HW0: Alohomora

Mayank Bansal
Robotics Engineering
Worcester Polytechnic Institute
Email: mbansal1@wpi.edu

I. Phase1: Shake My Boundary

The aim here is to develop the pb-lite boundary detection algorithm, where 'pb' stands for Probability of Boundary. This terminology is used as the algorithm estimates the chance of each pixel in an image being a boundary. This approach not only looks at the gradients in intensity but also examines the variations in texture and color to identify edges. The process is divided into four main steps:

- Generation of filter bank.
- Generation of texton, brightness and color maps.
- Generation of texton, brightness and color gradient maps.
- Boundary detection using the maps, Sobel and Canny baselines.

A. Generation of filter banks

In order to get the texture information from the images, a number of different filters, collected into a filter bank is applied on to the images. The filters applied may be of different scales and orientations. Three main kinds of filters are used in this phase:

1) Oriented Derivative of Gaussian (DoG) filters: These filters, which are Derivative of Gaussian filters in various orientations, are created by applying a Sobel operator to a Gaussian kernel. The set includes Oriented DoG filters of 2 scales and 16 orientations, as illustrated in figure 1.
2) Leung-Malik Filters: This filter set contains 48 filters of multiple scales and orientations, including 36 first and second order derivatives of Gaussians in 6 orientations and 3 scales, 8 Laplacian of Gaussian (LoG) filters, and 4 Gaussians. For this experiment, two versions of Leung-Malik filter banks, namely LM small and LM large, were created. Figures 2 and 3 show the LM small and LM large filter banks, respectively.
3) Gabor filters: Inspired by the functioning of the human eye, these filters consist of a Gaussian kernel modulated by a sinusoidal wave. The Gabor filter bank created for this experiment is depicted in figure 4.

B. Texton, Brightness and Color Maps

After generating filter banks, each filter is applied to the input images, producing a vector of filter responses at each pixel. This vector encodes the texture properties of the image. With N filters in the bank, the output is an N-dimensional vector per pixel.

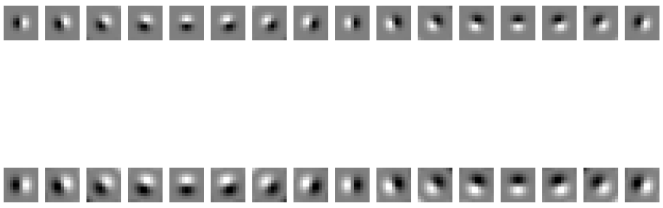

Fig. 1. Oriented Derivative of Gaussian Filter Bank.

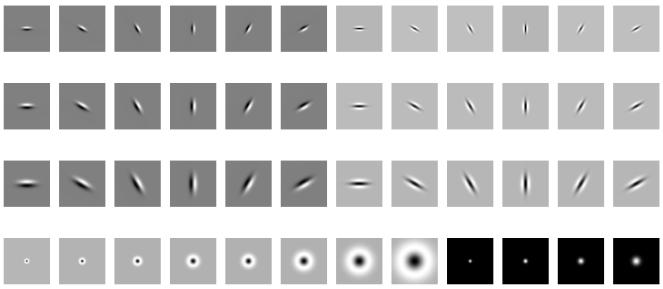

Fig. 2. Small Leung-Malik Filter Bank

Next, these N-dimensional vectors are converted into discrete texton IDs. This is achieved by clustering the filter responses into K textons using KMeans clustering, resulting in a Texton map (T) with values ranging from 1 to K. In this experiment, K is set to 64.

Similarly, Brightness (B) and Color maps (C) are created. The Brightness map clusters grayscale intensity values, while the Color map uses the three-channel color data.

C. Texton, Brightness and Color Gradient Maps

The texton, brightness, and color maps of the input images were created to identify gradients, thereby revealing areas where changes in texture, intensity, and color occur. Gradients are determined using half-disc masks, as illustrated in figure 5. These masks, binary images of half-discs in pairs, simplify the process of calculating χ^2 distances through straightforward filtering operations. This approach replaces the more complex method of accumulating histogram counts across pixel neighborhoods. For this study, half-disc masks were generated in 8 orientations and 3 scales. The texton, brightness, and color maps are filtered using the masks. Then, the χ^2 distances

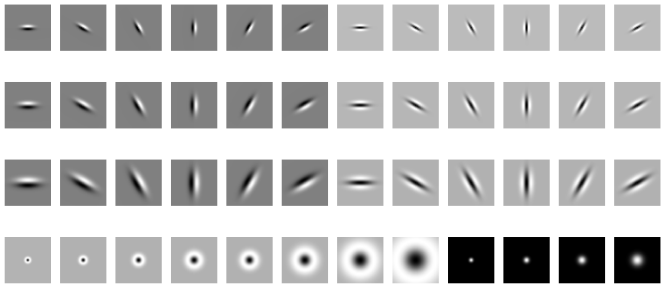

Fig. 3. Large Leung-Malik Filter Bank

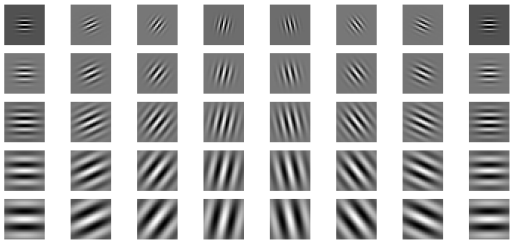

Fig. 4. Gabor Filter Bank

between two histograms g and h can be obtained by using the equation shown below:

$$\chi^2 = \frac{1}{2}\sum_{i=1}^{K}\frac{(g_i - h_i)^2}{g_i + h_i}$$

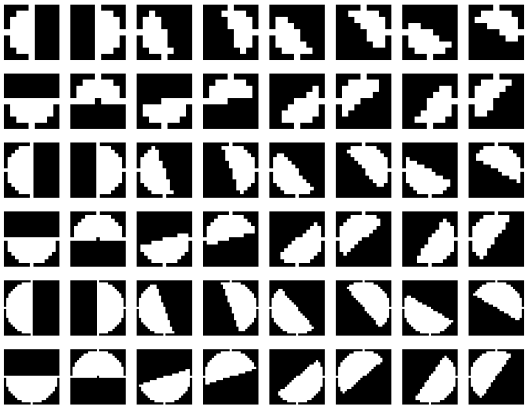

Fig. 5. Half-disc masks

D. Boundary Detection

Before proceeding to generating the pb-lite output, we must read all the Sobel and Canny baselines. These baselines can be used to generate the pb-lite output with the given equation:

$$PbEdges = \frac{T_a + B_b + C_c}{3} \cdot (w_1 \cdot \text{cannyPb} + w_2 \cdot \text{sobelPb})$$

II. Phase 2: Deep Dive on Deep Learning

In this stage, my goal is to apply various neural network architectures for classifying images from the CIFAR-10 dataset, which comprises 60,000 images (50,000 for training and 10,000 for testing) each with a resolution of 32x32 pixels, distributed across 10 different categories. Throughout the experiment, I have trained five different models, namely:

1) A Basic Neural Network designed by me.
2) An Advanced Neural Network of my own design.
3) Resnet.
4) ResNeXt.
5) DenseNet.

A. Basic Neural Network

The network consists of two convolutional layers, both of which are followed by ReLU activation functions to introduce non-linearity. After each ReLU, there is a max-pooling layer that reduces the spatial dimensions by taking the maximum value over a 2x2 window. The output from the convolutional layers is then flattened into a one-dimensional vector. This flattened vector is fed into a fully connected layer with an output size of 100, followed by another ReLU activation. The final output is produced by a fully connected layer that maps to the number of classes (OutputSize). This final layer would be used to determine the predicted class based on the highest output value.

For this experiment, I selected a batch size of 64 and the optimizer selected was AdamW with a learning rate of 1e-3. I also selected Cross Entropy loss for training the network. The training results are shown in figures 6 through 9.training

B. Modified Basic Neural Network

This model is a refined neural network designed for image classification, featuring two key convolutional layers to capture the input image's features. Each convolutional layer is enhanced with batch normalization, which standardizes the activations to aid in faster and more stable training. Non-linear transformation is introduced after each batch normalization via ReLU activation functions, allowing the network to handle complex patterns.

Spatial feature reduction is achieved with max-pooling layers following each ReLU, which also helps in making the detection of features somewhat invariant to scale and orientation. After the convolutional stages, the network flattens the data, preparing it for the dense layers.

The dense part of the network begins with a fully connected layer of 100 neurons, which is batch-normalized and passed through another ReLU activation. To combat overfitting, a dropout layer with a 50

Fig. 6. Architecture for BasicNet

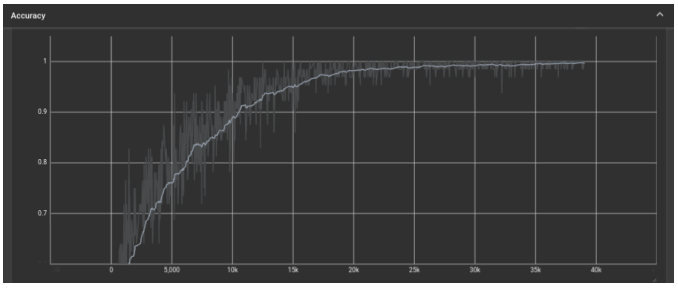

Fig. 7. Train Accuracy for BasicNet

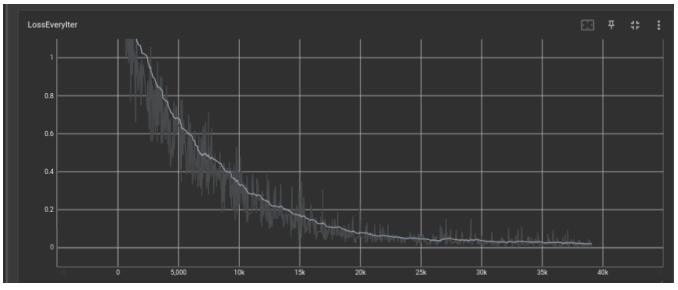

Fig. 8. Train Loss for BasicNet

```
Number of parameters in this model are 8
  0%|          | 0/10000 [00:00<?, ?it/s]
[742  20  52  12   7   8   5  79  36  39] (0)
[120 599   8  23   5   4   3  19  56 163] (1)
[192  18 371 118  12  70  20 125  25  49] (2)
[103  20  94 392  22 108  33 117  52  59] (3)
[151  17 154 159  98  83  44 211  52  31] (4)
[ 74  19  82 220   8 365  29 132  28  43] (5)
[ 64  41  98 199  23  59 314  66  88  48] (6)
[ 75  13  51  58   8  57  10 662  20  46] (7)
[325  31  44  16   4   5   1  98 373 103] (8)
[124  98  20  24   0   9   2  57  30 636] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 45.52 %
```

Fig. 9. Confusion matrix after training for BasicNet

The final layer is a fully connected layer with a neuron count equal to the number of classes (denoted by 'OutputSize'), which maps the learned features to class scores for classification. The enhanced network is designed to learn robustly with improved generalization capabilities over the basic network.

For this experiment, I selected a batch size of 64 and the optimizer selected was AdamW with a learning rate of 1e-3. I also selected Cross Entropy loss for training the network. The training results are shown in figures 10 through 13.

C. ResNet

This neural network architecture is the ResNet model, which is particularly known for its ability to efficiently train deeper networks. It starts with an initial convolutional layer that processes the input image, followed by batch normalization and a ReLU activation to introduce non-linearity.

The core of the network consists of a series of building blocks, each containing two convolutional layers. Each convolutional layer is followed by batch normalization and ReLU activation. These blocks have a distinctive feature: a shortcut connection that bypasses the two convolutional layers, directly adding the input of the block to its output. This design helps in preventing the vanishing gradient problem, allowing for deeper network architectures by ensuring that the gradient can flow through the network without diminishing.

The network then applies an adaptive average pooling layer, which adapts the output size to a consistent shape, regardless of the input size. This is particularly useful for connecting convolutional layers to fully connected layers, as it ensures that the fully connected layer always receives the same amount of input features.

Finally, the network concludes with a fully connected layer that maps the learned feature representations to the desired number of output classes. This final layer is what determines the classification output based on the features extracted and processed through the network.

For this experiment, I selected a batch size of 64 and the optimizer selected was Adam with a learning rate of 1e-3. I

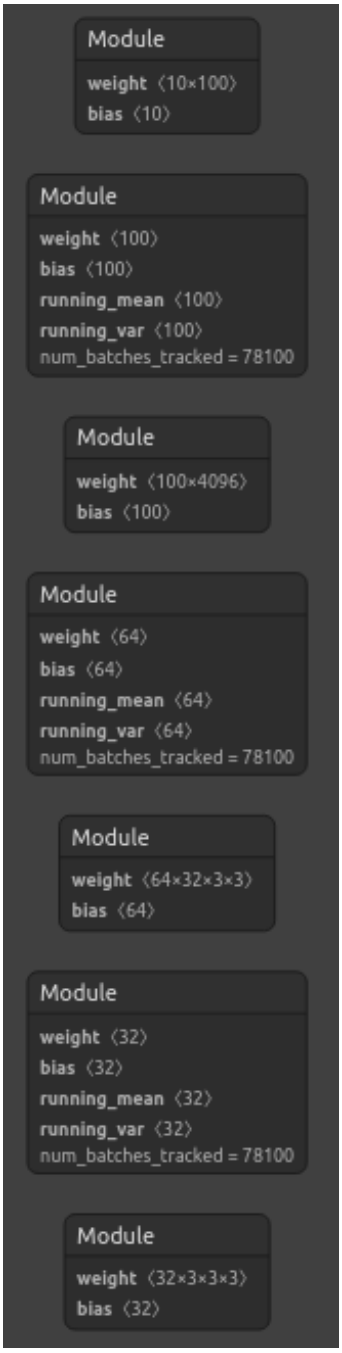

Fig. 10. Architecture for Modified BasicNet

also selected Cross Entropy loss for training the network. The training results are shown in figures 14 through 17.

D. ResNeXt

This neural network is the ResNeXt architecture, streamlined for image classification tasks. It commences with an initial convolutional layer to process the input image, reducing its spatial dimensionality. This layer is followed by batch normalization and a ReLU activation function to introduce non-linearity.

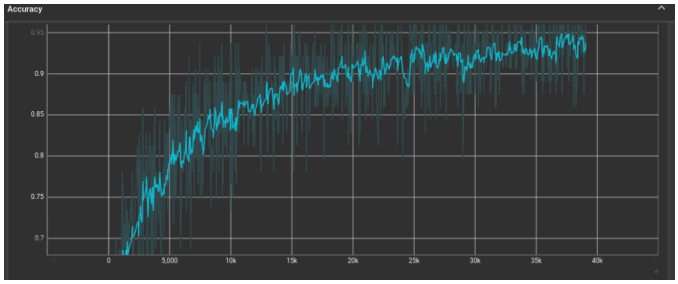

Fig. 11. Train Accuracy for Modified BasicNet

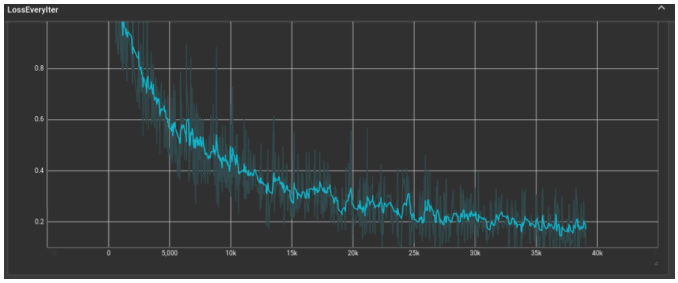

Fig. 12. Train Loss for Modified BasicNet

At its core, the network consists of several sequential blocks. Each block includes a convolutional layer, batch normalization, and a shortcut connection, similar to those found in ResNet architectures. The shortcut connection helps prevent the vanishing gradient problem by allowing gradients to flow more easily through the network. If the input and output channels differ in number or a stride is applied, the shortcut connection includes its own convolutional layer to match the dimensions.

The architecture features multiple layers, each comprising a series of these blocks. These layers are designed to progressively increase the number of channels while reducing the spatial dimensions of the feature maps. This structure

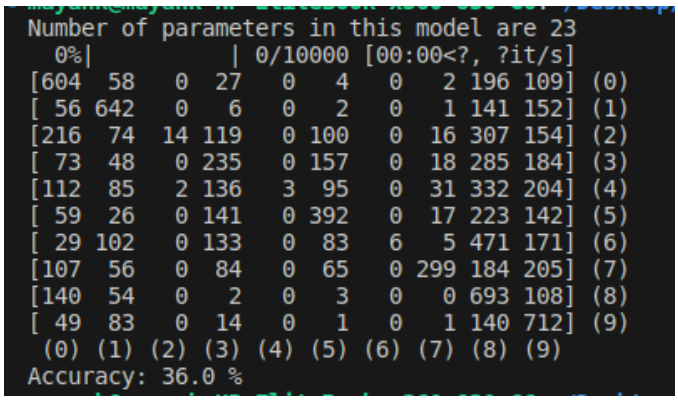

Fig. 13. Confusion matrix after training for Modified BasicNet

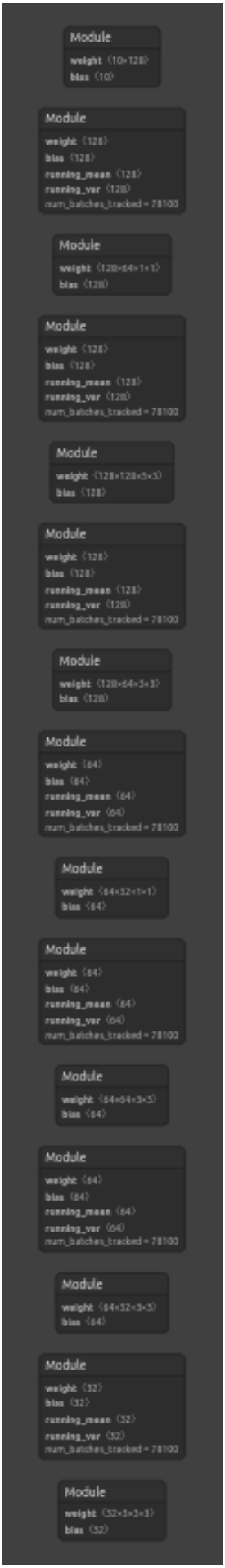

Fig. 14. Architecture for ResNet

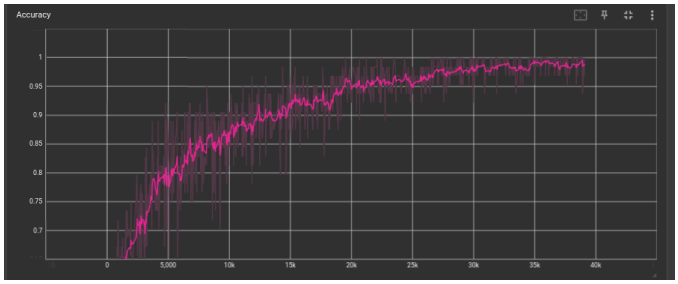

Fig. 15. Train Accuracy for ResNet

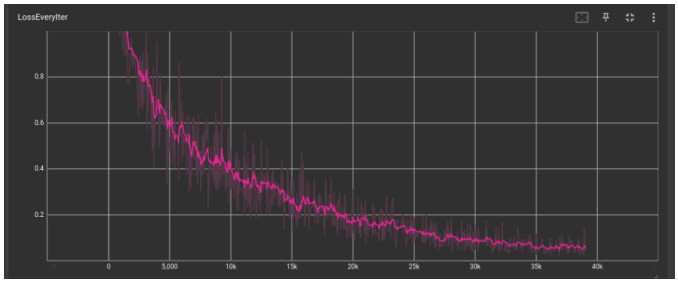

Fig. 16. Train Loss for ResNet

allows the network to learn increasingly complex and abstract representations of the input data.

After passing through these layers, the network employs an adaptive average pooling layer, which reduces the spatial dimensions to a fixed size, making it easier to connect to fully connected layers. The flattened output of the pooling layer is then fed into a fully connected layer, which maps the extracted features to a specified number of output classes, typically corresponding to different categories in a classification task.

For this experiment, I selected a batch size of 64 and the optimizer selected was Adam with a learning rate of 1e-3. I also selected Cross Entropy loss for training the network. The training results are shown in figures 18 through 21.

```
Number of parameters in this model are 51
  0%|          | 0/10000 [00:00<?, ?it/s]
[ 22   0  55 611   7   7   0   1   0 297] (0)
[  1   0   1 495   1   0   0   0   0 502] (1)
[  1   0  65 873   6  17   0   4   0  34] (2)
[  0   0   2 959   4   6   0   0   0  29] (3)
[  0   0   6 937  20  12   0   1   0  24] (4)
[  0   0   2 950   3  18   0   2   0  25] (5)
[  0   0   3 955   2   4   0   0   0  36] (6)
[  0   0   1 919  11  18   0  11   0  40] (7)
[  3   0   6 696   2   1   0   1   0 291] (8)
[  0   0   0 562   3   0   0   0   0 435] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 15.3 %
```

Fig. 17. Confusion matrix after training for ResNet

Fig. 18. Architecture for ResNeXt

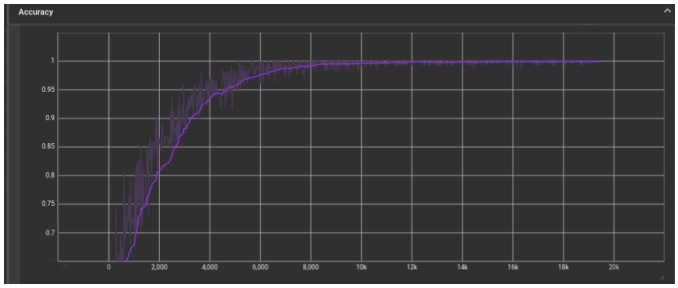

Fig. 19. Train Accuracy for ResNeXt

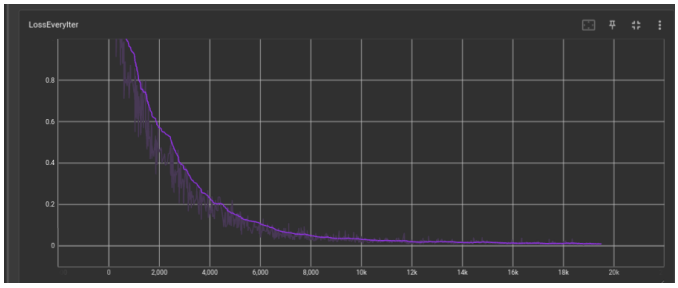

Fig. 20. Train Loss for ResNeXt

E. DenseNet

This network is the DenseNet architecture, designed for image classification. It starts with an initial convolutional layer that processes the input image, followed by batch normalization and a ReLU activation function for non-linearity.

The network's distinctive feature is its use of dense blocks, where each block includes a batch normalization layer, a convolutional layer, and then combines its output with the input features using concatenation. This approach allows each layer to access feature maps from all preceding layers, promoting feature reuse and reducing the number of parameters, making the network more efficient.

After each dense block, except for the last one, there's a transition layer that includes batch normalization, a convolutional layer with a smaller kernel size, and average pooling. This layer reduces the dimensionality of the feature maps, thus

```
Number of parameters in this model are 62
100%|██████████| 10000/10000 [00:29<00:00, 266.28it/s]
[  0   0  74 926   0   0   0   0   0   0] (0)
[  0   0  15 984   0   0   0   0   0   1] (1)
[  0   0 100 900   0   0   0   0   0   0] (2)
[  0   0  16 984   0   0   0   0   0   0] (3)
[  0   0  23 977   0   0   0   0   0   0] (4)
[  0   0  16 984   0   0   0   0   0   0] (5)
[  0   0  12 988   0   0   0   0   0   0] (6)
[  0   0  39 959   0   0   0   2   0   0] (7)
[  0   0  47 953   0   0   0   0   0   0] (8)
[  0   0  10 989   0   0   0   0   0   1] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 10.87 %
```

Fig. 21. Confusion matrix after training for ResNeXt

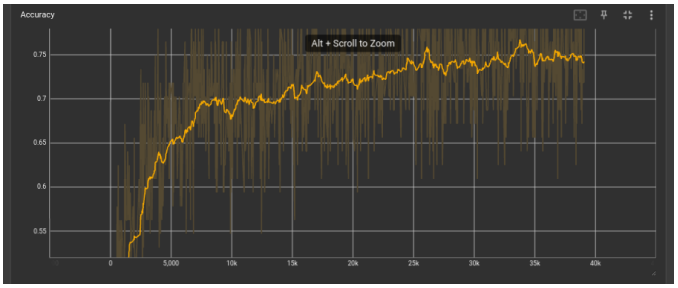

Fig. 22. Train Accuracy for DenseNet

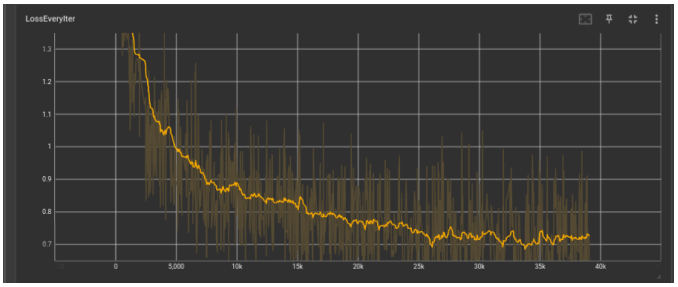

Fig. 23. Train Loss for DenseNet

controlling the growth of computational requirements.

The number of features in each dense block increases due to the concatenation of input and output feature maps, but the transition layers help in managing this growth by compressing the number of feature maps.

The network concludes with a final batch normalization, an adaptive average pooling layer to ensure a consistent output size, and a fully connected linear layer that maps the extracted features to the desired number of output classes, corresponding to different categories in the classification task.

For this experiment, I selected a batch size of 64 and the optimizer selected was Adam with a learning rate of 1e-3. I also selected Cross Entropy loss for training the network. The training results are shown in figures 22 through 25.

```
Number of parameters in this model are 43
100% ████████████████████ 10000/10000 [00:24<00:00, 467.97it/s]
[  2   0   6 992   0   0   0   0   0   0] (0)
[  1   0   0 999   0   0   0   0   0   0] (1)
[  1   0   2 997   0   0   0   0   0   0] (2)
[   0    0    0 1000    0    0    0    0    0    0] (3)
[  0   0   1 999   0   0   0   0   0   0] (4)
[   0    0    0 1000    0    0    0    0    0    0] (5)
[   0    0    0 1000    0    0    0    0    0    0] (6)
[  0   0   1 999   0   0   0   0   0   0] (7)
[  1   0   0 999   0   0   0   0   0   0] (8)
[   0    0    0 1000    0    0    0    0    0    0] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 10.04 %
```

Fig. 24. Confusion matrix after training for DenseNet

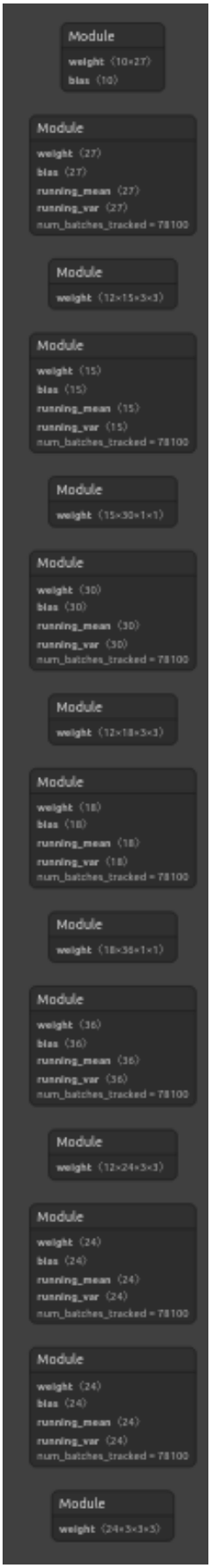

Fig. 25. Architecture for DenseNet