Final Project Report

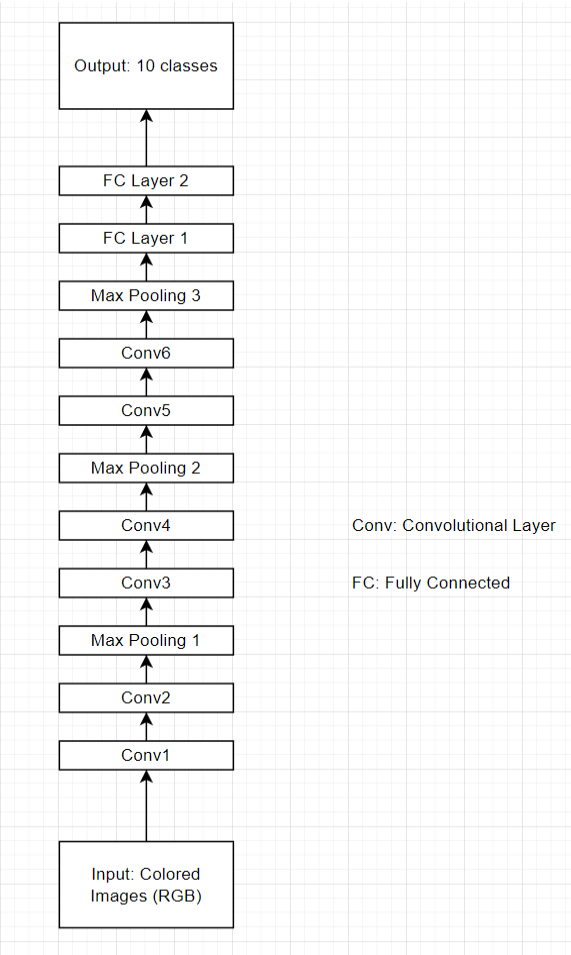
Huy Nguyen

I. Task Description

Out of the 2 topics given for this project, I chose to work on Topic A - Improving the performance on homework 7, which, in this case, is designing a different CNN model to recognize and classify 32x32x3 colored images from the CIFAR-10 dataset. The model should have at most 10 layers and should achieve higher accuracy than the model used in homework 7 (YourNet - 62% accuracy)

II. Final Model

- The final model would look like so:



- The activation function used is ReLU.
- A batch normalization function with appropriate parameters would follow each convolutional layer, right before the activation function. One batch normalization function follows the first fully connected layer before the activation function.
- One dropout function is added right before the first fully connected layer, and one is added right before the second fully connected layer. Both have 0.5 dropping rate.
- Hyperparameters used:
    + Learning rate = 0.01
    + Batch size = 128
    + Epochs = 200
    + SGD optimizer
    + Momentum = 0.9
    + Weight decay = 5e-4

- The actual in-code model looks like so:

```
class YourNet(nn.Module):
  def __init__(self):
    super(YourNet, self).__init__()

    self.feature_extractor = nn.Sequential(
            # C1
            nn.Conv2d(in_channels = 3, out_channels = 8, kernel_size = 11, stride = 1, padding = 5),
            nn.BatchNorm2d(num_features=8),
            nn.ReLU(),
            # C2
            nn.Conv2d(in_channels = 8, out_channels = 16, kernel_size = 7, stride = 1, padding = 3),
            nn.BatchNorm2d(num_features=16),
            nn.ReLU(),
            # MP1
            nn.MaxPool2d(kernel_size = 2, stride = 2, padding = 0),
            # C3
            nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 5, stride = 1, padding = 2),
            nn.BatchNorm2d(num_features=32),
            nn.ReLU(),
            # C4
            nn.Conv2d(in_channels = 32, out_channels = 64, kernel_size = 5, stride = 1, padding = 2),
            nn.BatchNorm2d(num_features=64),
            nn.ReLU(),
            # MP2
            nn.MaxPool2d(kernel_size = 2, stride = 2, padding = 0),
            # C5
            nn.Conv2d(in_channels = 64, out_channels = 128, kernel_size = 4, stride = 1, padding = 0),
            nn.BatchNorm2d(num_features=128),
            nn.ReLU(),
            # C6
            nn.Conv2d(in_channels = 128, out_channels = 256, kernel_size = 4, stride = 1, padding = 0),
            nn.BatchNorm2d(num_features=256),
            nn.ReLU(),
            # MP3
            nn.MaxPool2d(kernel_size = 2, stride = 2, padding = 0),
            #Dropout
            nn.Dropout(p=0.5)
        )

    self.classifier = nn.Sequential(
            # FC1
            nn.Linear(in_features = 256, out_features = 256),
            nn.BatchNorm1d(num_features=256),
            nn.ReLU(),
            #Dropout
            nn.Dropout(p=0.5),
            # FC2
            nn.Linear(in_features = 256, out_features = 10)
        )
```

- A summary of the model using *torchsummary*:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 8, 32, 32]           2,912
       BatchNorm2d-2           [-1, 8, 32, 32]              16
              ReLU-3           [-1, 8, 32, 32]               0
            Conv2d-4          [-1, 16, 32, 32]           6,288
       BatchNorm2d-5          [-1, 16, 32, 32]              32
              ReLU-6          [-1, 16, 32, 32]               0
         MaxPool2d-7          [-1, 16, 16, 16]               0
            Conv2d-8          [-1, 32, 16, 16]          12,832
       BatchNorm2d-9          [-1, 32, 16, 16]              64
             ReLU-10          [-1, 32, 16, 16]               0
           Conv2d-11          [-1, 64, 16, 16]          51,264
      BatchNorm2d-12          [-1, 64, 16, 16]             128
             ReLU-13          [-1, 64, 16, 16]               0
        MaxPool2d-14            [-1, 64, 8, 8]               0
           Conv2d-15           [-1, 128, 5, 5]         131,200
      BatchNorm2d-16           [-1, 128, 5, 5]             256
             ReLU-17           [-1, 128, 5, 5]               0
           Conv2d-18           [-1, 256, 2, 2]         524,544
      BatchNorm2d-19           [-1, 256, 2, 2]             512
             ReLU-20           [-1, 256, 2, 2]               0
        MaxPool2d-21           [-1, 256, 1, 1]               0
          Dropout-22           [-1, 256, 1, 1]               0
           Linear-23                 [-1, 256]          65,792
      BatchNorm1d-24                 [-1, 256]             512
             ReLU-25                 [-1, 256]               0
          Dropout-26                 [-1, 256]               0
           Linear-27                  [-1, 10]           2,570
================================================================
Total params: 798,922
Trainable params: 798,922
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.30
Params size (MB): 3.05
Estimated Total Size (MB): 4.36
----------------------------------------------------------------
```

III. Performance

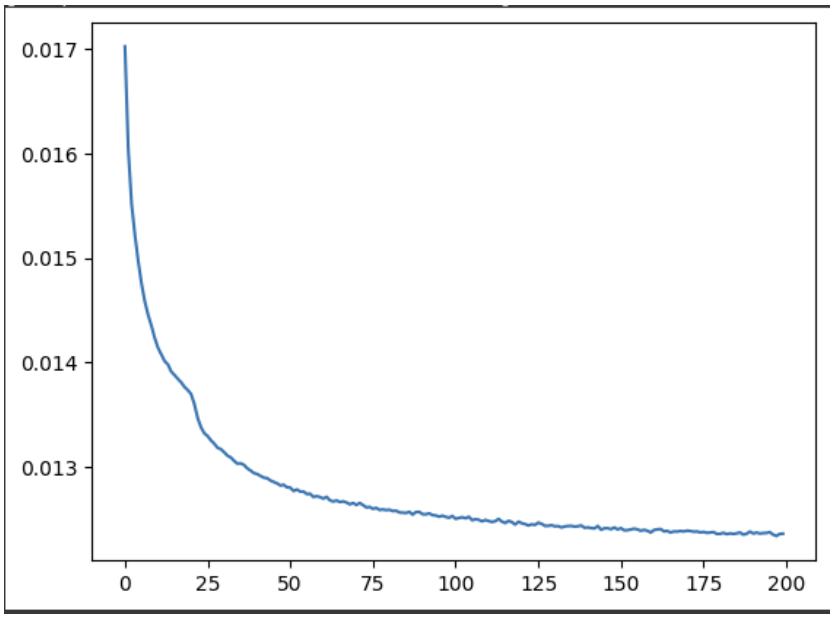
- The final accuracy achieved was 85%, significantly higher than that of the YourNet (62%)

```
Train Epoch: 200 [0/50000 (0%)] Loss: 1.544538
Train Epoch: 200 [12800/50000 (26%)]    Loss: 1.570643
Train Epoch: 200 [25600/50000 (51%)]    Loss: 1.532620
Train Epoch: 200 [38400/50000 (77%)]    Loss: 1.587359

Test set: Average loss: -0.8402, Accuracy: 8474/10000 (85%)
```

- The highest accuracy achieved during the training process was 86%, achieved on epoch 181

```
Train Epoch: 181 [0/50000 (0%)] Loss: 1.560429
Train Epoch: 181 [12800/50000 (26%)]    Loss: 1.629120
Train Epoch: 181 [25600/50000 (51%)]    Loss: 1.570524
Train Epoch: 181 [38400/50000 (77%)]    Loss: 1.568160

Test set: Average loss: -0.8477, Accuracy: 8552/10000 (86%)
```

- The *average loss* vs *epoch* plot:



IV. Discussion

- Since changing the learning rate, batch size, and epochs are not considered contributions, I decided to use the same value for these hyperparameters initially.
- I started by changing the number of channels of each convolutional layer. Each layer has 2 times more output channels than input channels. The last convolutional layer has 256 output channels. The intention of this change was to capture more complex and diverse patterns in the input data. I also added an extra max pooling layer.
- I added batch normalization before every layer to improve the input, speed up and improve the performance of the training process.
- I also added dropout to the model to prevent overfitting and improve overall performance.
- The SGD optimizer was effective in the YourNet model, so I used the same optimizer for this assignment.

- The initial run with 50 epochs reached 78% accuracy. The accuracy was still slightly increasing at the end, so I suspected the model was still capable of learning.
- I increased the epochs to 200 and got the results above.
- Even though the results might have shown that the model can still learn a bit more, at that point my Colab account had reached the GPU accelerator limit, and I did not expect the possible accuracy improvement to justify the extra effort and training time. Therefore I decided to stop at 200 epochs.

To sum up, for this project, I used the model described earlier, which could reach 86% accuracy with the settings provided above after training 200 epochs.