994 & 2 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 4999 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 4999 & 0 & 0 \\ 2 & 0 & & 0 & 0 & 0 & 0 & 0 & 04998 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4999 \end{bmatrix} \tag{18}$$

The confusion matrix of the ResneXt on testing data is:

$$\begin{bmatrix} 786 & 28 & 37 & 13 & 7 & 3 & 19 & 18 & 67 & 22 \\ 49 & 825 & 3 & 4 & 7 & 2 & 8 & 3 & 12 & 87 \\ 81 & 7 & 492 & 67 & 100 & 49 & 139 & 45 & 9 & 11 \\ 28 & 17 & 55 & 490 & 49 & 194 & 97 & 33 & 15 & 22 \\ 16 & 9 & 72 & 48 & 631 & 45 & 101 & 59 & 9 & 10 \\ 12 & 6 & 22 & 152 & 51 & 658 & 38 & 40 & 5 & 16 \\ 15 & 7 & 29 & 43 & 38 & 20 & 829 & 8 & 8 & 3 \\ 34 & 2 & 29 & 33 & 67 & 88 & 16 & 712 & 3 & 16 \\ 86 & 31 & 7 & 7 & 3 & 4 & 5 & 3 & 825 & 29 \\ 40 & 87 & 5 & 3 & 5 & 3 & 3 & 14 & 24 & 816 \end{bmatrix} \tag{19}$$

3) DenseNet: The DenseNet architecture is visualized in Fig. 35. There are 10634 parameters in this model. We use a stochastic gradient decent optimizer for learning, with a learning rate $l_r = 0.001$ and a batch size of 32. The train and test accuracy over epochs are visualized in Figs. 36 and 37. Loss over epochs is visualized in Fig. 38. The confusion

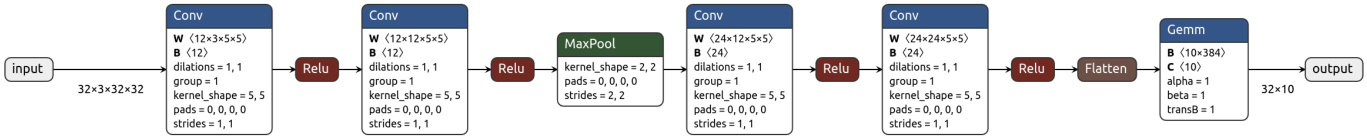

Fig. 20: Convolutional neural network.

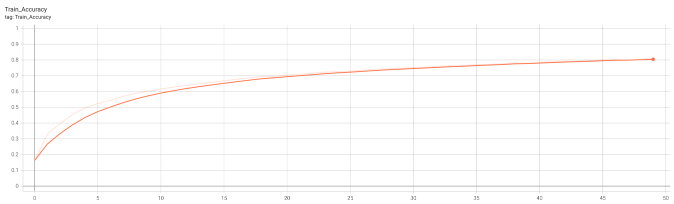

Fig. 21: Train accuracy over epochs.

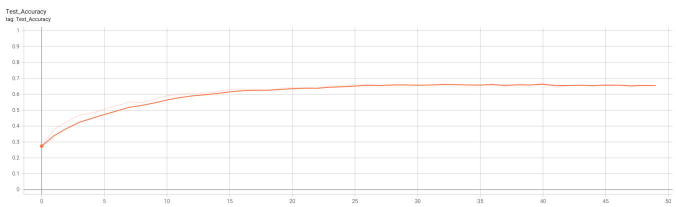

Fig. 22: Test accuracy over epochs.

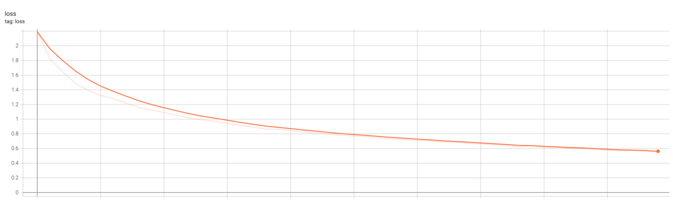

Fig. 23: Loss over epochs.

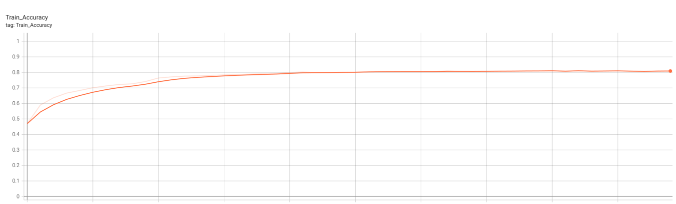

Fig. 24: Train accuracy over epochs of the improved network.

Fig. 25: Test accuracy over epochs of the improved network.

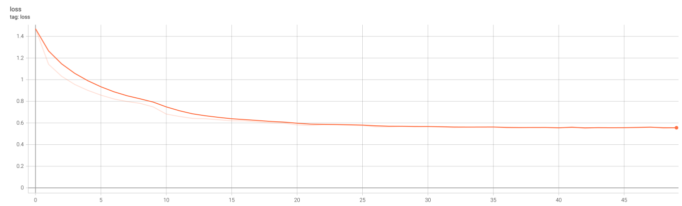

Fig. 26: Loss over epochs of the improved network.

matrix of the DenseNet on training data is:

$$\begin{bmatrix} 3951 & 118 & 119 & 104 & 34 & 38 & 24 & 48 & 437 & 127 \\ 105 & 4503 & 1 & 30 & 5 & 9 & 9 & 5 & 92 & 241 \\ 481 & 13 & 2849 & 471 & 506 & 179 & 252 & 168 & 68 & 13 \\ 72 & 20 & 177 & 3220 & 264 & 813 & 201 & 117 & 64 & 52 \\ 135 & 8 & 219 & 253 & 3727 & 126 & 129 & 342 & 52 & 9 \\ 31 & 15 & 113 & 873 & 278 & 3411 & 42 & 215 & 9 & 13 \\ 74 & 29 & 194 & 571 & 323 & 55 & 3689 & 9 & 47 & 9 \\ 104 & 12 & 98 & 180 & 357 & 302 & 6 & 3896 & 11 & 34 \\ 206 & 60 & 19 & 40 & 12 & 10 & 6 & 13 & 4553 & 81 \\ 115 & 272 & 8 & 47 & 15 & 6 & 5 & 33 & 93 & 44069 \end{bmatrix} \tag{20}$$

The confusion matrix of the Resnet on testing data is:

$$\begin{bmatrix} 750 & 34 & 25 & 24 & 9 & 7 & 5 & 11 & 108 & 27 \\ 20 & 869 & 0 & 4 & 2 & 0 & 3 & 2 & 28 & 72 \\ 113 & 2 & 505 & 98 & 114 & 51 & 64 & 27 & 18 & 8 \\ 24 & 7 & 45 & 582 & 58 & 179 & 45 & 30 & 21 & 9 \\ 21 & 1 & 59 & 60 & 710 & 25 & 31 & 74 & 18 & 1 \\ 11 & 6 & 30 & 193 & 62 & 639 & 8 & 42 & 4 & 5 \\ 19 & 3 & 49 & 117 & 76 & 8 & 720 & 2 & 4 & 2 \\ 28 & 3 & 28 & 31 & 76 & 92 & 2 & 725 & 2 & 13 \\ 57 & 19 & 2 & 15 & 3 & 5 & 2 & 2 & 878 & 17 \\ 35 & 61 & 1 & 11 & 5 & 4 & 2 & 10 & 26 & 845 \end{bmatrix} \tag{21}$$

D. *Comparison*

The comparison between different neural network architectures are summarized in Tab. I. We can see that the original and the improved convolutional neural network has the shortest inference time. In terms of the training accuracy, the ResNext achieved the highest because of its large number of parameters. However, ResNext also shows severe over-fitting symptoms as the high-accuracy in training data set is accompanied with low-accuracy in the testing data set. Finally, in terms of testing accuracy, ResNet has achieved best performance. The feed-forward residual in the ResNet may have enabled more efficient training with deeper neural network architecture.

TABLE I: Comparison between different neural network architectures.

network	parameter num	train accuracy	test accuracy	inference time
Conv	30166	78.926%	65.48%	0.5ms
Improved	30166	80.85%	70.69%	0.49ms
ResNet	98858	83.304%	75.66%	0.85ms
ResNeXt	1212266	99.922%	70.64%	3.3ms
DenseNet	10634	76.41%	72.23%	0.85ms

References

[1] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

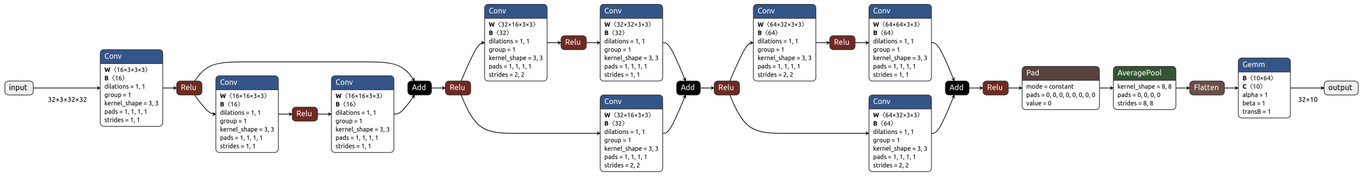

Fig. 27: Resnet.

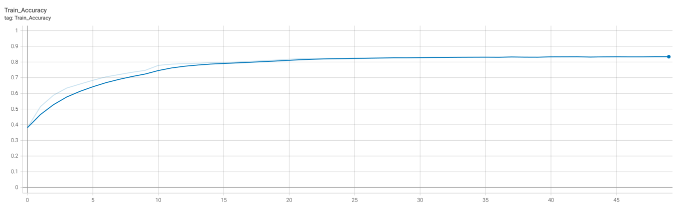

Fig. 28: Train accuracy over epochs of Resnet.

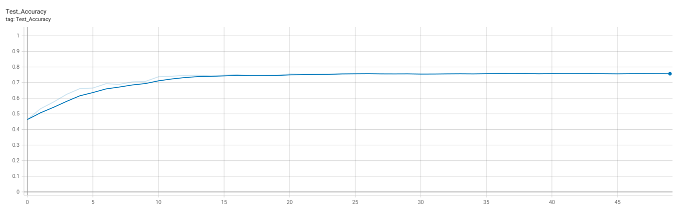

Fig. 29: Test accuracy over epochs of Resnet.

[2] T. Leung and J. Malik, “Representing and recognizing the visual appearance of materials using three-dimensional textons,” *International journal of computer vision*, vol. 43, no. 1, pp. 29–44, 2001.
[3] M. Haghighat, S. Zonouz, and M. Abdel-Mottaleb, “Identification using encrypted biometrics,” in *International Conference on Computer Analysis of Images and Patterns*. Springer, 2013, pp. 440–448.
[4] M. D. Fairchild and G. M. Johnson, “Image appearance modeling,” *Human Vision and Electronic Imaging VIII*, vol. 5007, pp. 149–160, 2003.
[5] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, “Contour detection and hierarchical image segmentation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 898–916, 2010.

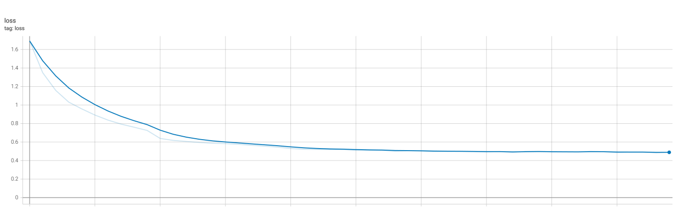

Fig. 30: Loss over epochs of Resnet.

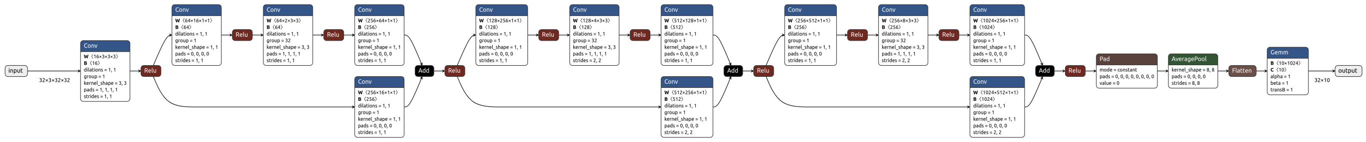

Fig. 31: Resnext.

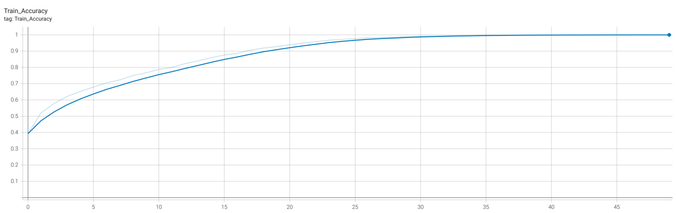

Fig. 32: Train accuracy over epochs of ResneXt.

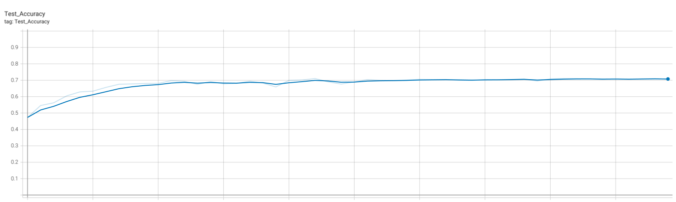

Fig. 33: Test accuracy over epochs of ResneXt.

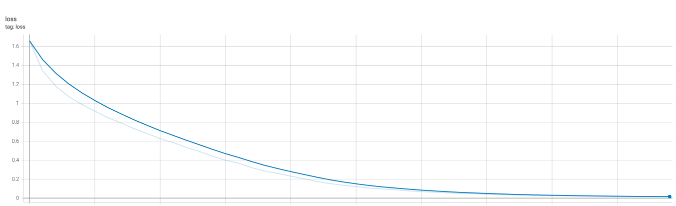

Fig. 34: Loss over epochs of ResneXt.

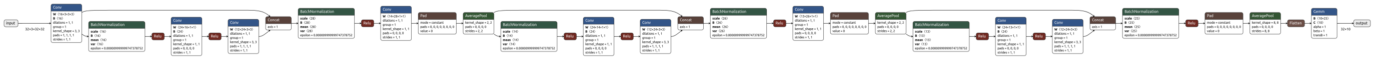

Fig. 35: DenseNet.

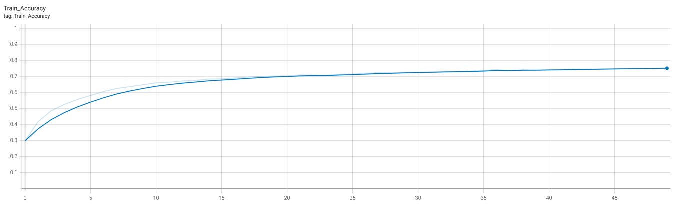

Fig. 36: Train accuracy over epochs of DenseNet.

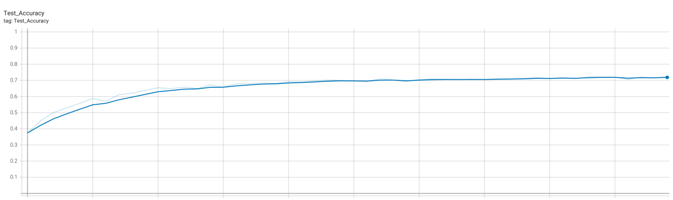

Fig. 37: Test accuracy over epochs of DenseNet.

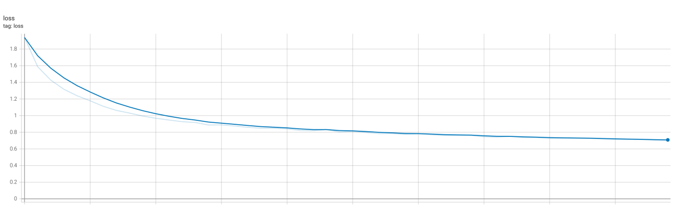

Fig. 38: Loss over epochs of DenseNet.