RBE/CS 549: Computer Vision
HW 0: Alohomora

Nikesh Walling
Robotics Engineering and
Computer Science
Worcester Polytechnic Institute
Worcester, Massachusetts
Email: ngwalling@wpi.edu

***Abstract*—This is the introductory homework assignment for RBE/CS 549: Computer Vision. It is divided into two phases. Phase 1, Shake my Boundary, involves implementing a simplified version of the Pb-Lite boundary detection algorithm. Phase 2, Deep Dive into Deep Learning, involves implementing several neural network architectures, such as ResNet, ResNeXt, and DenseNet, to evaluate the CIFAR10 dataset.**

Keywords- Gaussian Filter, Garbor Filter, Leung-Malik Filter, Canny Edge, Sobel Edge, ResNet, ResNeXt, DenseNet

I. Phase 1: Shame My Boundary

The first phase of this assignment required implementing a probability of boundary detection algorithm called PbLite. The algorithm attempts to discern boundaries by combining several pieces of image information such as color, brightness, and texture. It also combines the outputs of the standard Canny and Sobel filter edge detection algorithms. Overall, PbLite uses five main steps to detect likely edges:

1) Filter bank and mask creation
2) Creating a texton (texture), brightness, and color map using the filters and K-Means Clustering
3) Taking the Chi-Squared distances using the masks and the maps to perform gradient analysis
4) Perform Canny and Sobel functions on the input image
5) Combine the outputs of each method to create the output

A. Filters

Three filter banks were implemented in this assignment: the derivative of Gaussian filter, the Leung-Malik filter, and the Garbor filter. Additionally, half-disk masks were created for implementing gradient calculations. Each of these filters extract information from the image to be used for edge detection. Each filter type will be covered in the following sections.

1) Derviative of Gaussian Filters: Oriented Derivative of Gaussian (DoG) filters are an effective method for detecting edges and textures. These filters are constructed by convolving a Sobel filter with a Gaussian kernel and then rotating the result to different orientations. Given o orientations (ranging from 0° to 360°) and s scales, the total number of filters is $s \times o$. Figure 1 illustrates an example DoG filter bank with 16 orientations and 2 scales.

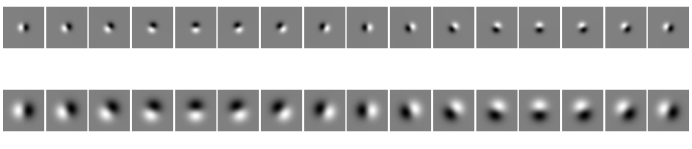

Fig. 1: Example of DoG Filters with 2 Scales and 16 Orientations

2) Leung-Malik Filters: The Leung-Malik (LM) filter bank is a multi-scale, multi-orientation collection consisting of 48 filters. It includes first and second derivatives of Gaussians at six orientations and three scales (36 filters), eight Laplacian of Gaussian (LoG) filters, and four Gaussian filters. Two versions of LM filter banks exist: LM Small (LMS) and LM Large (LML). The LMS filters are defined at scales $\sigma = \{1, \sqrt{2}, 2, 2\sqrt{2}\}$, while LML filters use $\sigma = \{\sqrt{2}, 2, 2\sqrt{2}, 4\}$. Figure **??** illustrates an LM filter bank.

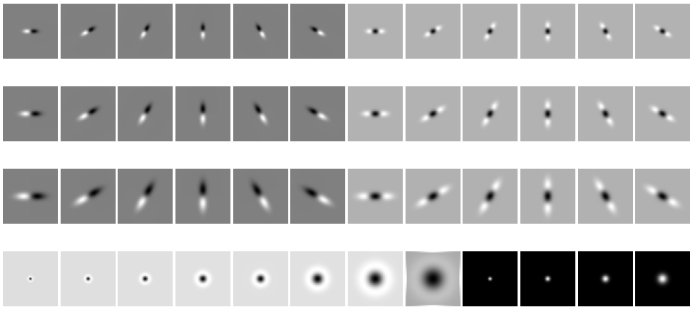

Fig. 2: Example of a Small Leung-Malik filter bank

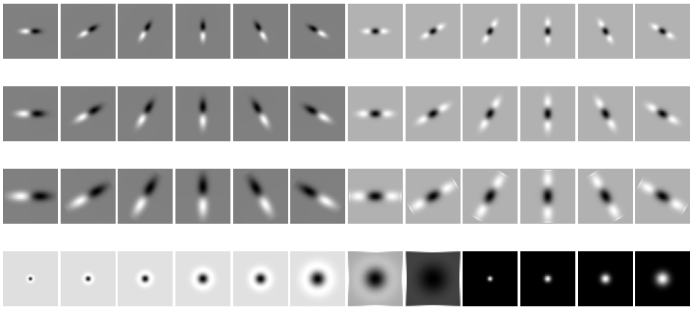

Fig. 3: Example of a Large Leung-Malik filter bank

3) Garbor Filters: The human visual system inspires Gabor filters and consist of a Gaussian kernel modulated by a sinusoidal plane wave. These filters are widely used for texture analysis due to their frequency and orientation selectivity. Figure 4 shows an example Gabor filter bank. The filters have a kernel size of 63 pixels and $\sigma = \{2, 4, 6\}$ across the rows.

B. Texton, Brightness, and Color Maps with Clustering

1) Texton Map: Filtering an image using a combination of filter banks results in a vector of filter responses at each pixel. If

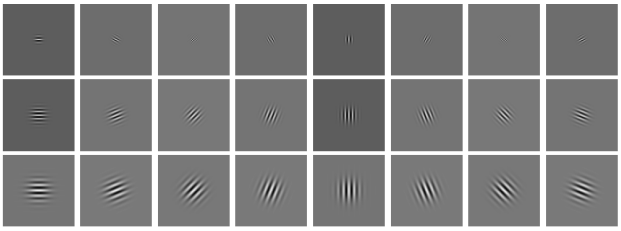

Fig. 4: Gabor filter bank.

the filter bank consists of N filters, then each pixel is associated with an N-dimensional response vector. This distribution of filter responses encodes texture properties. To simplify this representation, we perform vector quantization by clustering the filter responses across all pixels into K textons using K-Means clustering. Each pixel is then represented by a single discrete cluster ID instead of an N-dimensional response vector. The resulting **Texton Map** is a single-channel image where pixel values range from 1 to K. A typical choice for K is 64, though this parameter can be tuned.

2) Brightness Map: The **Brightness Map** captures variations in intensity across the image. This is achieved by converting the image to grayscale and clustering the brightness values using K-Means clustering. The output is a clustered grayscale image where each pixel is assigned to one of K brightness clusters. Typically, $K = 16$ is used, but other values may be experimented with.

3) Color Map: The **Color Map** captures chrominance variations in an image by clustering pixel color values. Instead of directly clustering RGB values, alternative color spaces such as YCbCr, HSV, or Lab can be used for better perceptual consistency. K-Means clustering is applied to assign each pixel to one of K clusters. A standard choice is $K = 16$, but different methods and clustering approaches can be explored.

The Texton Map, Brightness Map, and Color Map provide key feature representations for boundary detection. By reducing high-dimensional filter responses and pixel values into clustered feature maps, these techniques enable robust texture and edge analysis. Figure 5 shows the resulting feature maps of the input images with the typical clustering values. For the Texton Map, DoG, LM-Small, LM-Large, and Garbor filters were used to extract the responses before clustering.

C. Texture, Brightness, and Color Gradients

1) Half-Disk Masks: The Half-disc masks are binary representations of half-discs at different orientations and scales, as illustrated in Fig. 6. A sample set of masks consisting of 8 orientations and 3 scales is shown. These masks facilitate the computation of the χ^2 (chi-square) distance, which measures differences in texture, brightness, and color distributions. The scales used for these half disks are 8, 13, and 21.

D. Gradient Computation using χ^2 Distance

The gradients T_g, B_g, and C_g capture changes in texture, brightness, and color distributions at a pixel by comparing distributions in opposing half-disc pairs. If distributions are similar, the gradient remains small; otherwise, a significant gradient value is obtained. These gradients, spanning multiple scales and orientations, provide local measurements encoding the rate of change in texture, brightness, and color.

The χ^2 distance is used to compare histogram distributions between two regions. Given two histograms g and h with K bins, the χ^2 distance is defined as:

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

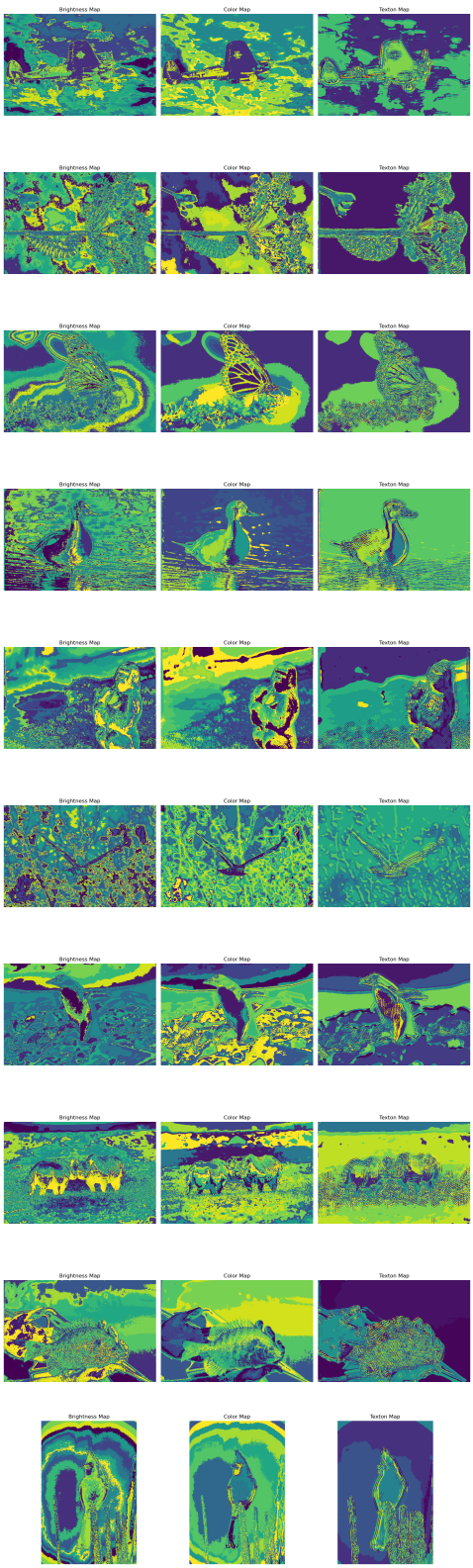

Fig. 5: The Maps for the given input images.

where i indexes the histogram bins. The denominator provides a normalization term, ensuring that less frequent elements still

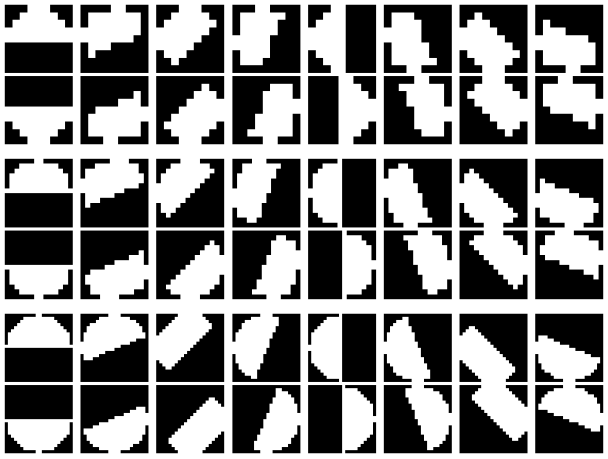

Fig. 6: Half-disc masks at different scales and orientations.

contribute meaningfully to the overall distance. The gradient outputs can be seen in Figure **??**.

II. Baseline Methods and Pb-lite Output

A. Sobel and Canny Baselines

To evaluate the performance of the proposed boundary detection pipeline, we utilize the Sobel and Canny edge detection methods as baselines. These baselines serve as references for assessing the effectiveness of the Pb-lite method in capturing boundary structures.

B. Pb-lite Output

The final boundary probability (Pb) map is obtained by integrating the extracted features with the baseline methods using a weighted combination. The Pb-lite computation is defined as:

$$PbEdges = \frac{(T_g + B_g + C_g)}{3} \odot (w_1 \cdot CannyPb + w_2 \cdot SobelPb) \tag{2}$$

where $\odot$ represents the Hadamard (element-wise) product, and w_1 and w_2 are weights assigned to the baseline methods. A simple choice is $w_1 = w_2 = 0.5$, ensuring that their sum equals 1. However, these weights can be dynamically adjusted based on data characteristics.

C. Feature Integration Strategy

The magnitude of texture, brightness, and color features influences the boundary strength. A simple mean of the feature vector at location i provides a proportional estimate of the probability of a boundary. While the initial formulation employs an element-wise product between the baseline output and mean feature strength, more advanced feature fusion strategies can be explored for improved performance. Figure 8 show the comparison between the input image, ground truth edges, and the output from the PbLite implementation.

III. Phase 2: Deep Dive into Deep Learning

For this phase, multiple neural network architectures will be implemented and evaluated based on criteria such as the number of parameters, training and testing accuracies, and overall performance.

The dataset used for evaluation is **CIFAR-10**, which consists of 60,000 color images of size 32×32 across 10 classes. The dataset is split into:

- **Training Set:** 50,000 images
- **Test Set:** 10,000 images

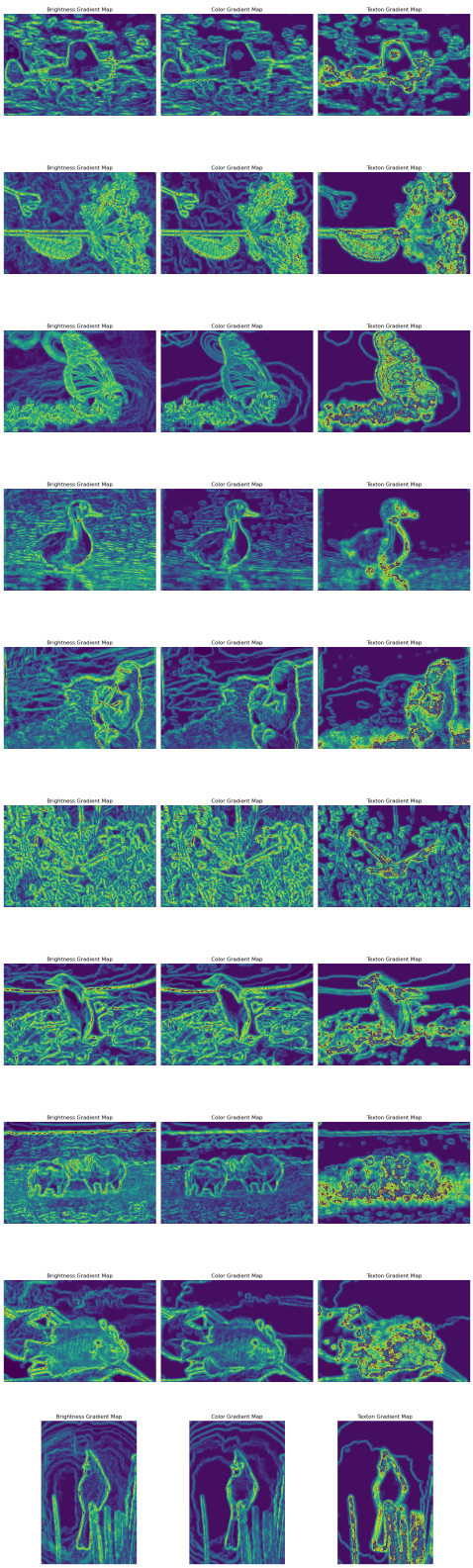

Fig. 7: The Gradient Maps for the given input images.

A. First Neural Network

As the first attempt, a simple fully connected neural network was implemented; its structure can be seen in Figure I with a total of

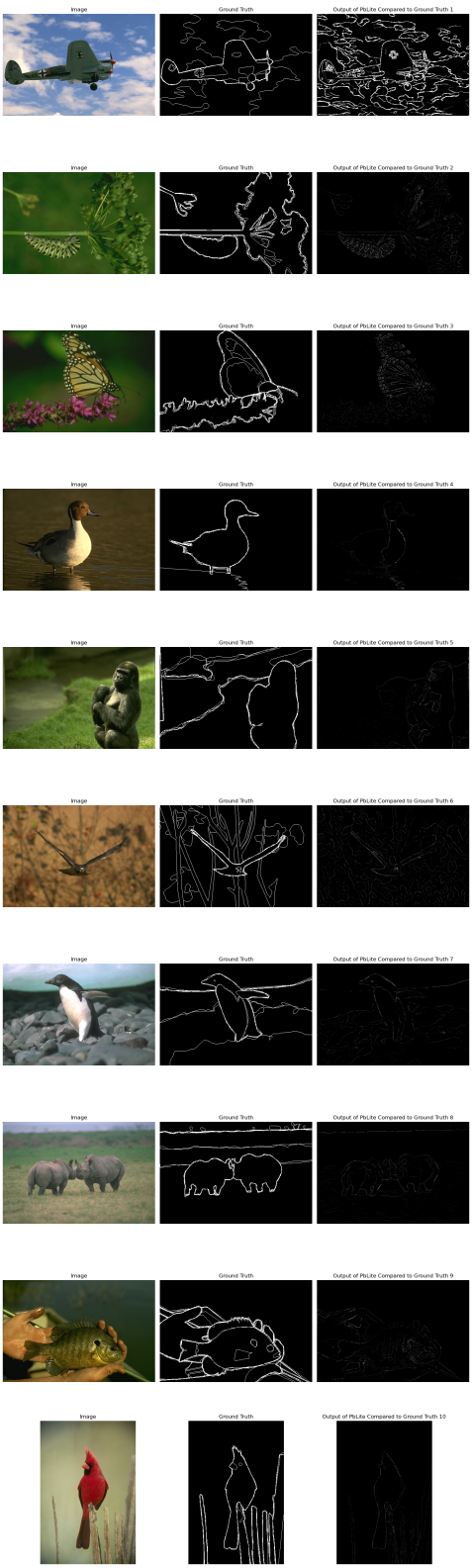

Fig. 8: The input, ground truth edges, and PbLite output.

828490 parameters.

The network was trained for 12 epochs with a batch size of 256, a learning rate of 0.01 with the Adam optimizer. Unfortunately, a picture of the architecture could not be obtained.

The following are the results of training the network:

TABLE I: Neural Network Architecture Summary

Layer	Type	Output Dimensions
FC1	Fully Connected	256
FC2	Fully Connected	128
FC3	Fully Connected	64
FC4	Fully Connected	OutputSize

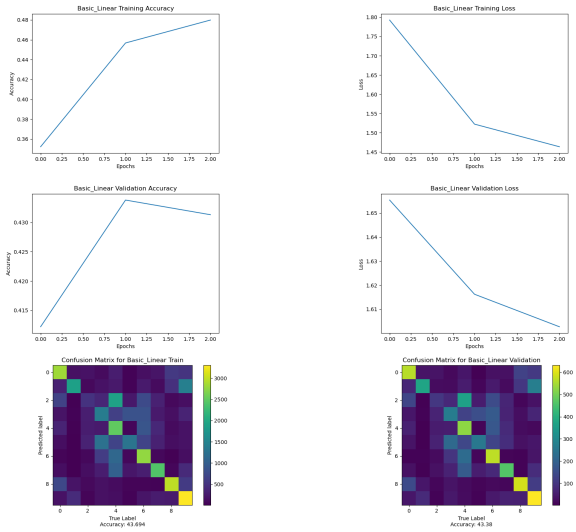

Fig. 9: Training and Testing results of the basic model

B. Next Model

As an improvement to the basic fully connected network, this one is a convolutional network. Augmenting features such as batch normalization were omitted to highlight the difference in capabilities for a similarly sized network in terms of parameters. The network consists of three convolutional layers followed by two fully connected layers. Table II summarizes the architecture of the network. This network has 8413706 parameters.

TABLE II: Convolutional Neural Network Architecture Summary

Layer	Type	Output Dimensions
Conv1	Convolutional (3x3, 16 filters)	16x32x32
Conv2	Convolutional (3x3, 32 filters)	32x32x32
Conv3	Convolutional (3x3, 64 filters)	64x32x32
FC1	Fully Connected	128
FC2	Fully Connected	OutputSize

The network was trained for 12 epochs with a batch size of 128, a learning rate of 0.01 with the Adam optimizer. Unfortunately, a picture of the architecture could not be obtained.

The following are the results of training the network:

C. ResNet, ResNext, and DenseNet Implementations

After implementing fully connected and convolutional neural networks, more advanced architectures were implemented and tested on the CIFAR10 dataset.

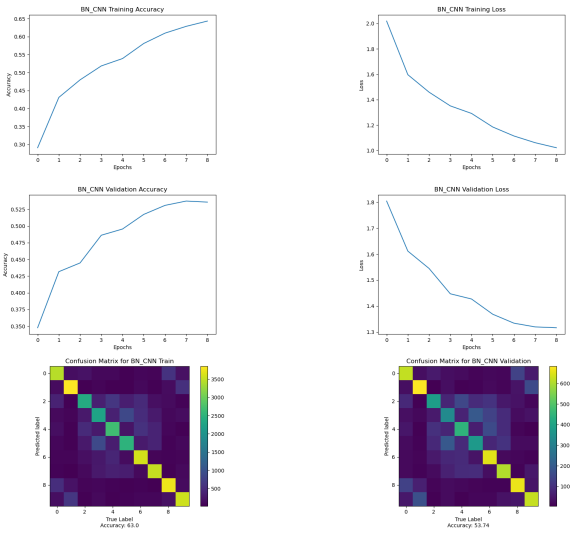

Fig. 10: Training and Testing results of the basic cnn model

1) ResNet: The first complex model implemented was based off ResNet. It has the following structure:

The `BasicBlock` and `CIFAR10Model_ResNet` models are based on the ResNet architecture, which is designed to solve the vanishing gradient problem by introducing residual (skip) connections. These connections allow the gradient to flow more easily through the network, enabling training of very deep models.

The `BasicBlock` consists of two 3x3 convolutional layers with batch normalization and ReLU activations. A skip connection is added to bypass the block, allowing the input to be added directly to the output, helping with gradient propagation.

TABLE III: BasicBlock Architecture Summary

Layer	Type	Output Dimensions
Conv2d1	Convolutional (3x3)	(OutChannels, H, W)
BatchNorm1	Batch Normalization	(OutChannels, H, W)
ReLU1	Activation Function	(OutChannels, H, W)
Conv2d2	Convolutional (3x3)	(OutChannels, H, W)
BatchNorm2	Batch Normalization	(OutChannels, H, W)
Shortcut	Identity Mapping / Conv2d	(OutChannels, H, W)
ReLU2	Activation Function	(OutChannels, H, W)

Explanation: The `BasicBlock` consists of two convolutional layers followed by batch normalization and ReLU activations. The skip connection is added to the output of the second convolutional layer, allowing the network to preserve information from earlier layers. This makes it easier to train deeper models by addressing the vanishing gradient problem.

TABLE IV: CIFAR10Model_ResNet Architecture Summary

Layer	Type	Output Dimensions
Conv1	Convolutional (3x3)	(64, 32, 32)
BatchNorm1	Batch Normalization	(64, 32, 32)
Layer1	Sequential (BasicBlock)	(64, 32, 32)
Layer2	Sequential (BasicBlock)	(128, 16, 16)
Layer3	Sequential (BasicBlock)	(256, 8, 8)
Layer4	Sequential (BasicBlock)	(512, 4, 4)
AvgPool	Adaptive Average Pooling	(512, 1, 1)
Dropout	Dropout (0.2)	(512, 1, 1)
FC	Fully Connected Layer	(OutputSize)

IV. Description

The `conv_block` consists of a convolutional layer followed by batch normalization and ReLU activation. An optional max-pooling operation is included to downsample the feature map.

The `BasicBlock` is a fundamental component of ResNet. It contains two convolutional layers with 3x3 kernels and skip connections (identity mapping) that help mitigate vanishing gradients. If the input and output dimensions do not match, a 1x1 convolution is used for the shortcut path to adjust the dimensions.

The model has 11173962 parameters, and inference takes roughly 0.00277 seconds on an NVIDIA RTX 3050Ti. This network was trained for 12 epochs with a batch size of 200 and learning rate of 0.01 using the Adam optimizer and a StepLR scheduler with a patience of 3 iterations.

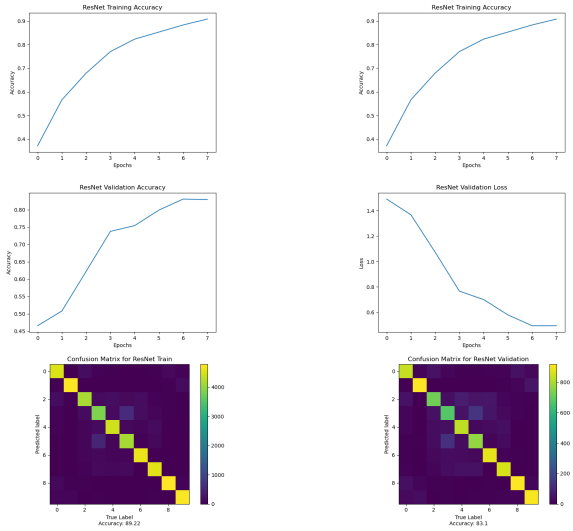

Fig. 11: Training and Testing results of the ResNet

1) ResNeXt: After ResNet, the ResNext architecture was implanted. The `ResNeXtBlock` is designed to improve the efficiency of the residual network by incorporating grouped convolutions (using the cardinality hyperparameter). This block consists of three convolutional layers and a skip connection.

TABLE V: ResNeXtBlock Architecture Summary

Layer	Type	Output Dimensions
Conv1x1_1	Convolutional (1x1)	(Bottleneck Channels)
Conv3x3	Convolutional (3x3, Grouped)	(Bottleneck Channels)
Conv1x1_2	Convolutional (1x1)	(Output Channels)
BatchNorm	Batch Normalization	(Output Channels)
Adapter	Convolutional (1x1)	(Output Channels)
Skip Connection	Identity Mapping or Conv1x1	(Output Channels)

Explanation: The `ResNeXtBlock` uses the grouped convolution method to split the input into multiple groups and apply convolutions in parallel, increasing the feature diversity. The residual connection is employed to combine the output of the block with the input, which helps mitigate the vanishing gradient problem and allows for better training.

Explanation: The architecture consists of an initial convolutional layer, followed by two sequential layers, `Layer1` and `Layer2`, each containing multiple `ResNeXtBlock`s. Each `ResNeXtBlock` processes the image with grouped convolutions, allowing for efficient representation learning. After the sequential layers, the output is passed through an adaptive average pooling layer, followed by a fully connected layer to produce the final classification output.

TABLE VI: CIFAR10Model_ResNeXt Architecture Summary

Layer	Type	Output Dimensions
Conv1	Convolutional (3x3)	(64, 32, 32)
BatchNorm	Batch Normalization	(64, 32, 32)
Layer1 (3 ResNeXtBlocks)	Stack of ResNeXtBlocks	(128, 32, 32)
Layer2 (3 ResNeXtBlocks)	Stack of ResNeXtBlocks	(512, 8, 8)
AvgPool	Adaptive Average Pooling	(512, 1, 1)
Flatten	Flatten Layer	(512)
FC	Fully Connected Layer	(OutputSize)

The network has the following structure with a total of 842890 parameters and an inference time of 0.00188 seconds. This network was trained for 8 epochs with a batch size of 80 and learning rate of 0.1 using an SGD optimizer and a StepLR scheduler with a patience of 3 iterations.

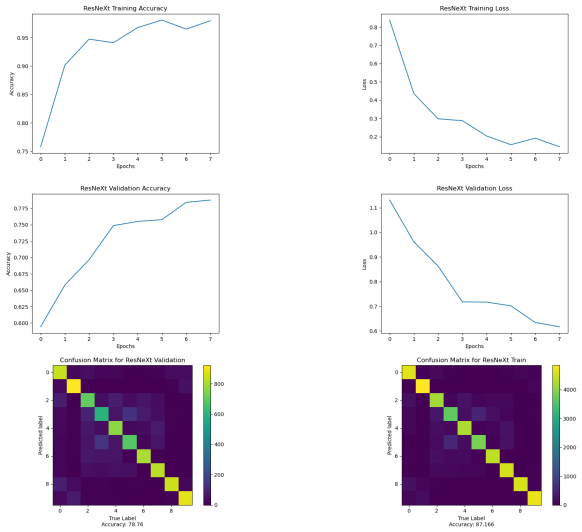

Fig. 12: Training and Testing results of the ResNext model

A. *DenseNet*

the final architecture implemented was DenseNet.

The `DenseBlock` and `CIFAR10Model_DenseNet` models are based on the DenseNet architecture, which introduces dense connections between layers to improve feature propagation and reduce the vanishing gradient problem. In DenseNet, each layer receives input from all previous layers, which increases the efficiency of feature reuse.

The `DenseBlock` consists of a series of convolutional layers where each layer's input is the concatenation of the previous layers' outputs. This architecture allows for better feature reuse and reduces the need for deeper networks.

TABLE VII: DenseBlock Architecture Summary

Layer	Type	Output Dimensions
BatchNorm	Batch Normalization	(Channels + GrowthRate * i)
ReLU	Activation Function	(Channels + GrowthRate * i)
Conv2d	Convolutional (3x3)	(GrowthRate)

Explanation: In the `DenseBlock`, each convolutional layer takes as input the concatenation of the input and the outputs from all previous layers. This results in a growing number of channels after each layer, leading to better feature reuse and more efficient representation learning. The growth rate controls how many new channels are added at each layer.

The `DenseNet` model applies multiple `DenseBlocks` sequentially, with transition layers in between. Transition layers reduce the number of channels after each dense block and downsample the spatial dimensions using average pooling. Finally, the output is passed through an adaptive average pooling layer followed by a fully connected layer to produce class predictions.

TABLE VIII: CIFAR10Model_DenseNet Architecture Summary

Layer	Type	Output Dimensions
Conv1	Convolutional (3x3)	(64, 32, 32)
BatchNorm	Batch Normalization	(64, 32, 32)
Dense1 (4 layers)	Dense Block	(64 + 4 * GrowthRate, 32, 32)
Trans1	Transition Layer	(128, 16, 16)
Dense2 (4 layers)	Dense Block	(128 + 4 * GrowthRate, 16, 16)
Trans2	Transition Layer	(256, 8, 8)
Dense3 (3 layers)	Dense Block	(256 + 3 * GrowthRate, 8, 8)
BatchNorm	Batch Normalization	(256 + 3 * GrowthRate, 8, 8)
AvgPool	Adaptive Average Pooling	(256 + 3 * GrowthRate, 1, 1)
Flatten	Flatten Layer	(256 + 3 * GrowthRate)
FC	Fully Connected Layer	(OutputSize)

Explanation: The architecture consists of an initial convolutional layer, followed by three dense blocks with transition layers in between. The transition layers downsample the feature maps using average pooling and reduce the number of channels. The final output is passed through an adaptive average pooling layer to flatten the features, followed by a fully connected layer that outputs the predicted class.

The network has the following structure with a total of 498370 parameters and an inference time of 0.002 seconds; the network has a growth factor of 24. This network was trained for 12 epochs with a batch size of 128 and learning rate of 0.1 using an SGD optimizer and a StepLR scheduler with a patience of 3 iterations.

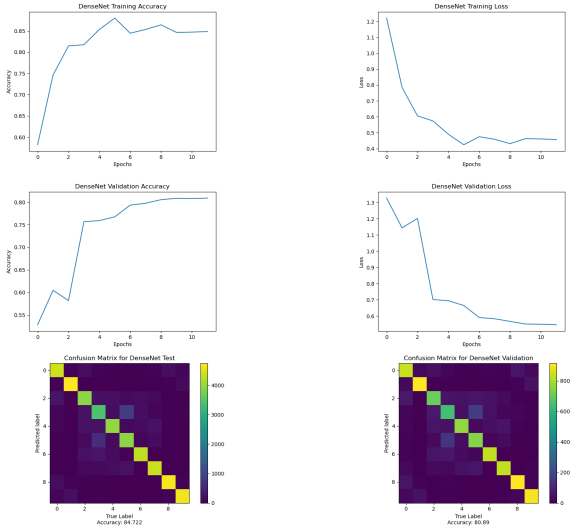

Fig. 13: Training and Testing results of the DenseNet model

V. Conclusion

Between all of the neural networks, the ResNeXt model has the best results in terms of accuracy in these implementations with the lowest inference time. Likely the other networks would perform better, but there were limitations in the training capabilities for these models.

Note, did not have time to format a table.

The authors would like to thank...