Muhammed Jammeh
200254645
ECS659P/ECS7026P

Report on Neural Network Architecture

This report examines the neural network architecture implemented in the provided code and highlights the deviations from the basic architecture initially described. Additionally, it briefly outlines the hyperparameters and techniques employed for training the model and the results. The implemented neural network architecture closely follows the described architecture, with some minor deviations. The network consists of three main components: intermediate blocks, an output block, and additional components for processing and classification.

Intermediate Blocks

The IntermediateBlock class represents the intermediate blocks of the network. Each block adheres to the described architecture, with the following key features:

1. **Independent Convolutional Layers**: Each block contains multiple independent convolutional layers (self.conv), where each layer receives the same input image x.
2. **Linear Combination Output**: The output image x' is computed as a linear combination of the outputs from each convolutional layer, weighted by a vector a (out = a.expand_as(self.conv(x)) * self.bn(self.conv(x))).
3. **Computation of Vector a**: The vector a is computed from the average value of each channel of the input image x (x_avg = x.view(batch_size, channels, -1).mean(dim=2)), passed through a fully connected layer (self.fc).
4. **Batch Normalization**: The implementation includes batch normalization (self.bn) applied to the output of the convolutional layers.

There are 3 intermediate blocks in the network below are the hyperparameters for Block 1 :

- Input Channels: 3 (RGB images), Output Channels: 64
- Convolutional Layer:
  - Kernel Size: 3
  - Padding: 1
- Batch Normalization: Applied after the convolutional layer.
- Fully Connected Layer:
  - Input Size: Same as the number of input channels (3)
  - Output Size: Same as the number of output channels (64)

The remaining two blocks follow the same system, the only difference being the input and output channels.

Output Block

The output block is implemented within the Net class and follows the described architecture:

1. It receives the output of the last intermediate block (self.block3).
2. The logits vector o (self.fc2(x)) is computed from the average value of each channel of the input image (x = x.view(-1, 256 * 4 * 4)), passed through one or more fully connected layers (self.fc1, self.fc2).

Muhammed Jammeh
200254645
ECS659P/ECS7026P

Additional Components

The Net class also includes the following components:

1. **Max Pooling**: A max pooling layer (self.pool) is applied after each intermediate block. Used after each block to reduce the spatial dimensions of the output
2. **Fully Connected Layers**: Two fully connected layers (self.fc1 and self.fc2) are used for classification, with a ReLU activation function applied to the output of self.fc1.

Hyperparameters and Training Techniques

The provided code employs the following hyperparameters and techniques for training the model:

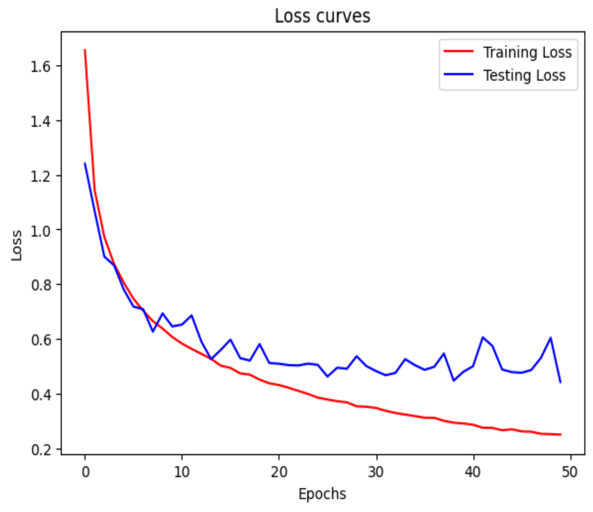
1. **Optimizer**: The Stochastic Gradient Descent (SGD) optimizer is used with a learning rate of 0.01, momentum of 0.9, and weight decay of 5e-4.
2. **Learning Rate Scheduler**: A ReduceLROnPlateau scheduler is employed to decrease the learning rate if the validation loss plateaus for five epochs.
3. **Loss Function**: The Cross-Entropy Loss (nn.CrossEntropyLoss) is used as the loss function.
4. **Data Augmentation**: The training data is augmented using random horizontal flipping and random cropping.
5. **Batch Size**: A batch size of 128 is used for both training and testing.
6. **Number of Epochs**: The model is trained for 50 epochs.

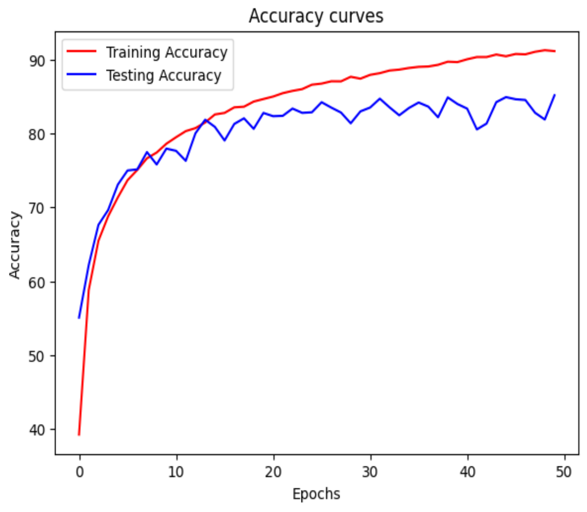
Initial Designs and Conclusion/Results

The first model that I designed was a bit different from my best-performing model. My previous iteration did not utilize batch normalization, which made my final version more stable and quicker through normalization of the layer's inputs. My previous iterations also utilized dropout, I thought the dropout would prevent overfitting, however, I set the dropout rate too high and used it inappropriately which caused the model to underfit, resulting in accuracies in the range of 30% to 40%.

This architecture is designed to progressively increase the depth of the network while also applying batch normalization and a unique method of weighting the outputs of convolutional layers using coefficients obtained from a fully connected layer that receives the average value of each channel of the input image. The highest accuracy I obtained in the testing dataset from the final iteration of my Neural Network was **85.17%**

```
Best Test Accuracy: 85.170
```


Loss curves
Training Loss
Testing Loss
Loss
Epochs


Accuracy curves
Training Accuracy
Testing Accuracy
Accuracy
Epochs