Homework-2

Fahrettin Ege Bilge

21070001052

Task1: Bottleneck Residual Block in ResNet

Design of Bottleneck Residual Block

A bottleneck residual block is a variant of the basic residual block used in ResNet architectures. It involves a sequence of layers that perform downsampling, convolution, and upsampling, followed by a residual connection. Below is the structure of a bottleneck block:

1x1 Convolution: Reduces the dimensionality (number of channels) for computational efficiency. 3x3 Convolution: Applies the main convolution operation. 1x1 Convolution: Restores the dimensionality back to the original. Each convolutional layer is followed by Batch Normalization (BN) and ReLU activation. The residual connection adds the input to the output of these layers.

```
Input
  |
 [1x1 Conv] --> BN --> ReLU
  |
 [3x3 Conv] --> BN --> ReLU
  |
 [1x1 Conv] --> BN
  |
 [Add] --> ReLU
  |
Output
```

Structure:

1.Convolutional Layer with 1x1 filters (Compression layer):

- The purpose of this layer is to reduce the dimensionality of the input feature maps, thus lowering the computational cost.
- Typically followed by a batch normalization layer and a rectified linear unit (ReLU) activation function.

2.Convolutional Layer with 3x3 filters:

- This layer applies a standard convolution operation to the feature maps obtained from the compression layer.
- Followed by batch normalization and ReLU activation.

3.Convolutional Layer with 1x1 filters (Expansion layer):

- This layer expands the dimensionality of the feature maps back to the original or desired dimensionality.
- Followed by batch normalization but **no** activation function.

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split
import torchvision
from torchvision import models, transforms, datasets
import matplotlib.pyplot as plt
import numpy as np
import time
```

In [ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [ ]:
```
# Load and prepare the dataset
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0
trn_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
vld_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
tst_dataset = datasets.CIFAR10(root='./data', train=False, download=True,
```

```
Files already downloaded and verified
Files already downloaded and verified
Files already downloaded and verified
```

In [ ]:
```
# Split the training set into training and validation partitions
trn_size = int(0.8 * len(trn_dataset))
vld_size = len(trn_dataset) - trn_size
torch.manual_seed(0)
trn_dataset, vld_dataset = random_split(trn_dataset, [trn_size, vld_size]
classes = 'Airplane', 'Car', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse
num_classes = len(classes)
batch_size = 128
trn_loader = DataLoader(trn_dataset, batch_size=batch_size, shuffle=True,
vld_loader = DataLoader(vld_dataset, batch_size=batch_size, shuffle=False
tst_loader = DataLoader(tst_dataset, batch_size=batch_size, shuffle=False
```

In [ ]:
```
# Define the bottleneck residual block
class BottleneckBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(BottleneckBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=1,
```

```python
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.conv3 = nn.Conv2d(out_channels, out_channels * 4, kernel_siz
        self.bn3 = nn.BatchNorm2d(out_channels * 4)

        self.relu = nn.ReLU(inplace=True)
        self.downsample = None
        if stride != 1 or in_channels != out_channels * 4:
            self.downsample = nn.Sequential(
                nn.Conv2d(in_channels, out_channels * 4, kernel_size=1, s
                nn.BatchNorm2d(out_channels * 4)
            )

    def forward(self, x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out
```

In [ ]:

```python
# Define the CNN model with bottleneck blocks
class CustomResNet(nn.Module):
    def __init__(self, num_classes=10):
        super(CustomResNet, self).__init__()
        self.in_channels = 64

        self.conv1 = nn.Conv2d(3, self.in_channels, kernel_size=7, stride
        self.bn1 = nn.BatchNorm2d(self.in_channels)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = self._make_layer(64, 3)
        self.layer2 = self._make_layer(128, 4, stride=2)
        self.layer3 = self._make_layer(256, 6, stride=2)
        self.layer4 = self._make_layer(512, 3, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * 4, num_classes)

    def _make_layer(self, out_channels, blocks, stride=1):
        layers = []
```

```python
        layers.append(BottleneckBlock(self.in_channels, out_channels, str
        self.in_channels = out_channels * 4
        for _ in range(1, blocks):
            layers.append(BottleneckBlock(self.in_channels, out_channels))
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

In [ ]:
```python
# Initialize model, criterion, and optimizer
model = CustomResNet(num_classes=10).to(device)
criterion = nn.CrossEntropyLoss()

# SGD with momentum
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9, weight_d
```

In [ ]:
```python
import time

def evaluate_model(model, testloader, criterion):
    model.eval()
    correct = 0
    total = 0
    running_loss = 0.0

    with torch.no_grad():
        for inputs, labels in testloader:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss.item() * inputs.size(0)

            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    avg_loss = running_loss / len(testloader.dataset)
    return avg_loss, accuracy

# Training function
def train_model(model, criterion, optimizer, train_loader, test_loader, n
    train_loss, train_acc = [], []
```

```python
    val_loss, val_acc = [], []

    for epoch in range(num_epochs):
        start_time = time.time()  # Start time of the epoch

        model.train()
        running_loss, correct, total = 0.0, 0, 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

        epoch_loss = running_loss / total
        epoch_acc = 100. * correct / total
        train_loss.append(epoch_loss)
        train_acc.append(epoch_acc)

        val_epoch_loss, val_epoch_acc = evaluate_model(model, test_loader
        val_loss.append(val_epoch_loss)
        val_acc.append(val_epoch_acc)

        end_time = time.time()  # End time of the epoch
        epoch_duration = end_time - start_time  # Duration of the epoch

        print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss[-

    return train_loss, train_acc, val_loss, val_acc
```

In [ ]:

```python
# Train the model
train_loss, train_acc, val_loss, val_acc = train_model(model,criterion,op
```

```
Epoch [1/20], Train Loss: 1.8882, Train Acc: 31.5800, Val Loss: 1.9297, Va
l Acc: 40.8100, Time: 66.15s
Epoch [2/20], Train Loss: 1.3432, Train Acc: 51.1150, Val Loss: 1.3381, Va
l Acc: 55.4900, Time: 66.22s
Epoch [3/20], Train Loss: 1.0255, Train Acc: 63.2725, Val Loss: 2.2532, Va
l Acc: 60.5100, Time: 66.21s
Epoch [4/20], Train Loss: 0.8122, Train Acc: 71.4175, Val Loss: 1.0896, Va
l Acc: 66.1500, Time: 66.43s
Epoch [5/20], Train Loss: 0.6345, Train Acc: 77.5325, Val Loss: 1.0777, Va
l Acc: 71.1600, Time: 66.34s
Epoch [6/20], Train Loss: 0.5075, Train Acc: 82.2650, Val Loss: 0.8778, Va
l Acc: 71.5900, Time: 66.35s
Epoch [7/20], Train Loss: 0.3922, Train Acc: 86.1875, Val Loss: 0.8714, Va
l Acc: 75.5000, Time: 66.46s
Epoch [8/20], Train Loss: 0.3034, Train Acc: 89.3075, Val Loss: 0.9066, Va
l Acc: 75.8300, Time: 66.36s
Epoch [9/20], Train Loss: 0.2146, Train Acc: 92.3975, Val Loss: 0.8682, Va
l Acc: 76.3300, Time: 66.38s
Epoch [10/20], Train Loss: 0.1830, Train Acc: 93.5375, Val Loss: 0.8449, V
al Acc: 78.9300, Time: 66.42s
Epoch [11/20], Train Loss: 0.1285, Train Acc: 95.4575, Val Loss: 0.9975, V
al Acc: 76.3600, Time: 66.41s
Epoch [12/20], Train Loss: 0.1070, Train Acc: 96.2125, Val Loss: 0.9837, V
al Acc: 77.7700, Time: 66.36s
Epoch [13/20], Train Loss: 0.0926, Train Acc: 96.8200, Val Loss: 1.0929, V
al Acc: 76.9400, Time: 66.47s
Epoch [14/20], Train Loss: 0.0703, Train Acc: 97.5975, Val Loss: 1.1711, V
al Acc: 77.6400, Time: 66.48s
Epoch [15/20], Train Loss: 0.0603, Train Acc: 97.8550, Val Loss: 1.0832, V
al Acc: 78.5100, Time: 66.35s
Epoch [16/20], Train Loss: 0.0526, Train Acc: 98.1400, Val Loss: 0.9550, V
al Acc: 79.4700, Time: 66.40s
Epoch [17/20], Train Loss: 0.0470, Train Acc: 98.4300, Val Loss: 1.4590, V
al Acc: 74.1800, Time: 66.39s
Epoch [18/20], Train Loss: 0.0457, Train Acc: 98.3950, Val Loss: 0.9686, V
al Acc: 80.4100, Time: 66.48s
Epoch [19/20], Train Loss: 0.0333, Train Acc: 98.8950, Val Loss: 1.1286, V
al Acc: 78.9600, Time: 66.55s
Epoch [20/20], Train Loss: 0.0288, Train Acc: 99.0600, Val Loss: 1.1242, V
al Acc: 79.3600, Time: 66.47s
```

In [ ]:

```python
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Save Model State Dict
import torch
# Assuming model is your trained PyTorch model
torch.save(model.state_dict(), '/content/drive/My Drive/std_model.pth')

# Load Model State Dict
# Assuming CustomResNet is the model class and num_classes is the same as
#model = CustomResNet(num_classes=10).to(device)
#model.load_state_dict(torch.load('/content/drive/My Drive/my_model.pth')
#model.eval()  # Set the model to evaluation mode
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

In [ ]:

```python
# Plot the learning curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss Curves with SGD')

plt.subplot(1, 2, 2)
plt.plot(train_acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy Curves with SGD')

plt.show()
```

Training Summary

The model appears to have learned from the training data as the training loss 0.0288 decreased and training accuracy 99.0600 increased over epochs.

However, there are signs of overfitting. While the validation loss 0.9686 also decreased, it did not decrease as significantly as the training loss. This suggests the model may be learning patterns specific to the training data that don't generalize well to unseen data.

I should onsider using techniques to mitigate overfitting, such as early stopping.

Best parameters for this training was:

- lr=0.01
- momentum=0.9
- weight_decay=1e-4

In [ ]:
```
try:
  from torchinfo import summary
except:
  print("[INFO] Couldn't find torchinfo... installing it.")
  !pip install -q torchinfo
```

```
[INFO] Couldn't find torchinfo... installing it.
```

In [ ]:
```
from torchinfo import summary
model_stats = summary(model)
print(model_stats)
```

```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
CustomResNet                             --
├─Conv2d: 1-1                            9,408
├─BatchNorm2d: 1-2                       128
├─ReLU: 1-3                              --
├─MaxPool2d: 1-4                         --
├─Sequential: 1-5                        --
│    └─BottleneckBlock: 2-1              --
│    │    └─Conv2d: 3-1                  4,096
│    │    └─BatchNorm2d: 3-2             128
│    │    └─Conv2d: 3-3                  36,864
│    │    └─BatchNorm2d: 3-4             128
│    │    └─Conv2d: 3-5                  16,384
│    │    └─BatchNorm2d: 3-6             512
│    │    └─ReLU: 3-7                    --
│    │    └─Sequential: 3-8              16,896
│    └─BottleneckBlock: 2-2              --
│    │    └─Conv2d: 3-9                  16,384
│    │    └─BatchNorm2d: 3-10            128
│    │    └─Conv2d: 3-11                 36,864
│    │    └─BatchNorm2d: 3-12            128
│    │    └─Conv2d: 3-13                 16,384
│    │    └─BatchNorm2d: 3-14            512
│    │    └─ReLU: 3-15                   --
│    └─BottleneckBlock: 2-3              --
│    │    └─Conv2d: 3-16                 16,384
│    │    └─BatchNorm2d: 3-17            128
│    │    └─Conv2d: 3-18                 36,864
│    │    └─BatchNorm2d: 3-19            128
│    │    └─Conv2d: 3-20                 16,384
│    │    └─BatchNorm2d: 3-21            512
│    │    └─ReLU: 3-22                   --
├─Sequential: 1-6                        --
│    └─BottleneckBlock: 2-4              --
│    │    └─Conv2d: 3-23                 32,768
│    │    └─BatchNorm2d: 3-24            256
│    │    └─Conv2d: 3-25                 147,456
│    │    └─BatchNorm2d: 3-26            256
│    │    └─Conv2d: 3-27                 65,536
│    │    └─BatchNorm2d: 3-28            1,024
│    │    └─ReLU: 3-29                   --
│    │    └─Sequential: 3-30             132,096
│    └─BottleneckBlock: 2-5              --
```

```
│    │    └─Conv2d: 3-31                    65,536
│    │    └─BatchNorm2d: 3-32               256
│    │    └─Conv2d: 3-33                    147,456
│    │    └─BatchNorm2d: 3-34               256
│    │    └─Conv2d: 3-35                    65,536
│    │    └─BatchNorm2d: 3-36               1,024
│    │    └─ReLU: 3-37                      --
│    └─BottleneckBlock: 2-6                 --
│    │    └─Conv2d: 3-38                    65,536
│    │    └─BatchNorm2d: 3-39               256
│    │    └─Conv2d: 3-40                    147,456
│    │    └─BatchNorm2d: 3-41               256
│    │    └─Conv2d: 3-42                    65,536
│    │    └─BatchNorm2d: 3-43               1,024
│    │    └─ReLU: 3-44                      --
│    └─BottleneckBlock: 2-7                 --
│    │    └─Conv2d: 3-45                    65,536
│    │    └─BatchNorm2d: 3-46               256
│    │    └─Conv2d: 3-47                    147,456
│    │    └─BatchNorm2d: 3-48               256
│    │    └─Conv2d: 3-49                    65,536
│    │    └─BatchNorm2d: 3-50               1,024
│    │    └─ReLU: 3-51                      --
├─Sequential: 1-7                           --
│    └─BottleneckBlock: 2-8                 --
│    │    └─Conv2d: 3-52                    131,072
│    │    └─BatchNorm2d: 3-53               512
│    │    └─Conv2d: 3-54                    589,824
│    │    └─BatchNorm2d: 3-55               512
│    │    └─Conv2d: 3-56                    262,144
│    │    └─BatchNorm2d: 3-57               2,048
│    │    └─ReLU: 3-58                      --
│    │    └─Sequential: 3-59                526,336
│    └─BottleneckBlock: 2-9                 --
│    │    └─Conv2d: 3-60                    262,144
│    │    └─BatchNorm2d: 3-61               512
│    │    └─Conv2d: 3-62                    589,824
│    │    └─BatchNorm2d: 3-63               512
│    │    └─Conv2d: 3-64                    262,144
│    │    └─BatchNorm2d: 3-65               2,048
│    │    └─ReLU: 3-66                      --
│    └─BottleneckBlock: 2-10                --
│    │    └─Conv2d: 3-67                    262,144
│    │    └─BatchNorm2d: 3-68               512
│    │    └─Conv2d: 3-69                    589,824
│    │    └─BatchNorm2d: 3-70               512
│    │    └─Conv2d: 3-71                    262,144
│    │    └─BatchNorm2d: 3-72               2,048
│    │    └─ReLU: 3-73                      --
│    └─BottleneckBlock: 2-11                --
│    │    └─Conv2d: 3-74                    262,144
│    │    └─BatchNorm2d: 3-75               512
│    │    └─Conv2d: 3-76                    589,824
│    │    └─BatchNorm2d: 3-77               512
│    │    └─Conv2d: 3-78                    262,144
│    │    └─BatchNorm2d: 3-79               2,048
│    │    └─ReLU: 3-80                      --
```

```
│    └─BottleneckBlock: 2-12              --
│    │    └─Conv2d: 3-81                  262,144
│    │    └─BatchNorm2d: 3-82             512
│    │    └─Conv2d: 3-83                  589,824
│    │    └─BatchNorm2d: 3-84             512
│    │    └─Conv2d: 3-85                  262,144
│    │    └─BatchNorm2d: 3-86             2,048
│    │    └─ReLU: 3-87                    --
│    └─BottleneckBlock: 2-13              --
│    │    └─Conv2d: 3-88                  262,144
│    │    └─BatchNorm2d: 3-89             512
│    │    └─Conv2d: 3-90                  589,824
│    │    └─BatchNorm2d: 3-91             512
│    │    └─Conv2d: 3-92                  262,144
│    │    └─BatchNorm2d: 3-93             2,048
│    │    └─ReLU: 3-94                    --
├─Sequential: 1-8                         --
│    └─BottleneckBlock: 2-14              --
│    │    └─Conv2d: 3-95                  524,288
│    │    └─BatchNorm2d: 3-96             1,024
│    │    └─Conv2d: 3-97                  2,359,296
│    │    └─BatchNorm2d: 3-98             1,024
│    │    └─Conv2d: 3-99                  1,048,576
│    │    └─BatchNorm2d: 3-100            4,096
│    │    └─ReLU: 3-101                   --
│    │    └─Sequential: 3-102             2,101,248
│    └─BottleneckBlock: 2-15              --
│    │    └─Conv2d: 3-103                 1,048,576
│    │    └─BatchNorm2d: 3-104            1,024
│    │    └─Conv2d: 3-105                 2,359,296
│    │    └─BatchNorm2d: 3-106            1,024
│    │    └─Conv2d: 3-107                 1,048,576
│    │    └─BatchNorm2d: 3-108            4,096
│    │    └─ReLU: 3-109                   --
│    └─BottleneckBlock: 2-16              --
│    │    └─Conv2d: 3-110                 1,048,576
│    │    └─BatchNorm2d: 3-111            1,024
│    │    └─Conv2d: 3-112                 2,359,296
│    │    └─BatchNorm2d: 3-113            1,024
│    │    └─Conv2d: 3-114                 1,048,576
│    │    └─BatchNorm2d: 3-115            4,096
│    │    └─ReLU: 3-116                   --
├─AdaptiveAvgPool2d: 1-9                  --
├─Linear: 1-10                            20,490
=================================================================
Total params: 23,528,522
Trainable params: 23,528,522
Non-trainable params: 0
=================================================================
```

In [ ]:

```python
def imshow(img):
    img = img / 2 + 0.5   # unnormalize
    npimg = img.numpy()
    # Clip the data to be between 0 and 1
    npimg = np.clip(npimg, 0, 1)
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

```python
def visualize_model_predictions(model, loader, num_images=6, classes=None
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()
    with torch.no_grad():
        for i, (inputs, labels) in enumerate(loader):
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images // 2, 2, images_so_far)
                ax.axis('off')
                ax.set_title(f'True: {classes[labels[j].item()]} | Pred:
                imshow(inputs.cpu().data[j])
                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
    model.train(mode=was_training)

visualize_model_predictions(model, loader=tst_loader, classes=classes)
```

True: Cat | Pred: Dog

True: Ship | Pred: Ship

True: Ship | Pred: Ship

True: Airplane | Pred: Airplane

True: Frog | Pred: Frog

True: Frog | Pred: Frog

Task2: Different Training Dynamics

(a)Using ADAM optimizer with weight decay

I will modify the optimizer to use ADAM with weight decay and compare the performance.

In [ ]:

```
# Using ADAM optimizer with weight decay
AdamModel = CustomResNet(num_classes=10).to(device)
criterion = nn.CrossEntropyLoss()
AdamOptimizer = optim.Adam(AdamModel.parameters(), lr=0.01, weight_decay=
adam_train_loss, adam_train_acc, adam_val_loss, adam_val_acc = train_mode
```

```
Epoch [1/20], Train Loss: 2.0896, Train Acc: 25.2425, Val Loss: 2.0030, Va
l Acc: 22.3600, Time: 66.83s
Epoch [2/20], Train Loss: 1.6330, Train Acc: 38.3550, Val Loss: 1.7551, Va
l Acc: 34.1100, Time: 66.55s
Epoch [3/20], Train Loss: 1.3816, Train Acc: 49.3800, Val Loss: 1.5931, Va
l Acc: 41.8200, Time: 66.70s
Epoch [4/20], Train Loss: 1.1992, Train Acc: 56.7650, Val Loss: 1.3544, Va
l Acc: 52.0300, Time: 66.73s
Epoch [5/20], Train Loss: 1.0839, Train Acc: 61.2900, Val Loss: 1.0765, Va
l Acc: 61.8200, Time: 66.67s
Epoch [6/20], Train Loss: 1.0142, Train Acc: 63.6700, Val Loss: 1.2664, Va
l Acc: 55.0300, Time: 66.60s
Epoch [7/20], Train Loss: 0.9648, Train Acc: 65.5625, Val Loss: 1.3130, Va
l Acc: 57.5000, Time: 66.64s
Epoch [8/20], Train Loss: 0.9271, Train Acc: 66.8850, Val Loss: 1.0579, Va
l Acc: 62.9600, Time: 66.64s
Epoch [9/20], Train Loss: 0.8945, Train Acc: 68.0900, Val Loss: 0.9865, Va
l Acc: 65.4900, Time: 66.81s
Epoch [10/20], Train Loss: 0.8635, Train Acc: 69.6200, Val Loss: 0.9367, V
al Acc: 67.1500, Time: 66.63s
Epoch [11/20], Train Loss: 0.8308, Train Acc: 70.6300, Val Loss: 0.9581, V
al Acc: 65.7700, Time: 66.62s
Epoch [12/20], Train Loss: 0.8113, Train Acc: 71.0975, Val Loss: 1.0567, V
al Acc: 62.8800, Time: 66.58s
Epoch [13/20], Train Loss: 0.7997, Train Acc: 71.6750, Val Loss: 0.8470, V
al Acc: 70.3800, Time: 66.54s
Epoch [14/20], Train Loss: 0.7735, Train Acc: 72.3325, Val Loss: 0.9366, V
al Acc: 67.3700, Time: 66.60s
Epoch [15/20], Train Loss: 0.7548, Train Acc: 72.9875, Val Loss: 1.0889, V
al Acc: 62.8400, Time: 66.59s
Epoch [16/20], Train Loss: 0.7455, Train Acc: 73.6675, Val Loss: 0.9206, V
al Acc: 67.6800, Time: 66.65s
Epoch [17/20], Train Loss: 0.7308, Train Acc: 74.2625, Val Loss: 0.8807, V
al Acc: 69.5500, Time: 66.55s
Epoch [18/20], Train Loss: 0.7146, Train Acc: 74.6425, Val Loss: 0.8639, V
al Acc: 69.6800, Time: 66.53s
Epoch [19/20], Train Loss: 0.7081, Train Acc: 74.8250, Val Loss: 0.9007, V
al Acc: 68.7400, Time: 66.74s
Epoch [20/20], Train Loss: 0.7013, Train Acc: 75.2125, Val Loss: 1.0005, V
al Acc: 66.5100, Time: 66.53s
```

In [ ]:

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Save Model State Dict
import torch
# Assuming model is your trained PyTorch model
torch.save(AdamModel.state_dict(), '/content/drive/My Drive/adam_model.pt

# Load Model State Dict
# Assuming CustomResNet is the model class and num_classes is the same as
#model = CustomResNet(num_classes=10).to(device)
#model.load_state_dict(torch.load('/content/drive/My Drive/my_model.pth')
#model.eval()  # Set the model to evaluation mode
```

```
Mounted at /content/drive
```

```
In [ ]: adam_model_stats = summary(AdamModel)
        print(adam_model_stats)
```

```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
CustomResNet                             --
├─Conv2d: 1-1                            9,408
├─BatchNorm2d: 1-2                       128
├─ReLU: 1-3                              --
├─MaxPool2d: 1-4                         --
├─Sequential: 1-5                        --
│    └─BottleneckBlock: 2-1              --
│    │    └─Conv2d: 3-1                  4,096
│    │    └─BatchNorm2d: 3-2             128
│    │    └─Conv2d: 3-3                  36,864
│    │    └─BatchNorm2d: 3-4             128
│    │    └─Conv2d: 3-5                  16,384
│    │    └─BatchNorm2d: 3-6             512
│    │    └─ReLU: 3-7                    --
│    │    └─Sequential: 3-8              16,896
│    └─BottleneckBlock: 2-2              --
│    │    └─Conv2d: 3-9                  16,384
│    │    └─BatchNorm2d: 3-10            128
│    │    └─Conv2d: 3-11                 36,864
│    │    └─BatchNorm2d: 3-12            128
│    │    └─Conv2d: 3-13                 16,384
│    │    └─BatchNorm2d: 3-14            512
│    │    └─ReLU: 3-15                   --
│    └─BottleneckBlock: 2-3              --
│    │    └─Conv2d: 3-16                 16,384
│    │    └─BatchNorm2d: 3-17            128
│    │    └─Conv2d: 3-18                 36,864
│    │    └─BatchNorm2d: 3-19            128
│    │    └─Conv2d: 3-20                 16,384
│    │    └─BatchNorm2d: 3-21            512
│    │    └─ReLU: 3-22                   --
├─Sequential: 1-6                        --
│    └─BottleneckBlock: 2-4              --
│    │    └─Conv2d: 3-23                 32,768
│    │    └─BatchNorm2d: 3-24            256
│    │    └─Conv2d: 3-25                 147,456
│    │    └─BatchNorm2d: 3-26            256
│    │    └─Conv2d: 3-27                 65,536
│    │    └─BatchNorm2d: 3-28            1,024
│    │    └─ReLU: 3-29                   --
│    │    └─Sequential: 3-30             132,096
│    └─BottleneckBlock: 2-5              --
│    │    └─Conv2d: 3-31                 65,536
│    │    └─BatchNorm2d: 3-32            256
│    │    └─Conv2d: 3-33                 147,456
│    │    └─BatchNorm2d: 3-34            256
│    │    └─Conv2d: 3-35                 65,536
│    │    └─BatchNorm2d: 3-36            1,024
│    │    └─ReLU: 3-37                   --
│    └─BottleneckBlock: 2-6              --
│    │    └─Conv2d: 3-38                 65,536
```

```
│    │    └─BatchNorm2d: 3-39             256
│    │    └─Conv2d: 3-40                  147,456
│    │    └─BatchNorm2d: 3-41             256
│    │    └─Conv2d: 3-42                  65,536
│    │    └─BatchNorm2d: 3-43             1,024
│    │    └─ReLU: 3-44                    --
│    └─BottleneckBlock: 2-7               --
│    │    └─Conv2d: 3-45                  65,536
│    │    └─BatchNorm2d: 3-46             256
│    │    └─Conv2d: 3-47                  147,456
│    │    └─BatchNorm2d: 3-48             256
│    │    └─Conv2d: 3-49                  65,536
│    │    └─BatchNorm2d: 3-50             1,024
│    │    └─ReLU: 3-51                    --
├─Sequential: 1-7                         --
│    └─BottleneckBlock: 2-8               --
│    │    └─Conv2d: 3-52                  131,072
│    │    └─BatchNorm2d: 3-53             512
│    │    └─Conv2d: 3-54                  589,824
│    │    └─BatchNorm2d: 3-55             512
│    │    └─Conv2d: 3-56                  262,144
│    │    └─BatchNorm2d: 3-57             2,048
│    │    └─ReLU: 3-58                    --
│    │    └─Sequential: 3-59              526,336
│    └─BottleneckBlock: 2-9               --
│    │    └─Conv2d: 3-60                  262,144
│    │    └─BatchNorm2d: 3-61             512
│    │    └─Conv2d: 3-62                  589,824
│    │    └─BatchNorm2d: 3-63             512
│    │    └─Conv2d: 3-64                  262,144
│    │    └─BatchNorm2d: 3-65             2,048
│    │    └─ReLU: 3-66                    --
│    └─BottleneckBlock: 2-10              --
│    │    └─Conv2d: 3-67                  262,144
│    │    └─BatchNorm2d: 3-68             512
│    │    └─Conv2d: 3-69                  589,824
│    │    └─BatchNorm2d: 3-70             512
│    │    └─Conv2d: 3-71                  262,144
│    │    └─BatchNorm2d: 3-72             2,048
│    │    └─ReLU: 3-73                    --
│    └─BottleneckBlock: 2-11              --
│    │    └─Conv2d: 3-74                  262,144
│    │    └─BatchNorm2d: 3-75             512
│    │    └─Conv2d: 3-76                  589,824
│    │    └─BatchNorm2d: 3-77             512
│    │    └─Conv2d: 3-78                  262,144
│    │    └─BatchNorm2d: 3-79             2,048
│    │    └─ReLU: 3-80                    --
│    └─BottleneckBlock: 2-12              --
│    │    └─Conv2d: 3-81                  262,144
│    │    └─BatchNorm2d: 3-82             512
│    │    └─Conv2d: 3-83                  589,824
│    │    └─BatchNorm2d: 3-84             512
│    │    └─Conv2d: 3-85                  262,144
│    │    └─BatchNorm2d: 3-86             2,048
│    │    └─ReLU: 3-87                    --
│    └─BottleneckBlock: 2-13              --
```

```
│    │    └─Conv2d: 3-88                  262,144
│    │    └─BatchNorm2d: 3-89             512
│    │    └─Conv2d: 3-90                  589,824
│    │    └─BatchNorm2d: 3-91             512
│    │    └─Conv2d: 3-92                  262,144
│    │    └─BatchNorm2d: 3-93             2,048
│    │    └─ReLU: 3-94                    --
├─Sequential: 1-8                         --
│    └─BottleneckBlock: 2-14              --
│    │    └─Conv2d: 3-95                  524,288
│    │    └─BatchNorm2d: 3-96             1,024
│    │    └─Conv2d: 3-97                  2,359,296
│    │    └─BatchNorm2d: 3-98             1,024
│    │    └─Conv2d: 3-99                  1,048,576
│    │    └─BatchNorm2d: 3-100            4,096
│    │    └─ReLU: 3-101                   --
│    │    └─Sequential: 3-102             2,101,248
│    └─BottleneckBlock: 2-15              --
│    │    └─Conv2d: 3-103                 1,048,576
│    │    └─BatchNorm2d: 3-104            1,024
│    │    └─Conv2d: 3-105                 2,359,296
│    │    └─BatchNorm2d: 3-106            1,024
│    │    └─Conv2d: 3-107                 1,048,576
│    │    └─BatchNorm2d: 3-108            4,096
│    │    └─ReLU: 3-109                   --
│    └─BottleneckBlock: 2-16              --
│    │    └─Conv2d: 3-110                 1,048,576
│    │    └─BatchNorm2d: 3-111            1,024
│    │    └─Conv2d: 3-112                 2,359,296
│    │    └─BatchNorm2d: 3-113            1,024
│    │    └─Conv2d: 3-114                 1,048,576
│    │    └─BatchNorm2d: 3-115            4,096
│    │    └─ReLU: 3-116                   --
├─AdaptiveAvgPool2d: 1-9                  --
├─Linear: 1-10                            20,490
=================================================================
Total params: 23,528,522
Trainable params: 23,528,522
Non-trainable params: 0
=================================================================
```

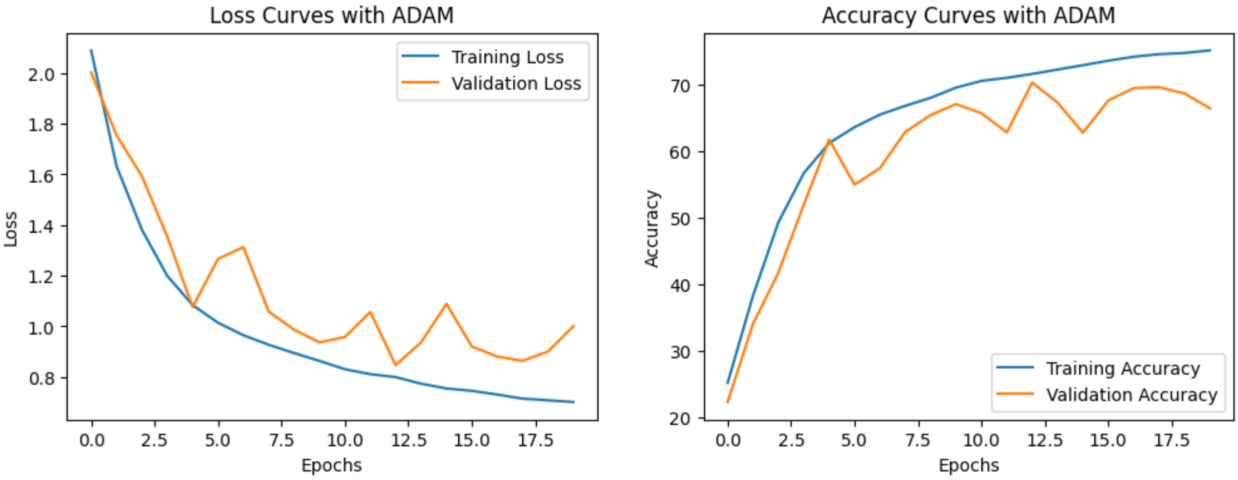
In [ ]:

```python
# Plot the learning curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(adam_train_loss, label='Training Loss')
plt.plot(adam_val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss Curves with ADAM')

plt.subplot(1, 2, 2)
plt.plot(adam_train_acc, label='Training Accuracy')
plt.plot(adam_val_acc, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```python
plt.title('Accuracy Curves with ADAM')

plt.show()
```



In [ ]:
```python
visualize_model_predictions(AdamModel, loader=tst_loader, classes=classes
```

True: Cat | Pred: Cat

True: Ship | Pred: Ship

True: Ship | Pred: Ship

True: Airplane | Pred: Airplane

True: Frog | Pred: Frog

True: Frog | Pred: Frog

The Adam optimizer likely performed better due to its ability to mitigate overfitting during training.

(b) Learning Rate Scheduling with ReduceLROnPlateau

I will add a learning rate scheduler to dynamically adjust the learning rate based on validation performance.

In [ ]:
```
# Learning rate scheduling with ReduceLROnPlateau
from torch.optim.lr_scheduler import ReduceLROnPlateau

# Define the scheduler
scheduler = ReduceLROnPlateau(AdamOptimizer, mode='max', factor=0.1, pati

# Train the model with scheduler
def train_model_with_scheduler(model, criterion, optimizer, scheduler, tr
    train_loss, train_acc = [], []
    val_loss, val_acc = [], []

    for epoch in range(num_epochs):
        start_time = time.time()  # Start time of the epoch

        model.train()
        running_loss, correct, total = 0.0, 0, 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            _, predicted = outputs.max(1)
            total += labels.size(0)
```

```
            correct += predicted.eq(labels).sum().item()

        epoch_loss = running_loss / total
        epoch_acc = 100. * correct / total
        train_loss.append(epoch_loss)
        train_acc.append(epoch_acc)

        val_epoch_loss, val_epoch_acc = evaluate_model(model, test_loader
        val_loss.append(val_epoch_loss)
        val_acc.append(val_epoch_acc)

        scheduler.step(val_epoch_acc)  # Step the scheduler based on vali

        end_time = time.time()  # End time of the epoch
        epoch_duration = end_time - start_time  # Duration of the epoch

        # Get the current learning rate
        current_lr = optimizer.param_groups[0]["lr"]

        print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss[-

    return train_loss, train_acc, val_loss, val_acc

# Train the model with scheduler
adam_train_loss_sched, adam_train_acc_sched, adam_val_loss_sched, adam_va
```

```
Epoch [1/20], Train Loss: 0.6830, Train Acc: 75.8125, Val Loss: 0.9713, Va
l Acc: 66.7700, Time: 66.51s, LR: 0.010000
Epoch [2/20], Train Loss: 0.6694, Train Acc: 76.2925, Val Loss: 0.8880, Va
l Acc: 68.7700, Time: 66.44s, LR: 0.010000
Epoch [3/20], Train Loss: 0.6607, Train Acc: 76.6350, Val Loss: 1.0998, Va
l Acc: 63.5800, Time: 66.58s, LR: 0.010000
Epoch [4/20], Train Loss: 0.6477, Train Acc: 77.0900, Val Loss: 0.8753, Va
l Acc: 70.4300, Time: 66.54s, LR: 0.010000
Epoch [5/20], Train Loss: 0.6364, Train Acc: 77.7900, Val Loss: 0.8321, Va
l Acc: 70.9700, Time: 66.51s, LR: 0.010000
Epoch [6/20], Train Loss: 0.6281, Train Acc: 77.9850, Val Loss: 0.7908, Va
l Acc: 72.3600, Time: 66.44s, LR: 0.010000
Epoch [7/20], Train Loss: 0.6231, Train Acc: 78.0400, Val Loss: 0.8611, Va
l Acc: 70.5600, Time: 66.52s, LR: 0.010000
Epoch [8/20], Train Loss: 0.6087, Train Acc: 78.3225, Val Loss: 0.8819, Va
l Acc: 69.7600, Time: 66.51s, LR: 0.010000
Epoch [9/20], Train Loss: 0.5991, Train Acc: 78.9400, Val Loss: 0.7064, Va
l Acc: 75.8800, Time: 66.43s, LR: 0.010000
Epoch [10/20], Train Loss: 0.5998, Train Acc: 78.7600, Val Loss: 0.7942, V
al Acc: 72.8600, Time: 66.53s, LR: 0.010000
Epoch [11/20], Train Loss: 0.5891, Train Acc: 79.0875, Val Loss: 0.7422, V
al Acc: 74.6300, Time: 66.41s, LR: 0.010000
Epoch [12/20], Train Loss: 0.5878, Train Acc: 79.2850, Val Loss: 0.8717, V
al Acc: 71.2500, Time: 66.46s, LR: 0.001000
Epoch [13/20], Train Loss: 0.4447, Train Acc: 84.6900, Val Loss: 0.5604, V
al Acc: 80.5100, Time: 66.49s, LR: 0.001000
Epoch [14/20], Train Loss: 0.3991, Train Acc: 86.2050, Val Loss: 0.5580, V
al Acc: 80.7000, Time: 66.49s, LR: 0.001000
Epoch [15/20], Train Loss: 0.3797, Train Acc: 86.9750, Val Loss: 0.5600, V
al Acc: 80.7000, Time: 66.57s, LR: 0.001000
Epoch [16/20], Train Loss: 0.3640, Train Acc: 87.5475, Val Loss: 0.5598, V
al Acc: 80.8800, Time: 66.54s, LR: 0.001000
Epoch [17/20], Train Loss: 0.3482, Train Acc: 88.1400, Val Loss: 0.5609, V
al Acc: 80.8000, Time: 66.47s, LR: 0.001000
Epoch [18/20], Train Loss: 0.3377, Train Acc: 88.4775, Val Loss: 0.5730, V
al Acc: 80.4800, Time: 66.52s, LR: 0.001000
Epoch [19/20], Train Loss: 0.3266, Train Acc: 89.1375, Val Loss: 0.5737, V
al Acc: 80.6400, Time: 66.51s, LR: 0.000100
Epoch [20/20], Train Loss: 0.2948, Train Acc: 90.2675, Val Loss: 0.5613, V
al Acc: 81.0400, Time: 66.58s, LR: 0.000100
```

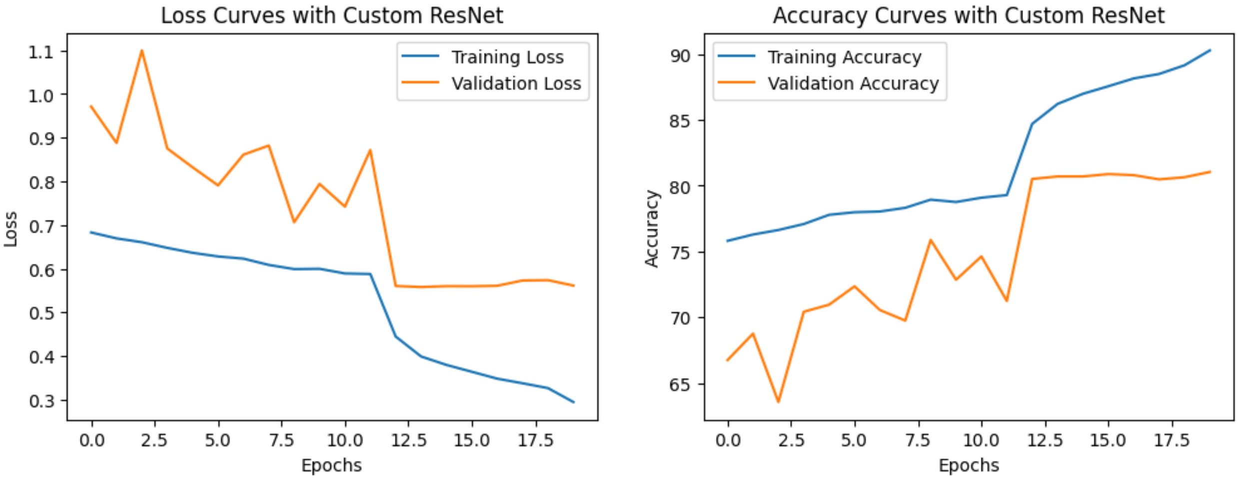
In [ ]:

```python
# Plot the learning curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(adam_train_loss_sched, label='Training Loss')
plt.plot(adam_val_loss_sched, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss Curves with Custom ResNet')

plt.subplot(1, 2, 2)
plt.plot(adam_train_acc_sched, label='Training Accuracy')
plt.plot(adam_val_acc_sched, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

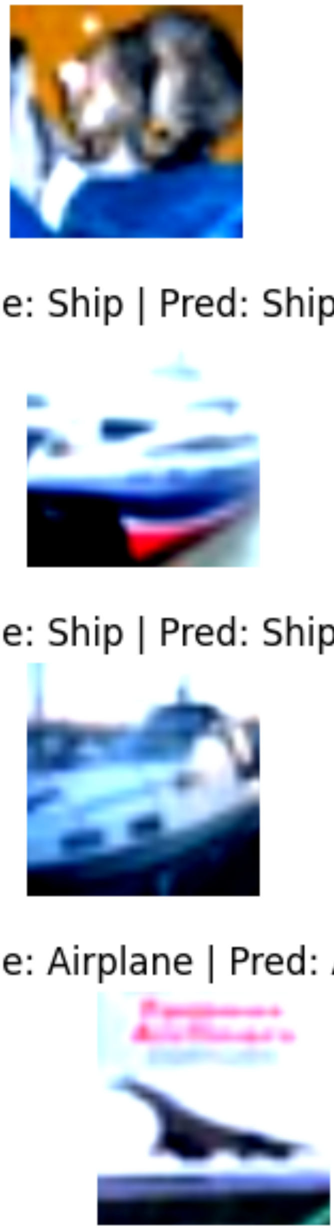
```python
plt.title('Accuracy Curves with Custom ResNet')

plt.show()
```



In [ ]:
```python
visualize_model_predictions(AdamModel, loader=tst_loader, classes=classes
```

True: Cat | Pred: Cat



True: Ship | Pred: Ship

True: Ship | Pred: Ship

True: Airplane | Pred: Airplane

True: Frog | Pred: Frog

True: Frog | Pred: Frog

Task 3: Transfer Learning with ResNet-50

I will use a pre-trained ResNet-50 model and fine-tune its last layer for CIFAR-10 classification.

In [ ]:
```
# Load pre-trained ResNet-50 model
pre_model = models.resnet50(pretrained=True)

# Freeze all the layers
for param in pre_model.parameters():
    param.requires_grad = False

# Modify and train only the last layer
num_ftrs = model.fc.in_features
pre_model.fc = nn.Sequential(
    nn.Linear(num_ftrs, 512),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(512, 10)
)

# Transfer model to GPU if available
pre_model = pre_model.to(device)

# Define optimizer and criterion
optimizer = optim.Adam(pre_model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

In [ ]:
```
# Train the model
def train_model(model, criterion, optimizer, train_loader, val_loader, nu
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0
```

```python
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

        epoch_loss = running_loss / total
        epoch_acc = 100. * correct / total

        # Validate the model
        val_loss, val_acc = evaluate_model(model, criterion, val_loader)

        print(f'Epoch [{epoch + 1}/{num_epochs}], Train Loss: {epoch_loss
              f'Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%')

def evaluate_model(model, criterion, data_loader):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for inputs, labels in data_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            running_loss += loss.item() * inputs.size(0)
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

    avg_loss = running_loss / total
    accuracy = 100. * correct / total

    return avg_loss, accuracy
```

In [ ]:

```python
# Train the model
train_model(pre_model, criterion, optimizer, trn_loader, vld_loader, num_

# Evaluate on the test set
test_loss, test_acc = evaluate_model(pre_model, criterion, tst_loader)
print(f'Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}%')
```

```
Epoch [1/10], Train Loss: 0.9070, Train Acc: 68.60%, Val Loss: 0.7140, Val
Acc: 75.84%
Epoch [2/10], Train Loss: 0.7274, Train Acc: 75.08%, Val Loss: 0.6602, Val
Acc: 76.86%
Epoch [3/10], Train Loss: 0.7043, Train Acc: 75.62%, Val Loss: 0.6498, Val
Acc: 77.56%
Epoch [4/10], Train Loss: 0.6776, Train Acc: 76.68%, Val Loss: 0.6125, Val
Acc: 78.62%
Epoch [5/10], Train Loss: 0.6575, Train Acc: 77.20%, Val Loss: 0.6226, Val
Acc: 78.30%
Epoch [6/10], Train Loss: 0.6499, Train Acc: 77.48%, Val Loss: 0.6185, Val
Acc: 78.30%
Epoch [7/10], Train Loss: 0.6330, Train Acc: 77.97%, Val Loss: 0.6055, Val
Acc: 78.57%
Epoch [8/10], Train Loss: 0.6330, Train Acc: 77.86%, Val Loss: 0.5922, Val
Acc: 79.52%
Epoch [9/10], Train Loss: 0.6153, Train Acc: 78.58%, Val Loss: 0.5921, Val
Acc: 79.34%
Epoch [10/10], Train Loss: 0.6073, Train Acc: 78.77%, Val Loss: 0.6250, Va
l Acc: 78.23%
Test Loss: 0.6247, Test Acc: 78.49%
```

Conclusion

In this homework, I implemented and trained a ResNet model with bottleneck residual blocks from scratch, experimented with different training dynamics using ADAM and learning rate scheduling, and applied transfer learning using a pre-trained ResNet-50 model. Each method was evaluated on the CIFAR-10 dataset. I aimed to achieve high test accuracy while ensuring robust training processes.

References

- Shi, Dachuan & Ye 叶运广, Yunguang & Gillwald, Marco & Hecht, Markus. (2020). Designing a lightweight 1D convolutional neural network with Bayesian optimization for wheel flat detection using carbody accelerations. 10.1080/23248378.2020.1795942.
- http://karpathy.github.io/2019/04/25/recipe/
- https://github.com/TylerYep/torchinfo
- https://colab.research.google.com/github/Mostafa-MR/Convert_ipynb_to_HTML_in_Colab/blob/main/Convert_ipynb_to_HTML_in_Colal JpA