Homework 0 - Alohomora

Manoj Velmurugan
Robotics Engineering
Worcester Polytechnic Institute
Email: v.manoj1996@gmail.com

***Abstract*—In this study, we employed the Probability of Boundaries algorithm for edge detection, showcasing superior performance compared to traditional Canny and Sobel filters. Furthermore, the latter part of this project involved the training of deep neural networks to classify images within the CIFAR10 dataset. Decent accuracy of edge detection and image classification is demonstrated. Additionally, performance, accuracy and other factors were critically analysed and experimented in this work.**

One late day was used!

I. Phase 1: Shake My Boundary

Boundary detection in computer vision is a challenging problem, especially from a single image, as it may necessitate object-specific reasoning. One approach is to identify edges by detecting intensity discontinuities, which serve as potential indicators of transitions into background. Classical edge detection algorithms including the Canny and Sobel filters, use some form of gradient of intensity in images to detect the edges. Although they are computationally efficient, they can detect too many false positive edges. Instead of looking at just the change in intensities, once can make use of color and texture information to detect edges more robustly. In this project, we obtain a probability metric that quantifies whether a pixel is an edge or not. We multiply the probability with the canny and/or sobel edges to make a more robust edge detection algorithm.

A. *Texture Clustering*

The initial step in the PB Lite boundary detection pipeline involves filtering the image with three sets of filter banks. Subsequently, the filter responses are utilized to generate a texton map (denoted as T), capturing texture information through clustering. This filtering process plays a pivotal role in measuring texture properties and aggregating regional texture and brightness distributions.

Three types of filters are used to extract the textures from the images. They are Oriented DoG filters, Leung-Malik Filters, and Gabor filters. Oriented DoG filters (figure 1) capture the direction in which intensity changes. Leung-Malik Filters captures the direction (gradient), double derivative/ laplacian of the smoothed image and the brightness at differnt scales as illustrated in figure 2. Gabor filters mimic how humans see texture. They are formed by modulating a Gaussian kernal with a sine wave as shown in figure 3.
These filters capture too many information from the input image. Finally all we need is a single number/index encoding the texture information. This is done by clustering the available texture data spanning over N dimensions (equal to the number of filters used) using KMeans algorithm. We reduce our texture information down to 64 numbers assigned to each pixel by doing so.

B. *Color Clustering*

The color map concept captures color changes in an image by clustering color values using k-means clustering, considering RGB. 3 Dimensions (RGB) varying from 0-255 gets reduced to 0-15 while doing so. A single index gets assigned to each pixel clustering the image to regions of identical color. Figure 4 illustrates colormap for the given set of sample images.

C. *Intensity Clustering*

The brightness map concept involves capturing brightness changes by clustering grayscale equivalent values using k-means clustering. Effectively a number 0-15 is assigned to each pixel in the image once the clustering is done. Figure 4 illustrates intensity map for the given set of sample images. Color images were converted to intensity images using NTSC conversion formula which takes human color perception into account.

D. *Gradient of Texture, Color, Intensity*

The edge information is effectively encoded in the gradient of the texture, color and intensity maps. Since convolution operation is more effective than for loops, we use a convolutions kernel at different scales to perform this operation similar to the previous sections. The kernels are half disks as shown in figure 5. Any pixel with high difference between complementary filter operations must have strong change in features and thereby it must be an edge. Here χ^2 distance is used to compare the two filter operations.
To avoid recomputing, g-h and g+h are computed by combining the complementary set of filters into a single operation. For more information, please refer to the attached code. The gradients of texture, color and intensity obtained for each K values were added together to obtain the texture, color and intensity gradients as shown in figure 6.
Since we treat these gradients as probability, I scaled them between 0 and 1.

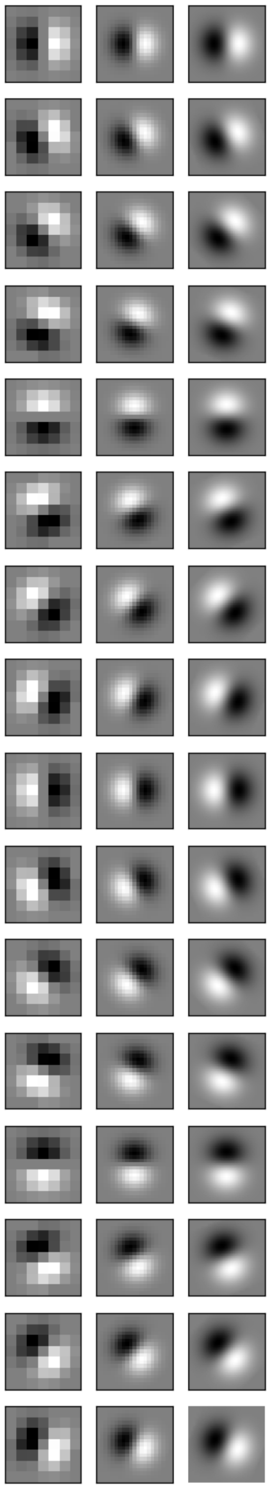
Fig. 1. Oriented DoG filters

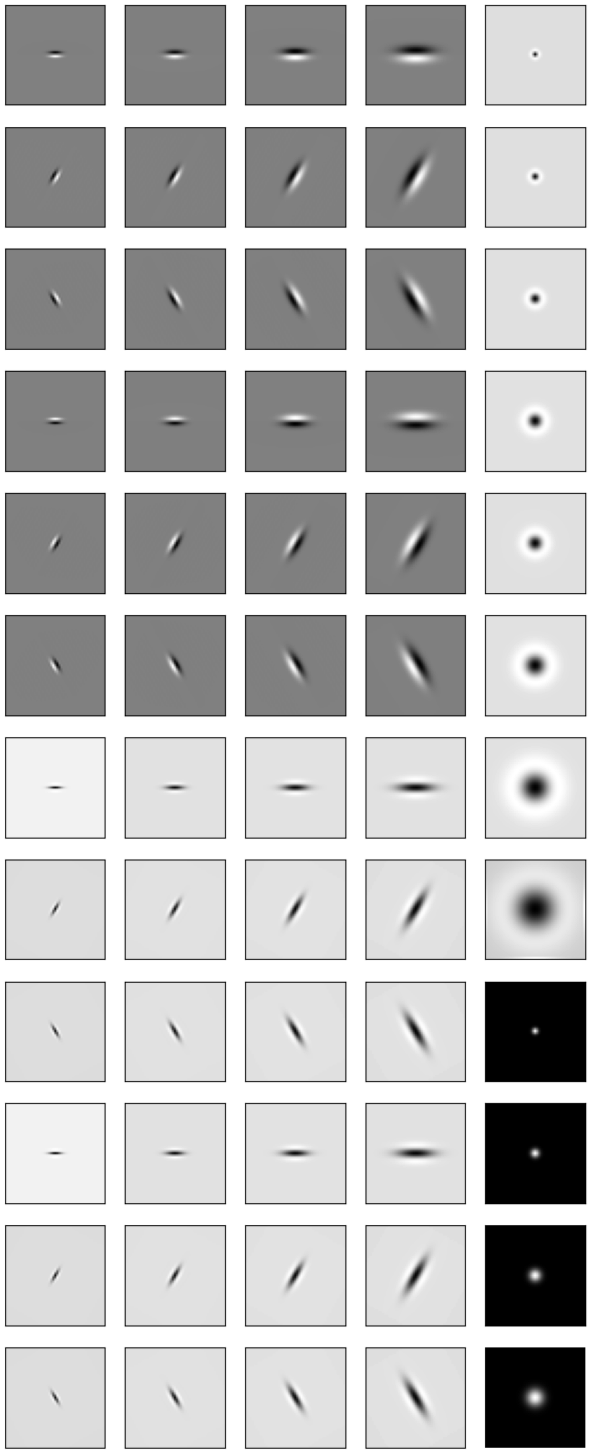
Fig. 2. Leung-Malik Filters

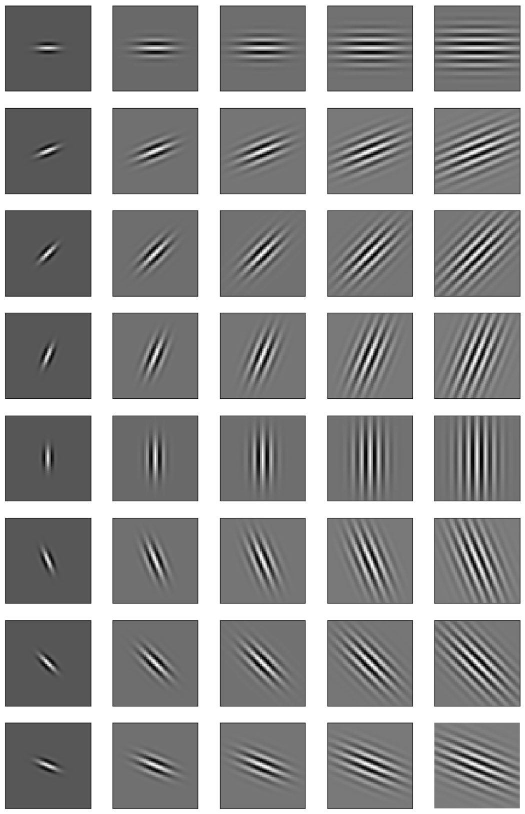
Fig. 3. Gabor Filters

E. Edge Detection

The provided canny and sobel filters were combined into one image in a complementary fashion and then multiplied with the normalized gradient of features to obtain the final set of edges as shown in figure 7.
Further, to improve the contrast and make it a binary image similar to the ground truth, the edges (Pb, Canny, Sobel) were threshold-ed as shown in figure 8.

F. Analysis and Key Challenges

- **Tuning** When I used different weights for Texture, Brightness and Color gradients, the resulting edges were inconsistent. As seen in figure 6, if texture works well for one image, color gradient works even better for a different image. So finally brightness, color and texture gradients were given similar weights.
- **Performance** KMeans clustering proved to have the biggest impact on performance in this work. A couple of techiques suggested by KMeans documentation was adopted to improve the same. Init method was changed to "k-means++" and the algorithm was changed to elkan.
- **Edge Detection Accuracy** Sobel filter leaves out important edges, while canny detector had detected too many edges. Pb detector maintains a balance between the two while suppressing edges in regions without much features . As shown in figure 7 and figure 8, the detector performed better than canny and sobel filters.
- **Color Map** The choice of color map makes a big difference in seeing the contrast in monochrome/grayscale images. Jet proved to have better contrast as seen in figure 7

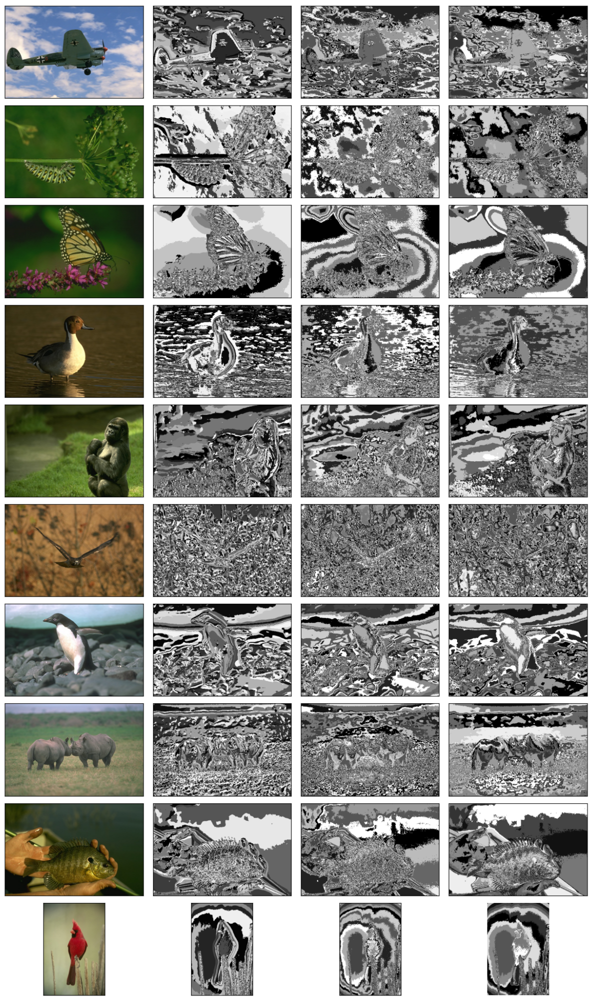
Fig. 4. Original Image, Texture, Brightness, Color Maps (Left to Right)

II. Phase 2: Deep Dive on Deep Learning

A. Problem Statement

Devise a Deep Neural Network to classify images on CIFAR10 Dataset.

B. Train your first neural network and Resnet architecture

1) Architecture: Resnet based architecture with six resnet layers was arbitrarily used for this work as shown in figure 9. After convolutional layers, the output is flattened and sent to 3 fully connected layers.

2) Training Process and Hyper Parameters: The loss function was chosen as Cross Entropy as its usually used for classification problems. Each epoch traverses over all the training batches and thereby the entire training set. Every

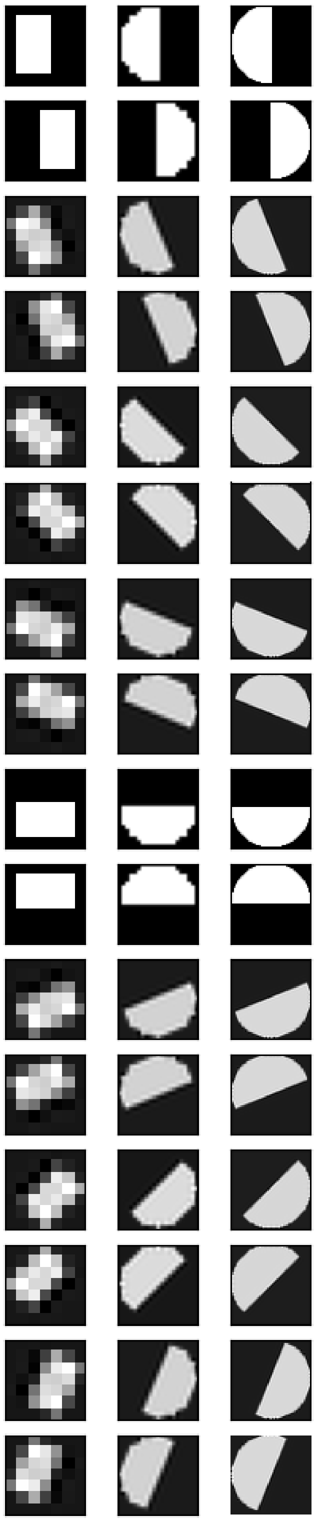

Fig. 5. Half Disk Mask

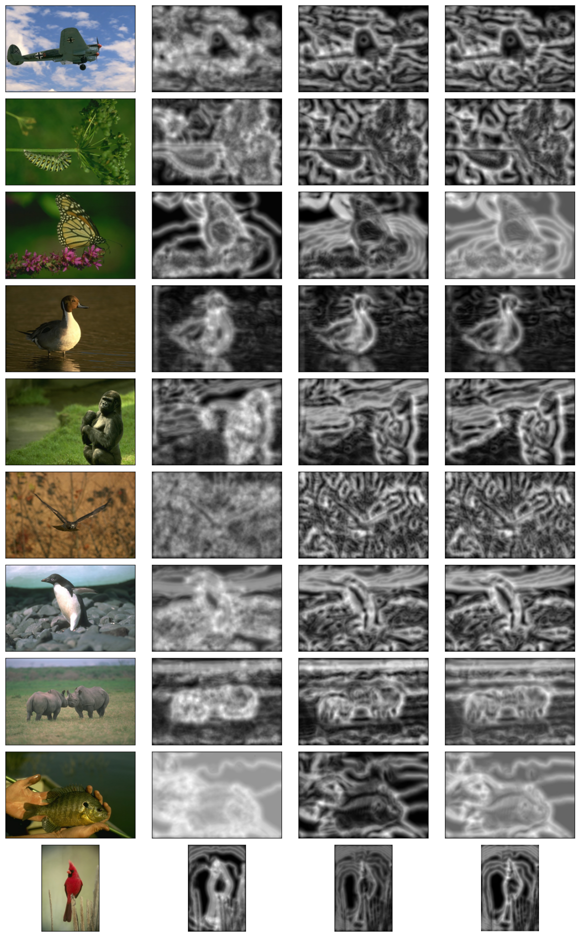

Fig. 6. Original Image, Texture, Brightness, Color Gradients (Left to Right)

Learning Rate	1e-3
Batch Size	4096
Parameter Count	0.326M
Optimizer	Adam
Loss function	Cross Entropy Loss
Train Epochs	100

TABLE I
BASELINE HYPER PARAMETERS

epoch, training loss, testing loss, training accuracy and testing accuracy were logged. Inputs were pre-normalized by the given training pipeline. Check point files were stored during every epoch and the one with best accuracy was chosen. Since the input image sizes were quite small and the machine had 24 GB VRAM, Batch size was increased from 32 to 4096. A complete list of training parameters can be seen in table I

3) *Results and Observations::* During the training process, the training loss kept consistently reducing over epochs as shown in figure 12. Training accuracy kept going up as shown in figure 10. But the test accuracy started dropping after 17th epoch when the model started overfitting as seen in figure 11. The 17th epoch model was chosen to evaluate the confusion matrix (figure 13). Test confusion matrix has higher values along the diagonal and lower off-diagonal elements indicating

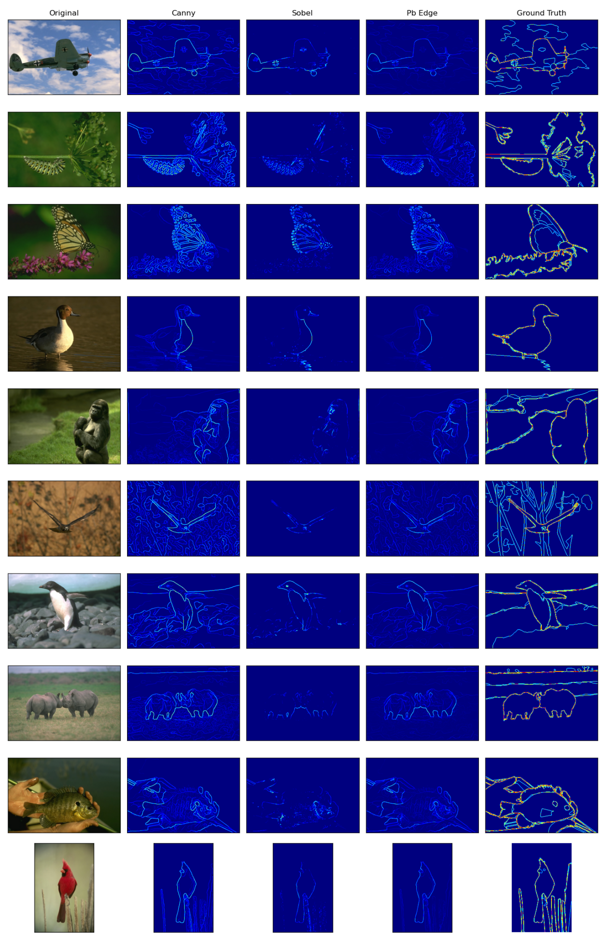

Fig. 7. Edges (Original, Canny, Sobel, Pb, Ground truth - left to right)

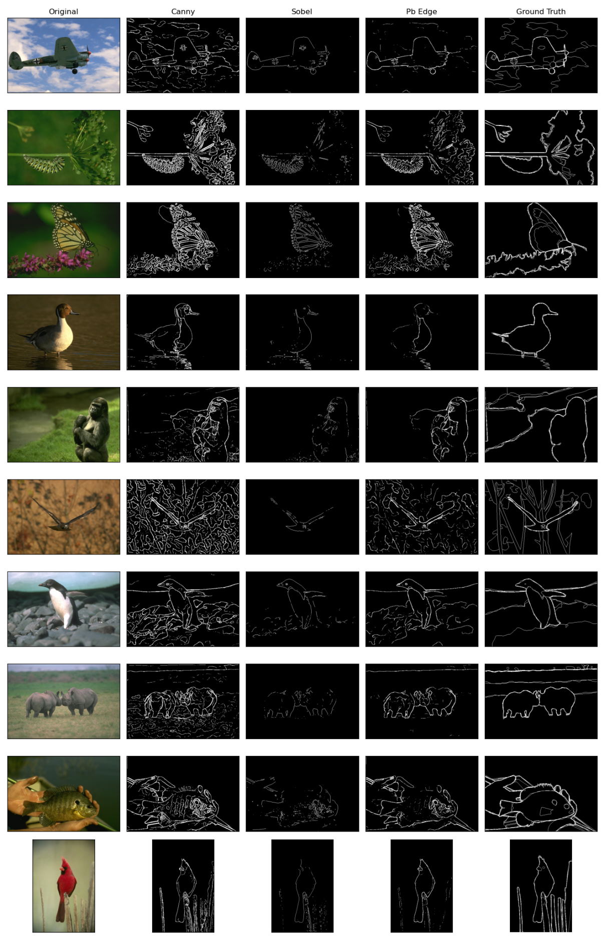

Fig. 8. Binary Edges (Original, Canny, Sobel, Pb, Ground truth - left to right)

good testing accuracy. The testing accuracy is at 48.16% for the baseline implementation for the entire test-set. As expected train confusion matrix has high values on diagonal elements (figure 14)

C. *Improving Accuracy of your neural network*

1) Increasing the layer count: Instead of using 6 Resnet blocks, 11 Resnet blocks were used in this experiment. Additionally, fully connected layer count was increased to 4 from 3. This increased the model parameter count to 305M. Other parameters were not changed.
Train accuracy of 94% and test accuracy of 48% was obtained in this case. Even with a massive model, these was no increase in test accuracy.

2) Dropout layers: Now dropout layers were added to improve the generalisation of the network. These were added to the baseline network. With this change, the test accuracy also consistently increased (figure 15). I was able to achieve a max test accuracy of 49.23%.

3) Analysis and comparison across experiments:

- As illustrated in table II, 48% accuracy was obtained using vanilla Resnet framework.

Metric	Baseline	Improved 1	Improved 2
Architecture	Resnet6	Resnet11	Resnet6
Layer Count	6	11	6
Parameter Count	0.326M	305M	0.326M
BatchNorm	Yes	Yes	Yes
DropOut	No	No	Yes
Test Accuracy	48.5%	48%	49.2%

TABLE II
CIFAR10 NETWORK EXPERIMENTS

- Increasing the network size blindly does not result in increased accuracy.
- Introducing dropout seems to make the training generalize better as seen in figure 15.
- Increasing the batchsize, seems to increase GPU memory but also increase the speed of training via massive parallelism.

III. CONCLUSION

As illustrated above, edge detection was performed using the probability of boundaries algorithm. The algorithm achieved better performance than canny and sobel filters. Additionally in the second half of this work, deep neural networks were trained to classify images in CIFAR10 dataset.

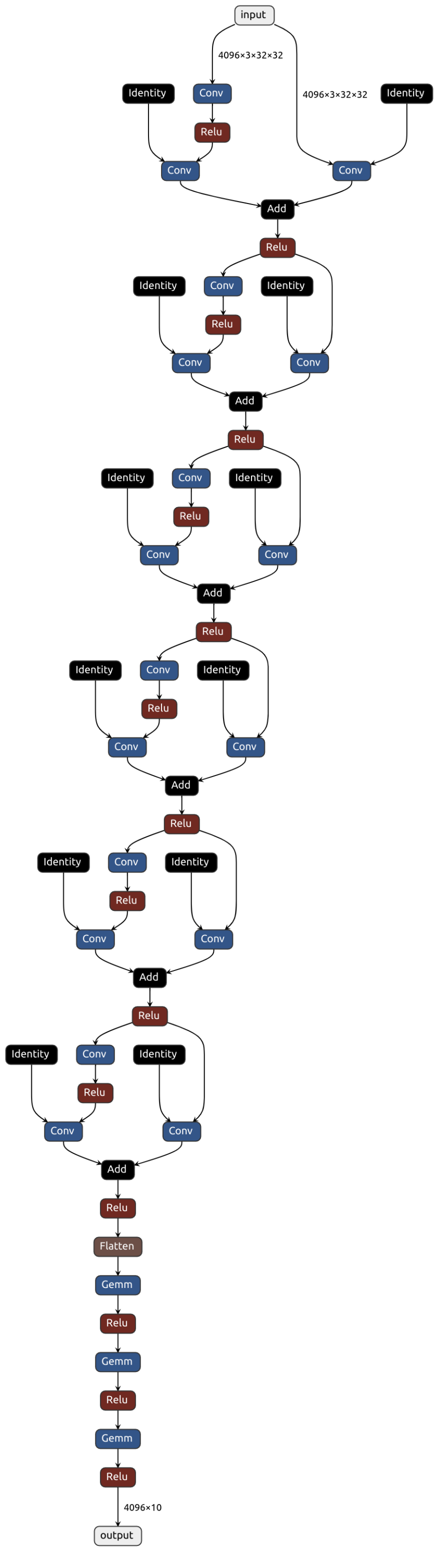

Fig. 9. Baseline - Resnet architecture

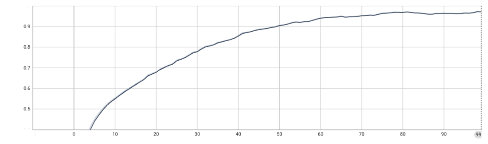

Fig. 10. Baseline - Train epoch-accuracy vs epoch count

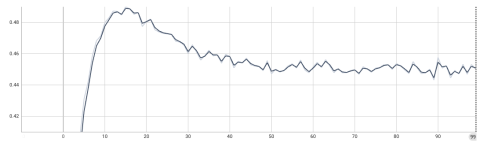

Fig. 11. Baseline - Test epoch-accuracy vs epoch count

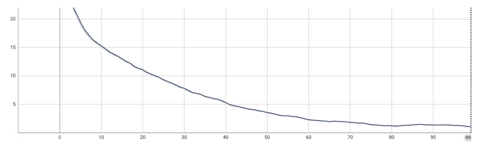

Fig. 12. Baseline - Train epoch-loss vs epoch count

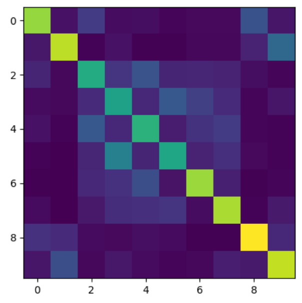

Fig. 13. Baseline - Test Confusion Matrix

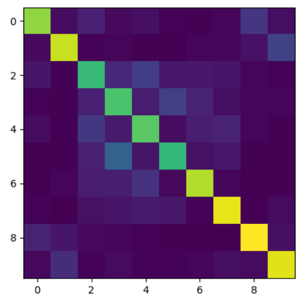

Fig. 14. Baseline - Train Confusion Matrix

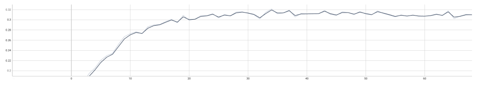

Fig. 15. Improved 2 - Testing accuracy over epochs