CIFAR-10 Image Classification with Deep Learning

Improved Model Architecture and Training Strategy

Anthony Angeles
Santa Clara University

February 3, 2026

a) Model Architecture and Loss Function

CIFAR-10 Data

Let's familiarize ourself with the data we used from PyTorch's documentation. We will use the CIFAR10 dataset. It has the classes: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'. The images in CIFAR-10 are of size 3x32x32, i.e. 3-channel color images of 32x32 pixels in size.

Figure 1: Sample Training Data

Figure 2: Ground Truth Images

Model Architecture: ResNet-34 with LeakyReLU Activation

The improved model is based on a **ResNet-34** architecture, modified for the CIFAR-10 dataset. The key architectural choices and improvements are:

1. ResNet-34 Deep Residual Network

The model uses a 34-layer residual network with BasicBlock building blocks. The layer configuration is $[3, 4, 6, 3]$, meaning 3 blocks in layer1, 4 blocks in layer2, 6 blocks in layer3, and 3 blocks in layer4. This results in approximately **21.3 million trainable parameters**.

2. CIFAR-10 Adaptation

Unlike standard ResNet designed for ImageNet (224×224), this implementation is adapted for CIFAR-10's 32×32 images by:

- Using a 3×3 initial convolution with stride=1 instead of a 7×7 convolution with stride=2
- Removing the initial max pooling layer

3. LeakyReLU Activation Function

The model replaces standard ReLU activations with LeakyReLU (negative slope = 0.1). This modification helps prevent the "dying ReLU" problem where neurons can become permanently inactive during training.

Loss Function: Cross-Entropy Loss

The Cross-Entropy Loss function is used for multi-class classification. For a batch of N samples with C classes ($C = 10$ for CIFAR-10)

Training Strategy and Hyperparameters

Data Augmentation

- RandomCrop (32×32 with 4-pixel padding)
- RandomHorizontalFlip ($p = 0.5$)
- RandomErasing/Cutout ($p = 0.5$, scale=0.02–0.33)
- Normalization: $\mu = (0.4914, 0.4822, 0.4465)$, $\sigma = (0.2470, 0.2435, 0.2616)$

Optimizer Configuration

- Optimizer: SGD with momentum
- Initial Learning Rate: 0.1122 (via LR Range Test)
- Momentum: 0.9
- Weight Decay: 5×10^{-4} (L2 regularization)
- LR Scheduler: Cosine Annealing ($T_{max} = 200$)

Training Configuration

- Batch Size: 512 • Number of Epochs: 200 • Mixed Precision Training (AMP): Enabled • Total Training Time: ∼15 minutes on NVIDIA GeForce RTX 5090

Performance Improvement Summary

Model	Test Accuracy	Parameters	Epochs
Basic CNN (2 conv, 3 fc)	67.35%	∼60K	100
ResNet-18 with LeakyReLU	86.34%	11.2M	100
ResNet-34 with LeakyReLU	**96.00%**	21.3M	200

Table 1: Comparison of model architectures and their performance on CIFAR-10

b) Code Implementation

How it works

When we go through the PyTorch documentation code and run it for the first time we hit about 75 percent accuracy, so the first initial steps were to increase batch size and epoch size while still keeping the original CNN that we implemented.

While looking into methods for improving accuracy and learning efficiency, I went ahead and made the model network deeper as well as switching to the ResNet Model, you will see that at first I went with ResNet-18 but later opted for a deeper network, ResNet-34.

Another aspect we changed from the original code is to allow the code itself to run on the GPU and take advantage of the hardware we had access to, an RTX FE 5090 with 32GB of memory.

After we started to change some of the original code to use our own implementations built of PyTorch starting example we saw a drop in accuracy, but that was to be expected as we needed to find the right methods to use in conjunction to reach our desire results, which in my case was a personal goal of at least 95 percent accuracy.

Where we saw the biggest jump in accuracy, and allowing us to reach 96 percent accuracy overall with some classes being 98, was when we switched from ResNet-18 to -34 and when we also introduced a method to optimize the learning rate with Leslie Smith Method. These methods in combination with also adding data augmentation while training, for example flipping the image or clipping it really showed big improvements. These were the biggest changes in my opinion that resulted in such an increase in accuracy, from 67 percent to 96 overall. The code for these demonstrations and the final implementation can be found in a GitHub repository found here: `https://github.com/aneangel/improving_CiFAR10`.

I must disclose that the methods for tracking multiple test sessions and plot generation, as well as, minor help with brainstorming how to implement the Leslie Smith Method were implemented in conjuction with Claude Sonnet 4.5.

ResNet Model Definition

```python
class BasicBlockLeakyReLU(nn.Module):
    expansion = 1

    def __init__(self, in_channels,
                 out_channels, stride=1,
                 downsample=None):
        super().__init__()
        self.conv1 = nn.Conv2d(
            in_channels, out_channels,
            kernel_size=3, stride=stride,
            padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(
    out_channels)
        self.leaky_relu = nn.LeakyReLU(
            negative_slope=0.1, inplace=True)
        self.conv2 = nn.Conv2d(
            out_channels, out_channels,
            kernel_size=3, stride=1,
            padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(
    out_channels)
        self.downsample = downsample

    def forward(self, x):
        identity = x
        out = self.leaky_relu(
            self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        if self.downsample is not None:
            identity = self.downsample(x)
        out += identity
        return self.leaky_relu(out)
```

BasicBlockLeakyReLU Class

```python
class ResNetLeakyReLU(nn.Module):
    def __init__(self, block, layers,
                 num_classes=10):
        super().__init__()
        self.in_channels = 64
        # CIFAR-10 adapted: 3x3 conv
        self.conv1 = nn.Conv2d(
            3, 64, kernel_size=3,
            stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.leaky_relu = nn.LeakyReLU(
            negative_slope=0.1, inplace=True)

        self.layer1 = self._make_layer(
            block, 64, layers[0], stride=1)
        self.layer2 = self._make_layer(
            block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(
            block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(
            block, 512, layers[3], stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d
    ((1,1))
        self.fc = nn.Linear(512, num_classes)

        # Kaiming initialization
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(
                    m.weight, mode='fan_out',
                    nonlinearity='leaky_relu'
    )

def resnet34_leaky():
    return ResNetLeakyReLU(
        BasicBlockLeakyReLU, [3, 4, 6, 3])
```

ResNetLeakyReLU Class

Data & Training Setup

```python
# CIFAR-10 statistics
cifar10_mean = (0.4914, 0.4822, 0.4465)
cifar10_std = (0.2470, 0.2435, 0.2616)

train_transform = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(
        cifar10_mean, cifar10_std),
    transforms.RandomErasing(
        p=0.5, scale=(0.02, 0.33),
        ratio=(0.3, 3.3))
])

test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(
        cifar10_mean, cifar10_std)
])
```

Data Augmentation Pipeline

```python
# Hyperparameters
learning_rate = 0.1122  # LR Range Test
momentum = 0.9
num_epochs = 200
weight_decay = 5e-4
batch_size = 512

# Model, Loss, Optimizer
net = resnet34_leaky().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(
    net.parameters(), lr=learning_rate,
    momentum=momentum,
    weight_decay=weight_decay)
scheduler = optim.lr_scheduler.
    CosineAnnealingLR(
    optimizer, T_max=num_epochs, eta_min=1e
    -4)

# Mixed Precision Training
scaler = torch.amp.GradScaler(
    'cuda', enabled=True)
```

Training Configuration

```python
for epoch in range(num_epochs):
    for inputs, labels in trainloader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()

        # AMP forward pass
        with torch.amp.autocast(
                'cuda', enabled=True):
            outputs = net(inputs)
            loss = criterion(outputs, labels)

        # Scaled backward pass
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

    scheduler.step()
```

Training Loop with AMP

Evaluation Code (Overall + Per-Class Accuracy)

```python
# Initialize counters
correct = 0
total = 0
class_correct = {classname: 0 for classname in classes}
class_total = {classname: 0 for classname in classes}

with torch.no_grad():
    for images, labels in testloader:
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        # Per-class accuracy
        for label, prediction in zip(labels, predicted):
            classname = classes[label]
            class_total[classname] += 1
            if label == prediction:
                class_correct[classname] += 1

test_accuracy = 100 * correct / total
print(f'Overall Accuracy: {test_accuracy:.2f}%')

# Display per-class accuracy
for classname in classes:
    acc = 100 * class_correct[classname] / class_total[classname]
    print(f'{classname}: {class_correct[classname]}/{class_total[classname]} = {acc:.2f}%')
```

c) Screenshots and Results

Overall Network Accuracy

The improved ResNet-34 with LeakyReLU achieved a test accuracy of **96.00%** on the CIFAR-10 dataset, correctly classifying 9,600 out of 10,000 test images. This represents a significant improvement from the baseline Basic CNN (67.35%) and the intermediate ResNet-18 model (86.34%).

Per-Class Accuracy Results

The per-class accuracy results demonstrate strong performance across all 10 CIFAR-10 classes, with accuracy ranging from 90.90% (cat) to 98.40% (car). These are the actual results from running the code:

Class	Correct / Total	Accuracy
plane	964 / 1000	96.40%
car	**984 / 1000**	**98.40%**
bird	950 / 1000	95.00%
cat	*909 / 1000*	*90.90%*
deer	970 / 1000	97.00%
dog	932 / 1000	93.20%
frog	974 / 1000	97.40%
horse	974 / 1000	97.40%
ship	972 / 1000	97.20%
truck	971 / 1000	97.10%

Table 2: Per-class accuracy. **Bold**: Best (car). *Italic*: Worst (cat).

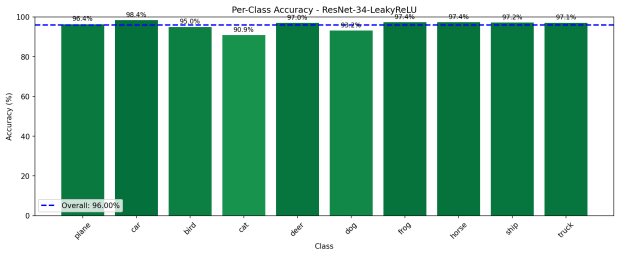

Figure 3: Per Class Accuracies

Best Class: car (98.40%) **Worst Class:** cat (90.90%)

Experiment Tracking Summary

Exp #	Timestamp	Model	Accuracy	LR	Epochs
2	2026-02-02 11:45	ResNet-34-LeakyReLU	**96.00%**	0.1122	200
1	2026-02-02 10:22	ResNet-18-LeakyReLU	86.34%	0.0100	100
0	2026-02-02 09:59	Basic CNN	67.35%	0.0010	100

Table 3: Experiment tracking showing progressive improvement in model accuracy