CMSC733: Homework 0 - Alohomora!

Abubakar Siddiq Palli
absiddiq@umd.edu
Using 1 late day

I. Phase 1: Shake my Boundary

In Phase 1, the objective is to redefine the approach to boundary detection within images, a task critical to the field of computer vision. This phase is focused on the development of a Pb-lite algorithm, a simplified adaptation of the more complex probability of boundary (pb) algorithms. By integrating analysis of brightness, color, and texture across various image scales, the algorithm aims to accurately identify the transition points between different objects or areas within an image. The methodology is straightforward yet effective, comprising the *creation of a filter bank, computation of texton, brightness, and color maps,* followed by the *generation of gradient maps for texture, brightness, and color,* culminating in the detection of boundaries. This simplified approach is designed to enhance the accuracy of boundary detection, surpassing traditional methods like the Canny and Sobel edge detectors, through a meticulous examination of the image's features.

A. Generating the Filter Banks

Three filter banks are created, each designed to capture distinct texture features within images.

1. Oriented DoG Filters: Utilizing the oriented Derivative of Gaussian (DoG) filters (*Fig 1*), we created a versatile set for capturing edge orientations and scales by convolving Sobel and Gaussian filters. Convolving with the Sobel operator essentially performs differentiation on the image which was smoothed using a Gaussian filter.

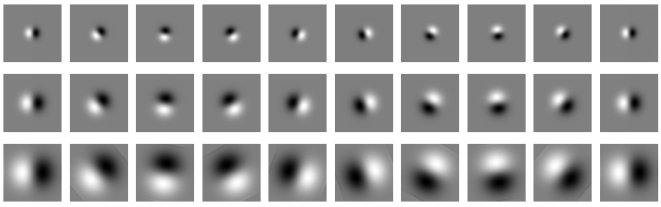

Fig. 1: Oriented DOG Filters

2. Leung-Malik Filters: The Leung-Malik (LM) filter bank (*Fig 2*), with 48 multi-scale, multi-orientation filters, was implemented in two variations, LM Small (LMS) and LM Large (LML), to capture a wide range of texture details across different scales and orientations.

3. Gabor Filters: Inspired by the human visual system, Gabor filters (*Fig 3*) are crafted to detect texture patterns through a combination of Gaussian kernels and multi-dimensional(2D) sinusoidal waves, mirroring the way humans perceive spatial frequencies and orientations.

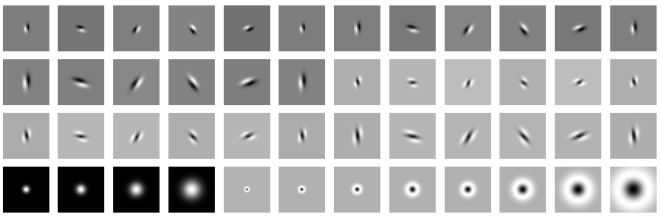

Fig. 2: Leung-Malik Filters

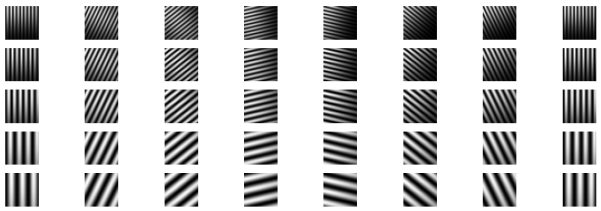

Fig. 3: Gabor Filters

These filter banks were applied to the image to produce a texton map, effectively clustering filter responses to highlight texture patterns. This step is crucial for the texture, brightness, and color analysis that follows, setting a solid groundwork for our boundary detection algorithm.

B. Texton, Brightness and Color maps computation

For the computation of the texton map, we begin by applying each filter from our assembled filter bank to the input image. Assuming the presence of N filters in our bank, this process yields N distinct responses for every pixel, effectively transforming each pixel into an N-dimensional vector. Following this, we employ KMeans clustering on these vectors for the entire image, assigning a unique texton ID to each pixel. Substituting each pixel with its respective texton ID results in the creation of the texton map, a representation that encapsulates the textural characteristics of the image at the pixel level. Similarly, Brightness map is generated by clustering the grayscale image and the color map is generated by clustering each color channels in the image. *Figures 4-13* are *Texton (T), Brightness (B) and Color (C)* Maps for input images.

Fig. 4: *T, B, C of image 1*

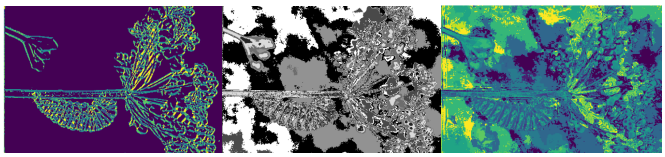

Fig. 5: *T, B, C of image 2*

Fig. 6: *T, B, C of image 3*

Fig. 7: *T, B, C of image 4*

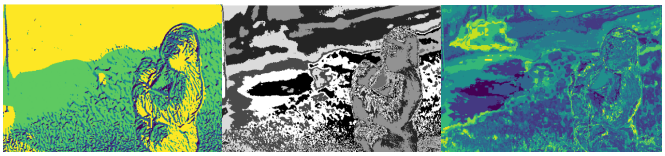

Fig. 8: *T, B, C of image 5*

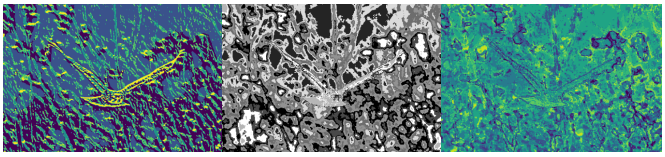

Fig. 9: *T, B, C of image 6*

Fig. 10: *T, B, C of image 7*

Fig. 11: *T, B, C of image 8*

Fig. 12: *T, B, C of image 9*

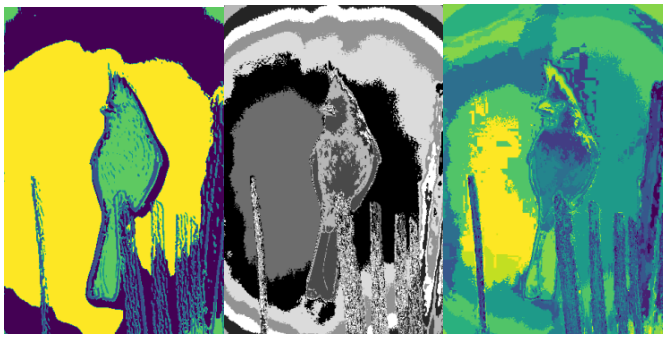

Fig. 13: *T, B, C of image 10*

C. Computing Gradient maps of Texture, Brightness, and Color

This section will delve into the methodology for calculating gradient maps based on the texture, brightness, and color information extracted from the image. The aim here is to quantify changes or discontinuities in these features across the image, which are indicative of potential boundaries between different regions or objects. Gradient maps serve as a crucial component for the subsequent boundary detection process, as they highlight areas with significant changes in texture, luminance, and color, providing a detailed understanding of the image's structural and compositional elements. To compute these maps half-disc maps are utilized. These masks *Fig 14* are binary images designed as pairs of semi-circles, enabling the efficient computation of X^2(chi-square) distances across the image. This approach significantly optimizes the process by replacing the traditional, computationally intensive method of iterating over pixel neighborhoods with a faster, filtering operation. Consequently, this technique allows us to rapidly generate gradient values (T_g *for texture,* B_g *for brightness, and* C_g *for color*). *Figures 15-24* are Gradient maps of Texture, Brightness, and Color for input images.

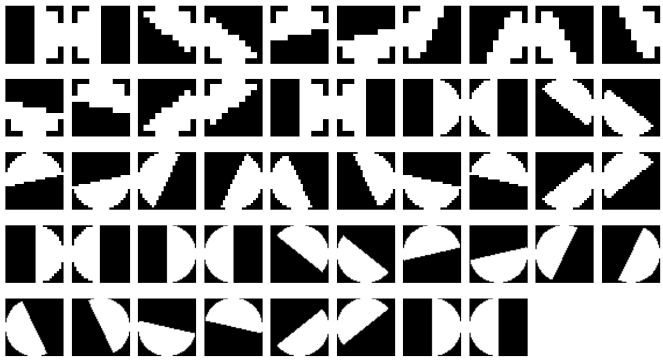

Fig. 14: Half discs masks

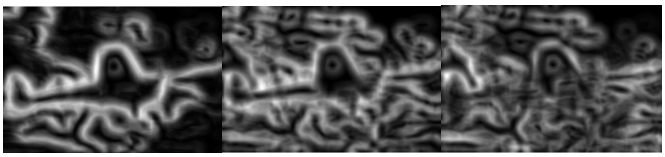

Fig. 15: T_g, B_g, C_g *of image 1*

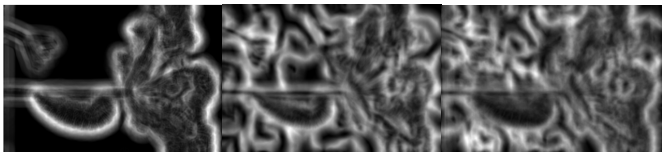

Fig. 16: T_g, B_g, C_g *of image 2*

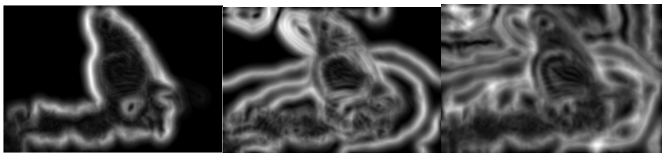

Fig. 17: T_g, B_g, C_g*of image 3*

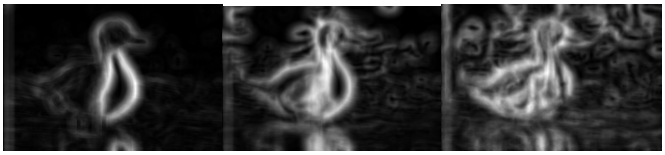

Fig. 18: T_g, B_g, C_g *of image 4*

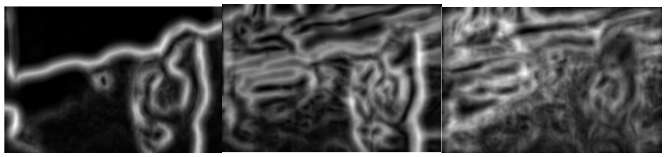

Fig. 19: T_g, B_g, C_g *of image 5*

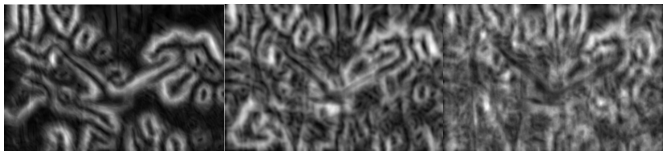

Fig. 20: T_g, B_g, C_g *of image 6*

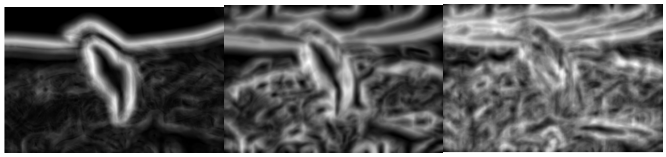

Fig. 21: T_g, B_g, C_g *of image 7*

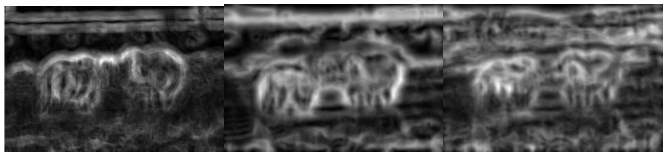

Fig. 22: T_g, B_g, C_g*of image 8*

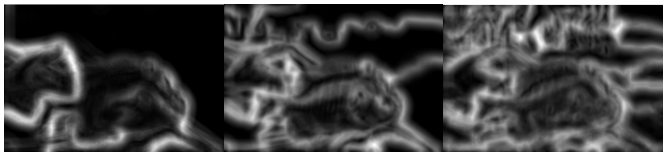

Fig. 23: T_g, B_g, C_g *of image 9*

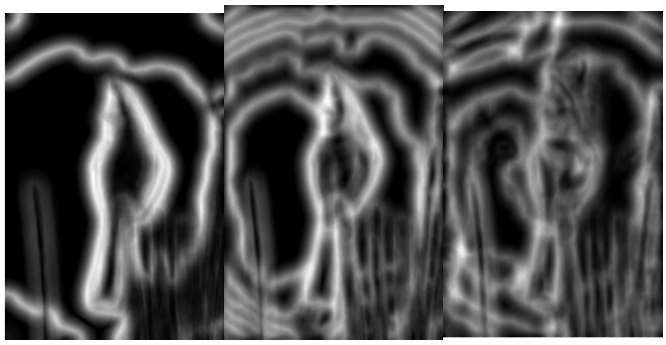

Fig. 24: T_g, B_g, C_g *of image 10*

D. Boundary detection - Pb-Lite

Now, Finally for the boundary detection we use the computed gradient maps for *texture (T_g), brightness (B_g), and color (C_g)* along with *Canny and Sobel* baselines to detect boundaries within the image. The Pb-Lite algorithm is as follows. w_1 = *0.9*, w_2 = *0.1*

$$\text{PbEdges} = \frac{T_g + B_g + C_g}{3} * w_1 * \text{Canny} + w_2 * \text{Sobel}$$

The result of the boundary of all the images are from *Figures 25-34*

Fig. 25: Pb-Lite result for image 1

Fig. 26: Pb-Lite result for image 2

Fig. 27: Pb-Lite result for image 3

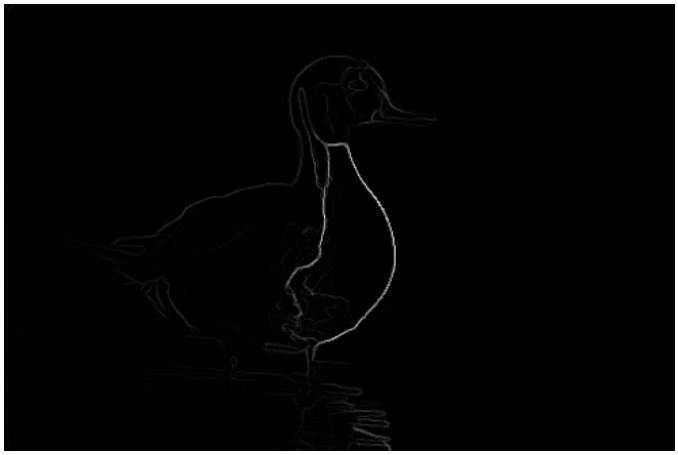

Fig. 28: Pb-Lite result for image 4

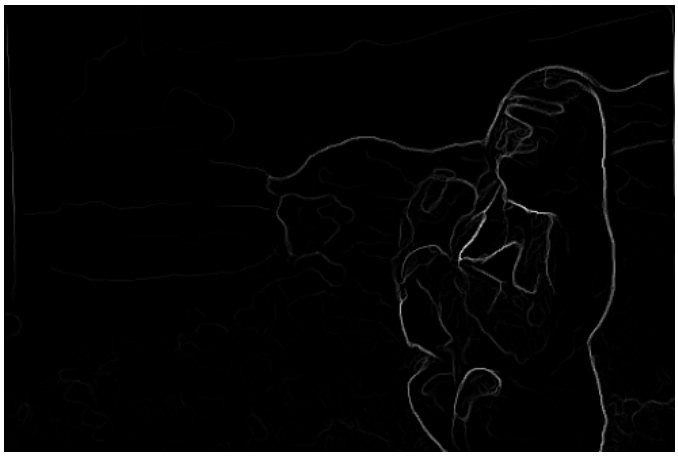

Fig. 29: Pb-Lite result for image 5

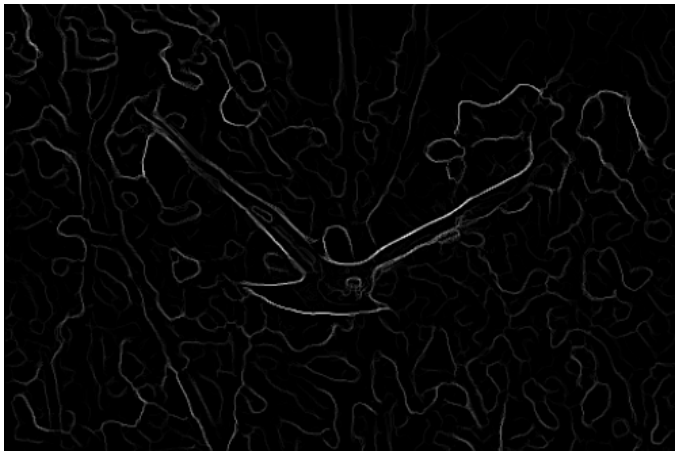

Fig. 30: Pb-Lite result for image 6

Fig. 31: Pb-Lite result for image 7

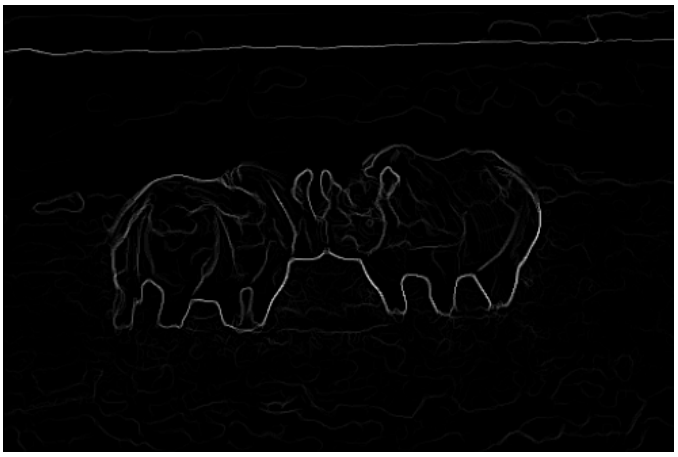

Fig. 32: Pb-Lite result for image 8

Fig. 33: Pb-Lite result for image 9

Fig. 34: Pb-Lite result for image 10

II. Phase 2: Deep Learning

In Phase 2 we implemented and did a comparative analysis of multiple neural network architectures, focusing on key performance metrics such as the number of parameters, as well as training and testing accuracies. Training is done on the CIFAR-10 dataset, comprising 60,000 color images across 10 classes, with a standard split of 50,000 training and 10,000 test images.

The architectures implemented are a basic Convolutional Neural Network (CNN) model, and more advanced models such as ResNet, ResNeXt, and DenseNet. Each of these models brings a unique approach to dealing with the complexities of image data, from ResNet's residual learning framework to ResNeXt's aggregated transformations and DenseNet's feature reuse strategy.

A. Simple CNN Model

This is a simple CNN model with two convolutional layers and three fully connected layers and Max pooling was used for dimensionality reduction. Training accuracy of 96.68% was reached at the 10th epoch and 72.91% accuracy on the

test dataset. The model has 1702794 trainable parameters. In the *Figure 35* Training accuracy and loss over the epoches is plotted and the Confusion matrix for the Training and Test sets is shown in *Figure 36 and 37*

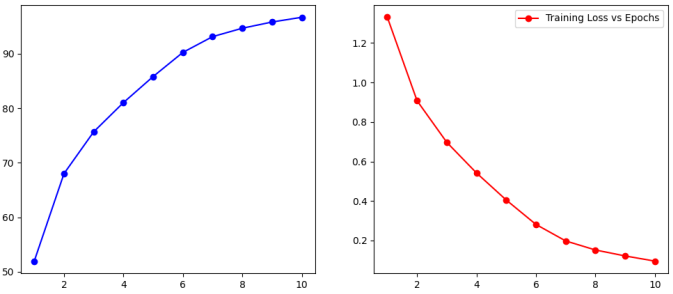

Fig. 35: Simple CNN model Training accuracy and Loss vs Epochs

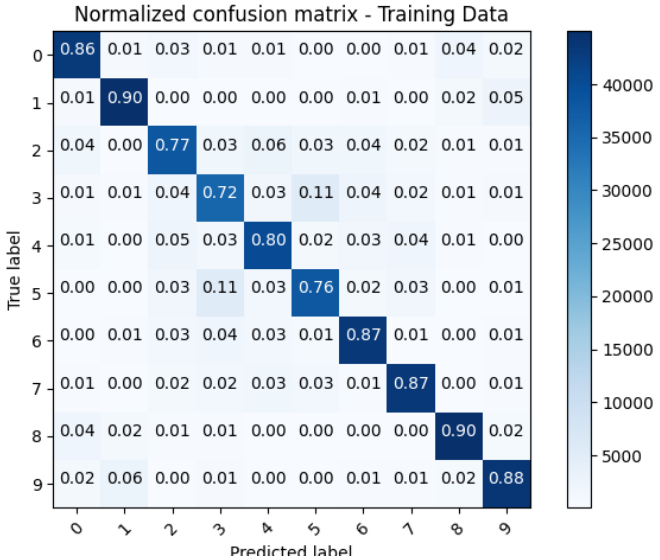

Fig. 36: Simple CNN model Confusion Matrix on Training Set

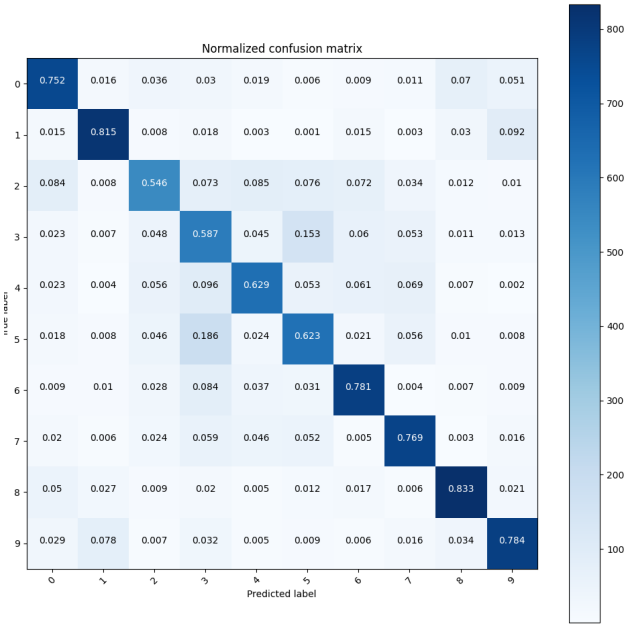

Fig. 37: Simple CNN model Confusion Matrix on Test Set

B. ResNet

The ResNet architecture utilizes the concept of residual learning to facilitate the training of deeper networks by addressing the vanishing gradient problem. The simplified architecture is built around the BasicBlock class, a fundamental unit that includes two convolutional layers. Each BasicBlock employs a sequence of operations: a convolution (with kernel size 3), batch normalization, and Rectified Linear Unit(ReLU) activation function, followed by another similar sequence. The architecture includes a shortcut connection that either directly passes the input to the output.

The ResNet class builds the complete network by initializing with a single convolutional layer (to process the initial input), followed by batch normalization and the creation of four main layers. Each of these layers consists of multiple BasicBlock instances, with the number of blocks per layer specified. The network adapts to increasing feature dimensions by doubling the number of output channels at each subsequent layer (64, 128, 256, 512) and applying a stride of 2 to reduce the spatial dimensions when transitioning between layers, except for the first layer.

The final output is produced by applying an average pooling operation to reduce spatial dimensions to a single value per feature map, flattening the result, and passing it through a fully connected layer to map to the desired class. This design allows ResNet to learn identity mappings efficiently, enabling the training of deep networks by effectively propagating gradients. Training accuracy of 95.43% was reached at the 10th epoch and 82.69% accuracy on the test dataset. The model has 11173962 trainable parameters.

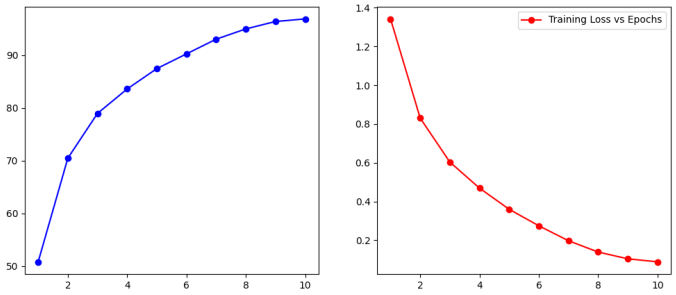

Fig. 38: ResNet model Training accuracy and Loss vs Epochs

C. ResNeXt

The ResNeXt model introduces a approach to feature aggregation and convolutional processing by incorporating

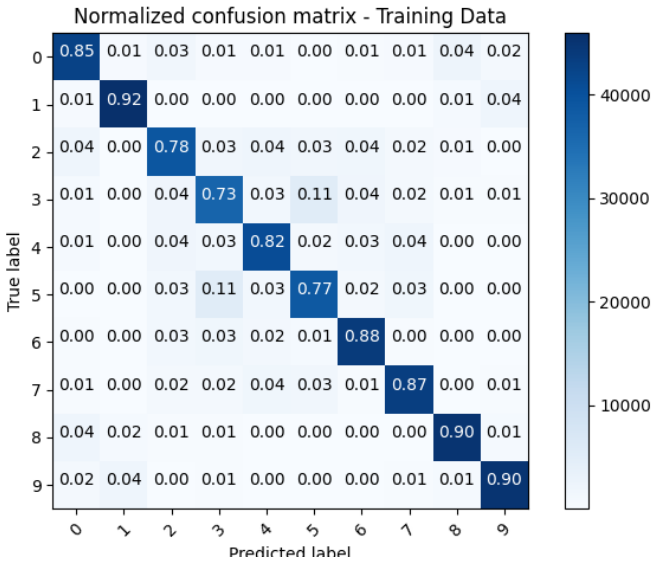

Fig. 39: ResNet model Confusion Matrix on Training Set

Fig. 40: ResNet model Confusion Matrix on Test Set

the concept of cardinality, which refers to the number of parallel paths within the network. The architecture is built around the ResNeXtBlock, a building block design with three convolutional layers: a 1x1 convolution for dimensionality reduction, a 3x3 grouped convolution for processing within subspaces, and another 1x1 convolution to expand the feature dimensionality, facilitating a deep yet computationally efficient structure. Each ResNeXtBlock begins with a 1x1 convolution that reduces the input channel size to manage computational complexity. It is followed by a 3x3 convolution with groups equal to the cardinality parameter, allowing the model to learn more diverse representations by processing inputs in separate subspaces. The final 1x1 convolution in the block aims to expand the feature maps in preparation for addition to the shortcut connection.

The ResNeXt class orchestrates the overall network structure, initiating with a standard 64-channel 3x3 convolutional layer, followed by batch normalization and then sequentially stacking ResNeXtBlock layers. These layers are constructed with increasing channel sizes (128, 256, 512) and are arranged to progressively refine the feature maps while reducing their spatial dimensions. Training accuracy of 95.98% was reached at the 10th epoch and 86.49% accuracy on the test dataset. The model has 8468298 trainable parameters.

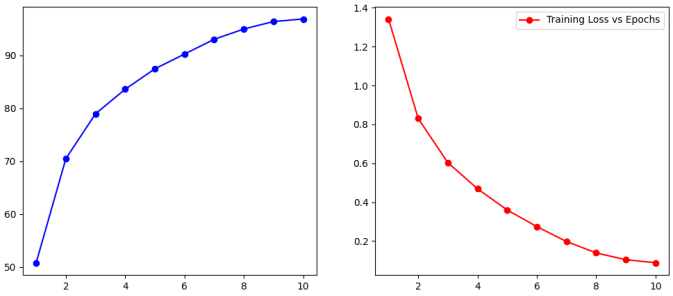

Fig. 41: ResNeXt model Training accuracy and Loss vs Epochs

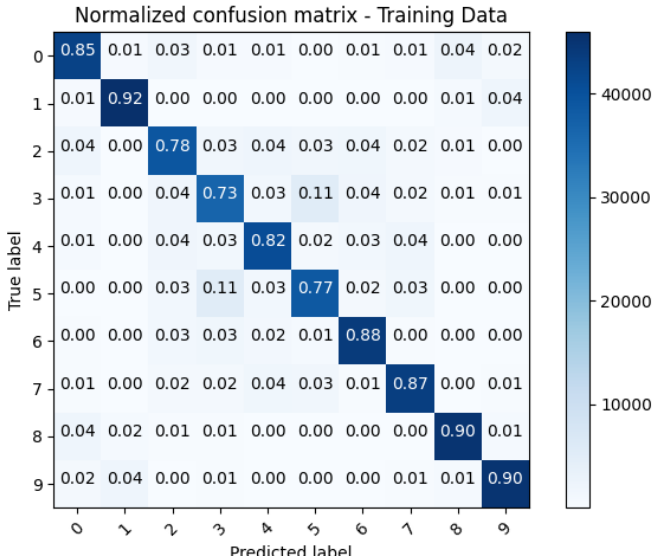

Fig. 42: ResNeXt model Confusion Matrix on Training Set

D. DenseNet

The DenseNet architecture introduces a highly efficient approach to deep learning models through its use of dense connections. At the heart of DenseNet is the ‘DenseLayer‘, designed to connect each layer to every other layer in a feed-forward fashion. Each ‘DenseLayer‘ applies batch normalization, followed by a ReLU activation and a 3x3 convolution, with the output feature maps being concatenated with the input feature maps, thereby promoting feature reuse and substantially reducing the number of parameters.

Fig. 43: ResNeXt model Confusion Matrix on Test Set

DenseNet's architecture is organized into multiple 'DenseBlocks' and 'TransitionLayers'. A 'DenseBlock' comprises several 'DenseLayers' that enhance the network's depth and feature-learning capability without a significant increase in complexity. The growth rate parameter controls the increase in the number of feature maps, thereby managing the computational efficiency. Following each 'DenseBlock', except the last one, is a 'TransitionLayer', which applies batch normalization, a 1x1 convolution, and an average pooling operation to reduce the size of the feature maps and prepare the network for the next 'DenseBlock'. This design allows DenseNet to maintain a compact and efficient model architecture. This particular DenseNet configuration, with its carefully chosen growth rate and block configuration, is tailored for the CIFAR-10 dataset, aiming to achieve high classification accuracy with a relatively low computational burden. Training accuracy of 84.86% was reached at the 10th epoch and 74.42% accuracy on the test dataset. The model has 1406506 trainable parameters.

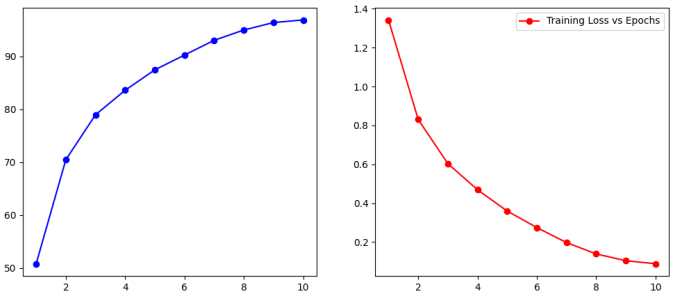

Fig. 44: DenseNet model Training accuracy and Loss vs Epochs

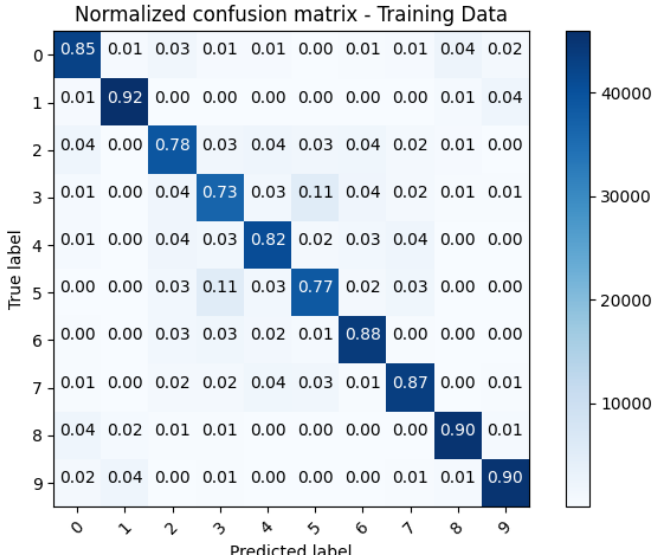

Fig. 45: DenseNet model Confusion Matrix on Training Set

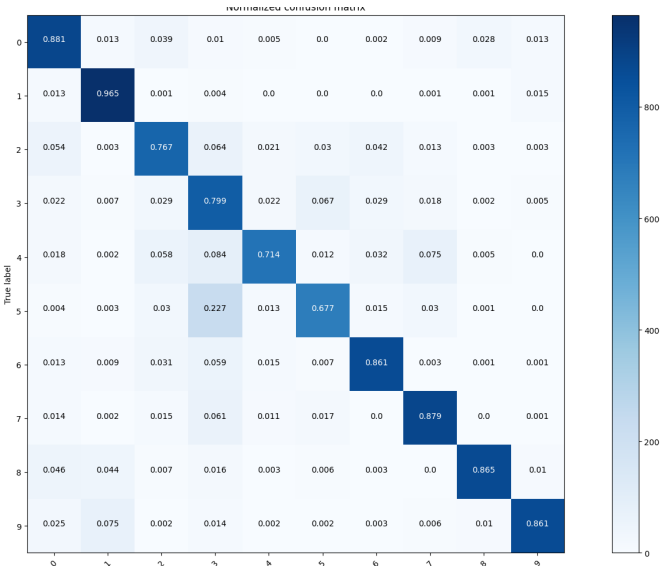

Fig. 46: DenseNet model Confusion Matrix on Test Set