Assignment 2 24133101g AMA564

1. (1) f_1 result: $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = 0$, like this from left to right

$$\text{result} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 2 & 2 & 1 \end{bmatrix}$$

f_2 result:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

f_3 result:

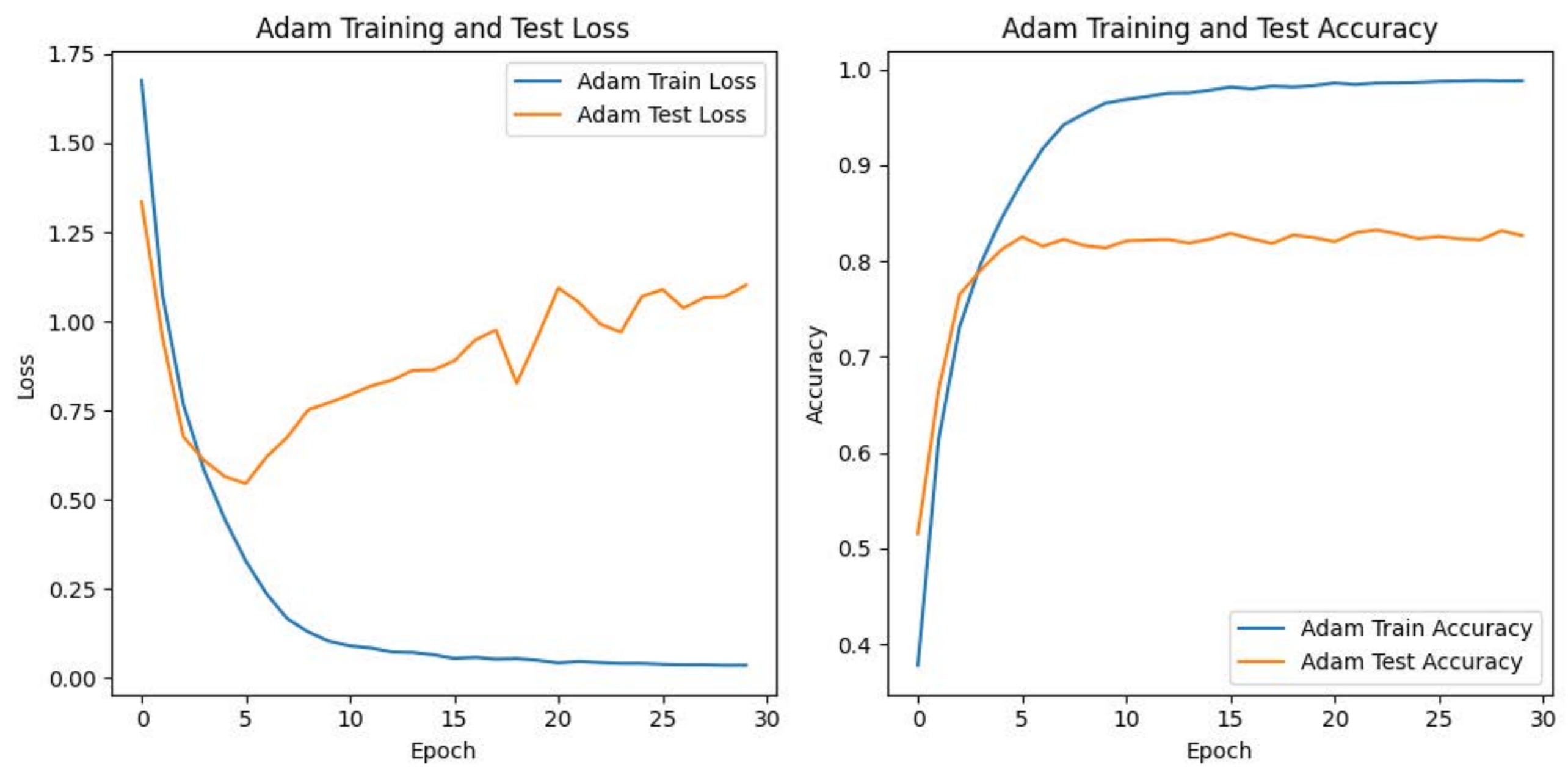
$$\begin{bmatrix} 0 & -1 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 & 0 \\ -1 & 1 & 0 & -1 & 1 \\ 0 & -1 & -1 & -1 & -1 \\ -1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(2) MaxPool result, for example, size $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ maxpool is 1.

then result is $\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

(3) By similar way, AvgPool result is $\begin{bmatrix} \frac{2}{9} & \frac{2}{9} \\ \frac{2}{9} & \frac{2}{9} \end{bmatrix}$


SGD Training and Test Loss
SGD Train Loss
SGD Test Loss
Loss
Epoch
SGD Training and Test Accuracy
SGD Train Accuracy
SGD Test Accuracy
Accuracy
Epoch


Adam Training and Test Loss
Adam Train Loss
Adam Test Loss
Loss
Epoch
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
30
Adam Training and Test Accuracy
Adam Train Accuracy
Adam Test Accuracy
Accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4

3.

(a) False

(b) True

(c) True

(d) False

(e) True

4.

(a) D (b) B (c) B (d) D (e) B

```
# Import some necessary library

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F
from torch.utils.data import DataLoader
```

```
# Build the 18-layer ResNet model
class BasicBlock(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1):
        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.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride,
bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        residual = x

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

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

        out += self.shortcut(residual)
        out = F.relu(out)

        return out


class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
```

```
        super().__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,
bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 , num_classes)

    def _make_layer(self, block, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_channels, out_channels, stride))
            self.in_channels = out_channels
        return nn.Sequential(*layers)

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

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

        out = self.avg_pool(out)
        out = torch.flatten(out, 1)
        out = self.fc(out)

        return out


def resnet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])
```

```
# Initialize two ResNet 18 models
device = "cuda" if torch.cuda.is_available() else "cpu"
model_SGD = resnet18().to(device)
model_ADAM = resnet18().to(device)
```

```
# Load CIFAR 10 Datasets

transform = transforms.Compose(
```

```
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 32

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=1)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=1)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data\cifar-10-python.tar.gz


100%|██████████| 170M/170M [03:18<00:00, 857kB/s]


Extracting ./data\cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

```
# Train function
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    train_loss = 0
    train_correct = 0
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)
        train_loss += loss.item()

        train_correct += (pred.argmax(1) == y).type(torch.float).sum().item()
        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
    train_loss /= num_batches
    train_acc = train_correct / size
    print(f"Avg Train loss: {loss:>8f} \n")
    return train_loss, train_acc
```

```
# Test function
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:
{test_loss:>8f} \n")
    return test_loss, correct
```

```
import torch.optim as optim

# Define the loss function
criterion = nn.CrossEntropyLoss()

# set the optimizer as SGD with Momentum
# Please finish this part
optimizer_SGD = optim.SGD(model_SGD.parameters(), lr=0.001, momentum=0.9)
optimizer_ADAM = optim.Adam(model_ADAM.parameters(), lr=0.01)
```

```
# Train model_SGD by SGD with Momentum optimization algorithm
# Please add code to finish this part.
num_epochs = 30

sgd_train_loss = []
sgd_test_loss = []
sgd_train_accuracy = []
sgd_test_accuracy = []

for epoch in range(num_epochs):
    train_loss, train_accuracy = train(trainloader, model_SGD, criterion,
optimizer_SGD)
    test_loss, test_accuracy = test(testloader, model_SGD, criterion)
    sgd_train_loss.append(train_loss)
```

```
    sgd_test_loss.append(test_loss)
    sgd_train_accuracy.append(train_accuracy)
    sgd_test_accuracy.append(test_accuracy)
    if epoch % 10 == 0:
        print(f"Epoch {epoch+1}: Train Loss: {train_loss:.4f}, Train Acc:
{train_accuracy:.4f}, Test Loss: {test_loss:.4f}, Test Acc: {test_accuracy:.4f}")
```

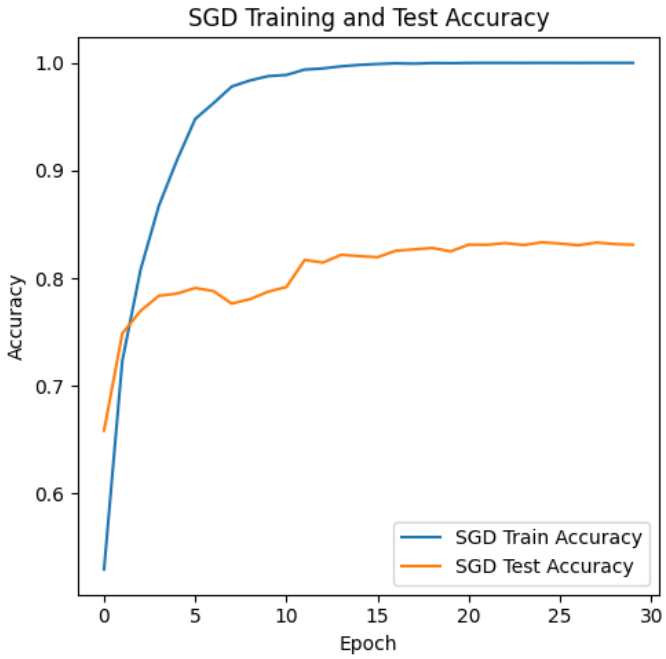
```
Avg Train loss: 0.781219

Test Error:
 Accuracy: 65.8%, Avg loss: 0.955261

Epoch 1: Train Loss: 1.2948, Train Acc: 0.5296, Test Loss: 0.9553, Test Acc:
0.6585
Avg Train loss: 0.584577

Test Error:
 Accuracy: 74.9%, Avg loss: 0.727701

Avg Train loss: 0.607731

Test Error:
 Accuracy: 77.0%, Avg loss: 0.662666

Avg Train loss: 0.596089

Test Error:
 Accuracy: 78.4%, Avg loss: 0.666445

Avg Train loss: 0.105106

Test Error:
 Accuracy: 78.6%, Avg loss: 0.684234

Avg Train loss: 0.117027

Test Error:
 Accuracy: 79.1%, Avg loss: 0.745769

Avg Train loss: 0.047488

Test Error:
 Accuracy: 78.8%, Avg loss: 0.809747

Avg Train loss: 0.287708

Test Error:
 Accuracy: 77.6%, Avg loss: 0.912338
```

```
Avg Train loss: 0.081669

Test Error:
 Accuracy: 78.0%, Avg loss: 0.949332

Avg Train loss: 0.230706

Test Error:
 Accuracy: 78.8%, Avg loss: 0.872903

Avg Train loss: 0.041926

Test Error:
 Accuracy: 79.2%, Avg loss: 0.901697

Epoch 11: Train Loss: 0.0356, Train Acc: 0.9888, Test Loss: 0.9017, Test Acc:
0.7918
Avg Train loss: 0.008251

Test Error:
 Accuracy: 81.7%, Avg loss: 0.811584

Avg Train loss: 0.041579

Test Error:
 Accuracy: 81.5%, Avg loss: 0.828728

Avg Train loss: 0.007044

Test Error:
 Accuracy: 82.2%, Avg loss: 0.793423

Avg Train loss: 0.001683

Test Error:
 Accuracy: 82.0%, Avg loss: 0.812661

Avg Train loss: 0.002475

Test Error:
 Accuracy: 82.0%, Avg loss: 0.812516

Avg Train loss: 0.001047

Test Error:
 Accuracy: 82.5%, Avg loss: 0.791074

Avg Train loss: 0.002522

Test Error:
```

```
 Accuracy: 82.7%, Avg loss: 0.813843

Avg Train loss: 0.000747

Test Error:
 Accuracy: 82.8%, Avg loss: 0.795753

Avg Train loss: 0.000205

Test Error:
 Accuracy: 82.5%, Avg loss: 0.819318

Avg Train loss: 0.010144

Test Error:
 Accuracy: 83.1%, Avg loss: 0.793875

Epoch 21: Train Loss: 0.0011, Train Acc: 0.9999, Test Loss: 0.7939, Test Acc:
0.8312
Avg Train loss: 0.000622

Test Error:
 Accuracy: 83.1%, Avg loss: 0.788106

Avg Train loss: 0.000180

Test Error:
 Accuracy: 83.3%, Avg loss: 0.789987

Avg Train loss: 0.000343

Test Error:
 Accuracy: 83.1%, Avg loss: 0.800912

Avg Train loss: 0.000270

Test Error:
 Accuracy: 83.3%, Avg loss: 0.785095

Avg Train loss: 0.001528

Test Error:
 Accuracy: 83.2%, Avg loss: 0.798864

Avg Train loss: 0.013174

Test Error:
 Accuracy: 83.1%, Avg loss: 0.800578

Avg Train loss: 0.000230
```

```
Test Error:
 Accuracy: 83.3%, Avg loss: 0.798864

Avg Train loss: 0.005760

Test Error:
 Accuracy: 83.2%, Avg loss: 0.820524

Avg Train loss: 0.000043

Test Error:
 Accuracy: 83.1%, Avg loss: 0.808757
```

```
# Visualize the results (You can refer to other tutorial notebooks)
import matplotlib.pyplot as plt

# 绘制损失曲线
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(range(num_epochs), sgd_train_loss, label='SGD Train Loss')
plt.plot(range(num_epochs), sgd_test_loss, label='SGD Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('SGD Training and Test Loss')
plt.legend()

# 绘制准确率曲线
plt.subplot(1, 2, 2)
plt.plot(range(num_epochs), sgd_train_accuracy, label='SGD Train Accuracy')
plt.plot(range(num_epochs), sgd_test_accuracy, label='SGD Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('SGD Training and Test Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
## Add codes to finish this part and save the images. Add the images to your
assignment solution.
```




```
# Repeat the training and testing procedure for model_ADAM
# Visualize the results and save the images. Add the images to your assignment
solution.
num_epochs = 30

adam_train_loss = []
adam_test_loss = []
adam_train_accuracy = []
adam_test_accuracy = []

for epoch in range(num_epochs):
    train_loss, train_accuracy = train(trainloader, model_ADAM, criterion,
optimizer_ADAM)
    test_loss, test_accuracy = test(testloader, model_ADAM, criterion)
    adam_train_loss.append(train_loss)
    adam_test_loss.append(test_loss)
    adam_train_accuracy.append(train_accuracy)
    adam_test_accuracy.append(test_accuracy)
    if epoch % 10 == 0:
        print(f"Epoch {epoch+1}: Train Loss: {train_loss:.4f}, Train Acc:
{train_accuracy:.4f}, Test Loss: {test_loss:.4f}, Test Acc: {test_accuracy:.4f}")
```

```
Avg Train loss: 1.040259

Test Error:
 Accuracy: 51.6%, Avg loss: 1.334983

Epoch 1: Train Loss: 1.6751, Train Acc: 0.3782, Test Loss: 1.3350, Test Acc:
0.5156
Avg Train loss: 0.770476
```

```
Test Error:
 Accuracy: 66.6%, Avg loss: 0.956502

Avg Train loss: 1.025675

Test Error:
 Accuracy: 76.5%, Avg loss: 0.677129

Avg Train loss: 0.361973

Test Error:
 Accuracy: 79.0%, Avg loss: 0.609984

Avg Train loss: 0.209990

Test Error:
 Accuracy: 81.2%, Avg loss: 0.564775

Avg Train loss: 0.058885

Test Error:
 Accuracy: 82.5%, Avg loss: 0.545372

Avg Train loss: 0.187804

Test Error:
 Accuracy: 81.5%, Avg loss: 0.619959

Avg Train loss: 0.100845

Test Error:
 Accuracy: 82.3%, Avg loss: 0.675700

Avg Train loss: 0.200338

Test Error:
 Accuracy: 81.6%, Avg loss: 0.752205

Avg Train loss: 0.447352

Test Error:
 Accuracy: 81.4%, Avg loss: 0.771865

Avg Train loss: 0.109476

Test Error:
 Accuracy: 82.1%, Avg loss: 0.794062

Epoch 11: Train Loss: 0.0900, Train Acc: 0.9687, Test Loss: 0.7941, Test Acc:
0.8211
```

```
Avg Train loss: 0.000206

Test Error:
 Accuracy: 82.2%, Avg loss: 0.818789

Avg Train loss: 0.083099

Test Error:
 Accuracy: 82.3%, Avg loss: 0.834852

Avg Train loss: 0.102807

Test Error:
 Accuracy: 81.9%, Avg loss: 0.862252

Avg Train loss: 0.029476

Test Error:
 Accuracy: 82.3%, Avg loss: 0.863798

Avg Train loss: 0.094390

Test Error:
 Accuracy: 82.9%, Avg loss: 0.889281

Avg Train loss: 0.246444

Test Error:
 Accuracy: 82.3%, Avg loss: 0.947478

Avg Train loss: 0.110304

Test Error:
 Accuracy: 81.8%, Avg loss: 0.975624

Avg Train loss: 0.101774

Test Error:
 Accuracy: 82.7%, Avg loss: 0.826052

Avg Train loss: 0.008079

Test Error:
 Accuracy: 82.5%, Avg loss: 0.956007

Avg Train loss: 0.001572

Test Error:
 Accuracy: 82.0%, Avg loss: 1.093773

Epoch 21: Train Loss: 0.0425, Train Acc: 0.9859, Test Loss: 1.0938, Test Acc:
```

```
0.8203
Avg Train loss: 0.043623

Test Error:
 Accuracy: 83.0%, Avg loss: 1.052545

Avg Train loss: 0.001397

Test Error:
 Accuracy: 83.3%, Avg loss: 0.992274

Avg Train loss: 0.062740

Test Error:
 Accuracy: 82.8%, Avg loss: 0.969702

Avg Train loss: 0.003820

Test Error:
 Accuracy: 82.3%, Avg loss: 1.069891

Avg Train loss: 0.048435

Test Error:
 Accuracy: 82.6%, Avg loss: 1.089307

Avg Train loss: 0.000088

Test Error:
 Accuracy: 82.3%, Avg loss: 1.037655

Avg Train loss: 0.025093

Test Error:
 Accuracy: 82.2%, Avg loss: 1.066776

Avg Train loss: 0.000743

Test Error:
 Accuracy: 83.2%, Avg loss: 1.069635

Avg Train loss: 0.380013

Test Error:
 Accuracy: 82.7%, Avg loss: 1.101858
```

```
# Visualize the results (You can refer to other tutorial notebooks)
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(range(num_epochs), adam_train_loss, label='Adam Train Loss')
plt.plot(range(num_epochs), adam_test_loss, label='Adam Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Adam Training and Test Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(num_epochs), adam_train_accuracy, label='Adam Train Accuracy')
plt.plot(range(num_epochs), adam_test_accuracy, label='Adam Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Adam Training and Test Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```