Optimized ResNet for CIFAR-10 Classification

Aishwarya Ghaiwat, Neha Ann Nainan, Vaibhav Rouduri

arg9653@nyu.edu, nan6504@nyu.edu, vr2470@nyu.edu
GitHub Link

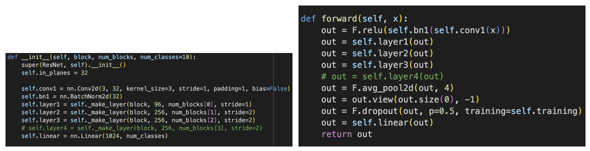
Abstract

This project presents a modified ResNet-18 architecture optimized for CIFAR-10 classification, achieving high accuracy while maintaining a constraint of less than 5 million parameters. We experimented with data augmentation techniques, multiple model architectures by varying the number of layers, the number of outputs per layer and the size of the filter, and different kinds of learning rates (constant, custom, dynamic and cyclic). We also checked different values of weight decay for L2 regularization, and experimented with dropout layers. We found that using a model architecture of approximately 4.7 million parameters with a cyclic learning rate along with our data augmentation techniques and increased weight decay led to a well-performing model, with the best model having a final training accuracy of 99.19%, a final validation accuracy of 96.26%, and a score of 0.8541 on our Kaggle submission.

Methodology

CIFAR-10 consists of 60,000 32x32 RGB images across 10 classes, with 50,000 training samples and 10,000 validation samples.

- Trainable Parameters and Architecture: The Basic Block (class BasicBlock in our code) is the fundamental building block of the ResNet architecture. It consists of two convolutional layers, each followed by batch normalization. There is also a skip connection that adds the input to the output of the block to help avoid vanishing gradients. The ResNet architecture (class ResNet in our code) consists of residual layers made of multiple Basic Blocks each. We observed that the base ResNet18 model had approximately 11.2M parameters. In order to get the number of parameters below 5M, we experimented with different filter sizes, layer sizes, and number of layers. In our first modification to the base ResNet architecture, we changed the number of outputs of the initial convolutional layer to 32, and then added 4 residual layers with 32, 64, 256 and 256 output units each. We also added a dropout layer with probability of dropout as 0.5. This led to us having 4.45M parameters in the architecture, which also gave good predictions overall. In our final architecture, we made some more modifications by having only 3 residual layers with 96, 256, and 256 output units each. This led to us having 4.7M parameters, improving on our previous architectures and leading to our best model.

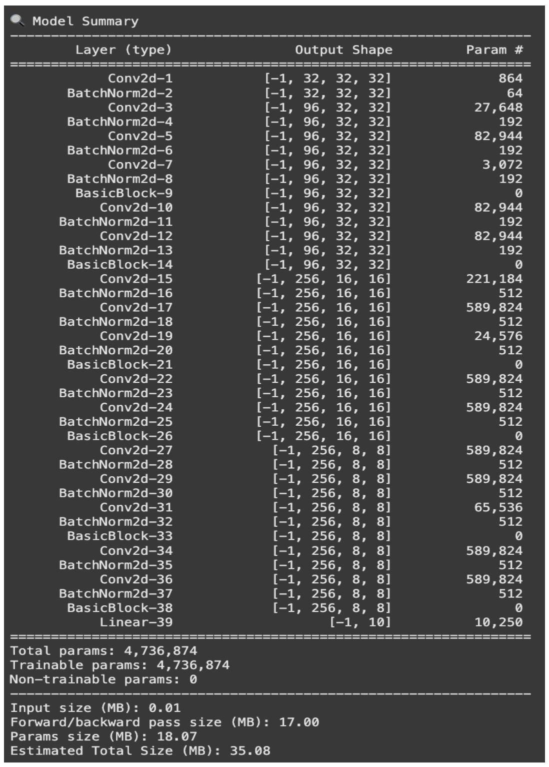


```python
def __init__(self, block, num_blocks, num_classes=10):
    super(ResNet, self).__init__()
    self.in_planes = 32

    self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1, bias=False)
    self.bn1 = nn.BatchNorm2d(32)
    self.layer1 = self._make_layer(block, 96, num_blocks[0], stride=1)
    self.layer2 = self._make_layer(block, 256, num_blocks[1], stride=2)
    self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
    # self.layer4 = self._make_layer(block, 256, num_blocks[3], stride=2)
    self.linear = nn.Linear(1024, num_classes)
```

```python
def forward(self, x):
    out = F.relu(self.bn1(self.conv1(x)))
    out = self.layer1(out)
    out = self.layer2(out)
    out = self.layer3(out)
    # out = self.layer4(out)
    out = F.avg_pool2d(out, 4)
    out = out.view(out.size(0), -1)
    out = F.dropout(out, p=0.5, training=self.training)
    out = self.linear(out)
    return out
```

Figure 1: Architecture Modifications

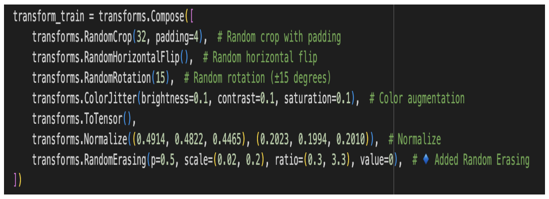


```
 Model Summary
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             864
       BatchNorm2d-2           [-1, 32, 32, 32]              64
            Conv2d-3           [-1, 96, 32, 32]          27,648
       BatchNorm2d-4           [-1, 96, 32, 32]             192
            Conv2d-5           [-1, 96, 32, 32]          82,944
       BatchNorm2d-6           [-1, 96, 32, 32]             192
            Conv2d-7           [-1, 96, 32, 32]           3,072
       BatchNorm2d-8           [-1, 96, 32, 32]             192
        BasicBlock-9           [-1, 96, 32, 32]               0
           Conv2d-10           [-1, 96, 32, 32]          82,944
      BatchNorm2d-11           [-1, 96, 32, 32]             192
           Conv2d-12           [-1, 96, 32, 32]          82,944
      BatchNorm2d-13           [-1, 96, 32, 32]             192
       BasicBlock-14           [-1, 96, 32, 32]               0
           Conv2d-15          [-1, 256, 16, 16]         221,184
      BatchNorm2d-16          [-1, 256, 16, 16]             512
           Conv2d-17          [-1, 256, 16, 16]         589,824
      BatchNorm2d-18          [-1, 256, 16, 16]             512
           Conv2d-19          [-1, 256, 16, 16]          24,576
      BatchNorm2d-20          [-1, 256, 16, 16]             512
       BasicBlock-21          [-1, 256, 16, 16]               0
           Conv2d-22          [-1, 256, 16, 16]         589,824
      BatchNorm2d-23          [-1, 256, 16, 16]             512
           Conv2d-24          [-1, 256, 16, 16]         589,824
      BatchNorm2d-25          [-1, 256, 16, 16]             512
       BasicBlock-26          [-1, 256, 16, 16]               0
           Conv2d-27            [-1, 256, 8, 8]         589,824
      BatchNorm2d-28            [-1, 256, 8, 8]             512
           Conv2d-29            [-1, 256, 8, 8]         589,824
      BatchNorm2d-30            [-1, 256, 8, 8]             512
           Conv2d-31            [-1, 256, 8, 8]          65,536
      BatchNorm2d-32            [-1, 256, 8, 8]             512
       BasicBlock-33            [-1, 256, 8, 8]               0
           Conv2d-34            [-1, 256, 8, 8]         589,824
      BatchNorm2d-35            [-1, 256, 8, 8]             512
           Conv2d-36            [-1, 256, 8, 8]         589,824
      BatchNorm2d-37            [-1, 256, 8, 8]             512
       BasicBlock-38            [-1, 256, 8, 8]               0
           Linear-39                   [-1, 10]          10,250
================================================================
Total params: 4,736,874
Trainable params: 4,736,874
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 17.00
Params size (MB): 18.07
Estimated Total Size (MB): 35.08
```

Figure 2: Model Layers

- Data Augmentation: After multiple iterations, we decided on the following data augmentation techniques to help prevent overfitting and increase the robustness of predictions.
  - Random cropping with padding was used to introduce spatial variation.

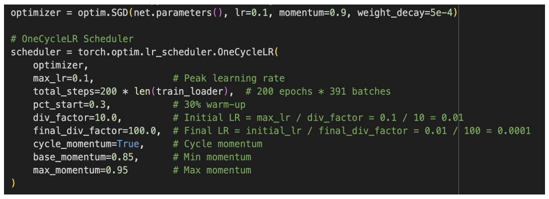
- Random horizontal flipping helped the model generalize to different orientations.
- Rotation up to 15 degrees added robustness against different image perspectives.
- Color jittering adjusted brightness, contrast, and saturation to simulate real-world variations.
- Normalization based on CIFAR-10 mean and standard deviation
- Random erasing with a 50% probability was introduced to improve model resilience against occlusions. These techniques significantly enhanced model robustness and improved performance on unseen data.



```python
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),  # Random crop with padding
    transforms.RandomHorizontalFlip(),  # Random horizontal flip
    transforms.RandomRotation(15),  # Random rotation (±15 degrees)
    transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1),  # Color augmentation
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),  # Normalize
    transforms.RandomErasing(p=0.5, scale=(0.02, 0.2), ratio=(0.3, 3.3), value=0),  # ♦ Added Random Erasing
])
```

Figure 3: Data Augmentation Techniques

- Learning Rate and Weight Decay:
  - We first tried training with the learning rate constant. This gave good results for lr = 0.05, but not above the kaggle benchmark accuracy. We then tried incorporating a custom decaying learning rate, with different step sizes and learning rates. These methods were good, but we observed that after a certain number of epochs the model stagnated, with the validation loss not reducing for many epochs. Finally, we tried cyclic learning rate with an initial learning rate of 0.1, which clearly showed better results than the previous learning rates, with the model learning fairly regularly throughout the 200 epochs of training.
  - When we were using constant and decaying learning rates, we decided to increase the weight decay to decrease overfitting. We tried with weight decay = 1e-3. However, when we tried cycling learning rate, the original weight decay of 5e-4 worked well, and we stuck with that for our submission.
- Training and Validation Accuracy Over Epochs:
  - This graph illustrates the accuracy of the model over 200 training epochs. The blue line represents the training accuracy, while the orange line represents the validation accuracy. Initially, both curves exhibit a steep rise, indicating rapid learning during the early epochs. Around epoch 50, the accuracy starts stabilizing, showing minor fluctuations due to the cyclic learning rate scheduler. The model eventually achieves a training accuracy of approximately 99% and a validation accuracy of around 96%, demonstrating strong generalization. The slight gap between training and validation accuracy suggests minimal overfitting, which is mitigated using dropout, data augmentation, and weight decay.
- Training and Validation Loss Over Epochs:
  - This plot shows the model's training loss (blue) and validation loss (orange) over 200 epochs. The steep decline at the beginning indicates rapid learning, with the loss decreasing significantly during the first 50 epochs. The model's loss continues to decline but at a slower rate as training progresses. The validation loss fluctuates more than the training loss, likely due to batch variations and augmentation effects. The OneCycleLR scheduler helps smoothen the convergence in later epochs. Around epoch 175, the validation loss stabilizes, indicating the model has reached optimal performance without significant overfitting.



```python
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4)

# OneCycleLR Scheduler
scheduler = torch.optim.lr_scheduler.OneCycleLR(
    optimizer,
    max_lr=0.1,              # Peak learning rate
    total_steps=200 * len(train_loader),  # 200 epochs * 391 batches
    pct_start=0.3,           # 30% warm-up
    div_factor=10.0,         # Initial LR = max_lr / div_factor = 0.1 / 10 = 0.01
    final_div_factor=100.0,  # Final LR = initial_lr / final_div_factor = 0.01 / 100 = 0.0001
    cycle_momentum=True,     # Cycle momentum
    base_momentum=0.85,      # Min momentum
    max_momentum=0.95        # Max momentum
)
```

Figure 4: Learning Rate and Weight Decay

Results

Final Model Performance

The best-performing model achieved:

- Training Accuracy: 99.19%
- Validation Accuracy: 96.26%
- Kaggle Submission Score: 0.8541
- Number of parameters: 4,736,874

Final Model Architecture

- **Input Layer:** $3 \times 32 \times 32$ RGB image input.
- **Initial Convolution:** A 3×3 convolutional layer with 32 filters, followed by batch normalization and ReLU activation.
- **Residual Blocks:**
  - **Layer 1:** Two residual blocks with 96 filters, stride 1.
  - **Layer 2:** Two residual blocks with 256 filters, stride 2 (reducing spatial dimensions).
  - **Layer 3:** Two residual blocks with 256 filters, stride 2.
- **Global Average Pooling:** 4×4 average pooling reduces feature maps to a 1024-dimensional vector.
- **Dropout Layer:** Applied with $p = 0.5$ to prevent overfitting.
- **Fully Connected Layer:** A final linear layer mapping 1024 features to 10 output classes.

Training Enhancements:

- **Loss Function:** Cross-entropy with label smoothing ($\alpha = 0.1$) to reduce overconfidence.
- **Optimizer:** Stochastic Gradient Descent (SGD) with momentum (0.9) and weight decay (5×10^{-4}).

- **Learning Rate Scheduler:** OneCycleLR with:
  - Peak Learning Rate: 0.1
  - Warm-up for first 30% of total training steps.
  - Final learning rate reduced to 0.0001.

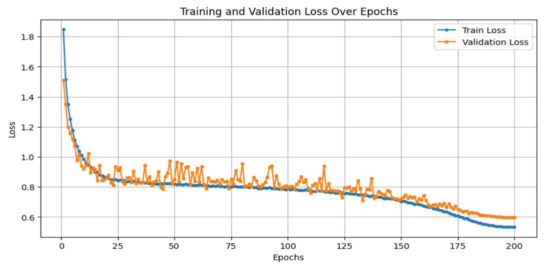


Figure 5: Training and Validation Loss

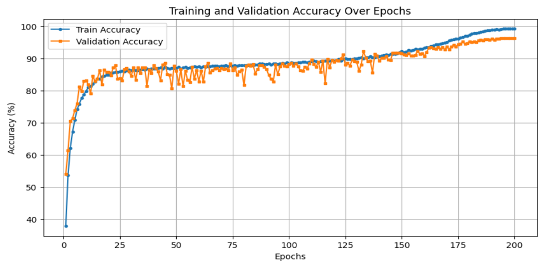


Figure 6: Training and Validation Accuracy

The training and validation loss plots demonstrate that our optimized ResNet-18 model achieved rapid convergence within the first 50 epochs, reflected by the steep decline in both curves. Beyond epoch 50, training continued to steadily reduce loss, indicating progressive learning, while validation loss exhibited fluctuations, likely due to the effects of data augmentation and batch variations. Notably, the training and validation accuracy plots closely mirror this trend, displaying rapid improvement initially, followed by gradual stabilization. The minimal divergence between the final training accuracy (99.19%) and validation accuracy (96.26%) suggests the effectiveness of our regularization strategies, including dropout, data augmentation, cyclic learning rate scheduling, and appropriate weight decay, effectively mitigating overfitting.

Conclusion

This work presents a modified ResNet-18 architecture that meets a 5 million parameter budget while achieving competitive performance on CIFAR-10. By reducing filter sizes, adjusting the number of layers, and incorporating dropout, our model achieves a training accuracy of 99.19% and a validation accuracy of 96.26%, along with a Kaggle score of 0.8541. Our experiments also demonstrate that cyclic learning rates and robust data augmentation are effective for steady convergence and minimizing overfitting.

Future work can explore integrating attention mechanisms and adaptive regularization to further improve efficiency and scalability. This approach offers a practical pathway for deploying high-performing deep learning models in resource-constrained settings.

References

[1] He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition. CVPR.

[2] Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS.

[3] Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. arXiv.

[4] OpenAI. (2025). ChatGPT [Large language model]. Retrieved from https://chat.openai.com/.