Assignment 1

Due by 11:59pm Sept. 18, 2023

Theory Questions (Question 1: 9 points, Question 2: 4 points, Question 3: 12 points)

1.Suppose `I` is a 5×5 image, `K` is a 3×3 convolving kernel. Compute the convolution of the image `I` with `K`, with the given settings.

`I`:

0	1	0	0	1
1	1	0	1	1
1	0	1	0	0
0	1	1	0	0
0	0	1	1	0

`K`:

0	1	0
1	0	1
0	1	0

a) Zero padding, stride $= 1$.

b) Zero padding, stride $= 2$.

b) No padding, stride $= 1$.

Answer: a

2.Let `I` be the input. Write the output using average pooling with 2×2 kernel, stride $= 2$.

9	2	6	3
1	5	0	8
5	4	5	2
6	3	1	7

`I`:

Answer: 4.1875

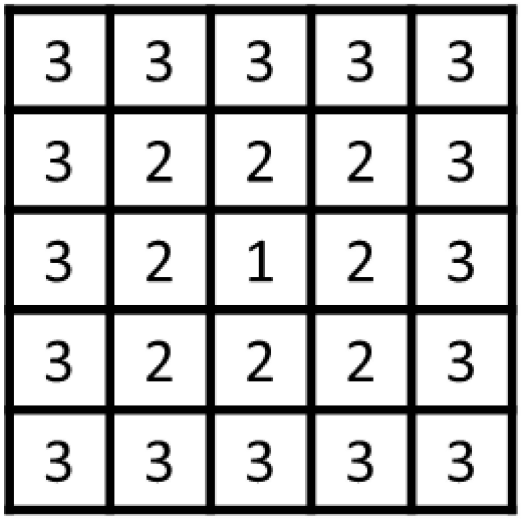
3. `I` is a 5×5 RGB image. `K` is a 3×3 convolving kernel with all its weights equal to -1. `b` is the bias equal to 1.

1	0	1	0	1
0	1	0	1	0
1	0	1	0	1
0	1	0	1	0
1	0	1	0	1

The `R` channel is given as

0	1	2	3	4
5	6	7	8	9
9	8	7	6	5
0	1	2	3	4
5	6	7	8	9

The `G` channel is given as



The `B` channel is given as

Please calculate the convolution of `I` with `K` and `b`, where stride $= 1$, and no padding is applied.

Answer: b_r = 2.0 b_g = -1.0 b_b = 0.5

Programming Questions (Question 4: 45 points, Question 5, 30 points)

4.Load mnist dataset. Normalize the data. Split the data into training, validation and testing set.

Build a CNN network with convolution layers, pooling layers to classify the number.

Plot the training loss and validation loss as a function of epochs.

Plot the both training accuracy and validation accuracy as a function of epochs.

Print the testing accuracy.

Note: Initial code has been provided to import the necessary packages and load the dataset. Now that we have introduced PyTorch programming, you should use it to solve the programming problems in this assignment.

In [1]:
```
# Import the necessary libraries for the problem
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils as utils
from torch.utils.data import DataLoader
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
```

In [2]:
```
# Loading the Mnist data and preprocessing it
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,)
```

```
train_dataset = torchvision.datasets.MNIST(root='./data', train=True, transform=transform,
test_dataset = torchvision.datasets.MNIST(root='./data', train=False, transform=transform,
```

In [3]:
```
# Splitting the data into three types ---> training, validation, and testing sets
train_size = int(0.8 * len(train_dataset))
val_size = len(train_dataset) - train_size

train_dataset, val_dataset = torch.utils.data.random_split(train_dataset, [train_size, val
```

In [4]:
```
# Loading the data
batch_size = 64
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=T
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size)
```

In [5]:
```
# Creating the CNN model by defining it
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 64 * 7 * 7)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

In [6]:
```
# Initializing the model, loss function, and the optimizer
model = CNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

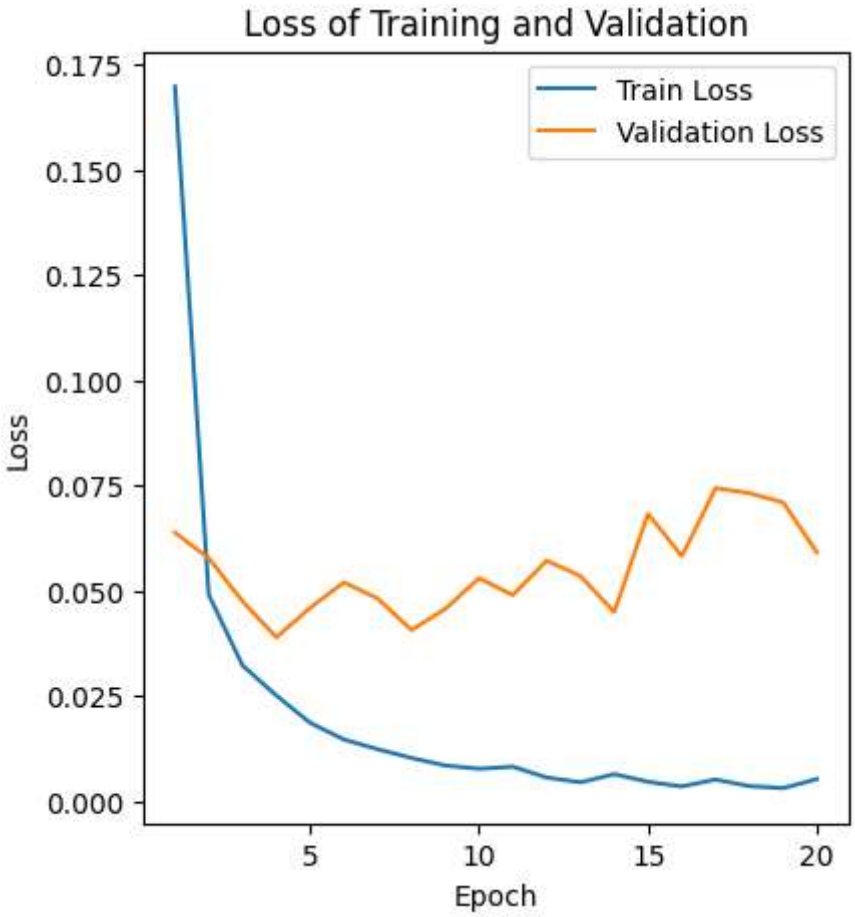
In [7]:
```
# Training model
num_epochs = 20
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0

    for images, labels in train_loader:
        optimizer.zero_grad()
```

```
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

        _, predicted = torch.max(outputs, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    train_accuracy = 100 * correct_train / total_train
    train_losses.append(running_loss / len(train_loader))
    train_accuracies.append(train_accuracy)

    # Validation Model
    model.eval()
    running_val_loss = 0.0
    correct_val = 0
    total_val = 0

    with torch.no_grad():
        for images, labels in val_loader:
            outputs = model(images)
            val_loss = criterion(outputs, labels)
            running_val_loss += val_loss.item()

            _, predicted = torch.max(outputs, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

        val_accuracy = 100 * correct_val / total_val
        val_losses.append(running_val_loss / len(val_loader))
        val_accuracies.append(val_accuracy)

    print(f'Epoch [{epoch + 1}/{num_epochs}] Train Loss: {train_losses[-1]:.4f} '
          f'Val Loss: {val_losses[-1]:.4f} Train Acc: {train_accuracy:.2f}% Val Acc: {val_
```

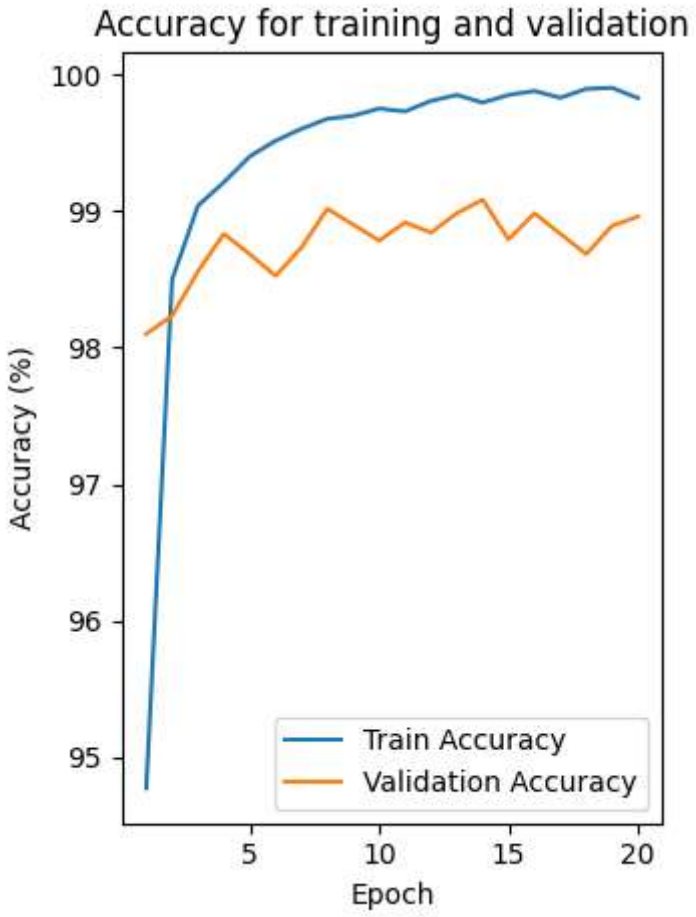
```
Epoch [1/20] Train Loss: 0.1698 Val Loss: 0.0638 Train Acc: 94.77% Val Acc: 98.10%
Epoch [2/20] Train Loss: 0.0490 Val Loss: 0.0577 Train Acc: 98.50% Val Acc: 98.23%
Epoch [3/20] Train Loss: 0.0322 Val Loss: 0.0475 Train Acc: 99.04% Val Acc: 98.56%
Epoch [4/20] Train Loss: 0.0251 Val Loss: 0.0389 Train Acc: 99.21% Val Acc: 98.83%
Epoch [5/20] Train Loss: 0.0186 Val Loss: 0.0459 Train Acc: 99.40% Val Acc: 98.68%
Epoch [6/20] Train Loss: 0.0146 Val Loss: 0.0520 Train Acc: 99.51% Val Acc: 98.53%
Epoch [7/20] Train Loss: 0.0123 Val Loss: 0.0481 Train Acc: 99.60% Val Acc: 98.73%
Epoch [8/20] Train Loss: 0.0103 Val Loss: 0.0407 Train Acc: 99.67% Val Acc: 99.02%
Epoch [9/20] Train Loss: 0.0085 Val Loss: 0.0457 Train Acc: 99.70% Val Acc: 98.90%
Epoch [10/20] Train Loss: 0.0077 Val Loss: 0.0530 Train Acc: 99.75% Val Acc: 98.78%
Epoch [11/20] Train Loss: 0.0082 Val Loss: 0.0490 Train Acc: 99.73% Val Acc: 98.92%
Epoch [12/20] Train Loss: 0.0056 Val Loss: 0.0571 Train Acc: 99.81% Val Acc: 98.84%
Epoch [13/20] Train Loss: 0.0045 Val Loss: 0.0535 Train Acc: 99.85% Val Acc: 98.98%
Epoch [14/20] Train Loss: 0.0064 Val Loss: 0.0449 Train Acc: 99.79% Val Acc: 99.08%
Epoch [15/20] Train Loss: 0.0046 Val Loss: 0.0683 Train Acc: 99.85% Val Acc: 98.79%
Epoch [16/20] Train Loss: 0.0035 Val Loss: 0.0582 Train Acc: 99.88% Val Acc: 98.98%
Epoch [17/20] Train Loss: 0.0051 Val Loss: 0.0744 Train Acc: 99.83% Val Acc: 98.83%
Epoch [18/20] Train Loss: 0.0036 Val Loss: 0.0732 Train Acc: 99.89% Val Acc: 98.68%
Epoch [19/20] Train Loss: 0.0031 Val Loss: 0.0710 Train Acc: 99.90% Val Acc: 98.89%
Epoch [20/20] Train Loss: 0.0052 Val Loss: 0.0591 Train Acc: 99.83% Val Acc: 98.96%
```

In [8]:
```
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss of Training and Validation')
```

Out[8]:
```
Text(0.5, 1.0, 'Loss of Training and Validation')
```



In [9]:
```
plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, num_epochs + 1), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.title('Accuracy for training and validation')

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



In [10]:
```
model.eval()
correct_test = 0
total_test = 0

with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total_test += labels.size(0)
        correct_test += (predicted == labels).sum().item()

test_accuracy = 100 * correct_test / total_test
print(f'Test Accuracy: {test_accuracy:.2f}%')
```

```
Test Accuracy: 99.05%
```

5.Load cifar10 dataset. Build a CNN network with convolution layers to classify the images.

Print the accuracy.

Tune the hyper parameters if needed to get a good accuracy.

In [11]:
```
# Define the dataset and data loaders
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
```

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

training_dataset = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True, transform=transform
)
test_dataset = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True, transform=transform
)

batch_size = 128

train_loader = torch.utils.data.DataLoader(
    dataset=training_dataset, batch_size=batch_size, shuffle=True
)
test_loader = torch.utils.data.DataLoader(
    dataset=test_dataset, batch_size=batch_size, shuffle=False
)
```

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

In [12]:
```
# Define the CNN architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(256 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.max_pool2d(x, 2)
        x = torch.relu(self.conv2(x))
        x = torch.max_pool2d(x, 2)
        x = torch.relu(self.conv3(x))
        x = torch.max_pool2d(x, 2)
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

In [13]:
```
# Creating an instance of network
net = Net()

# Defining the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
```

In [14]:
```
# Model Train
num_epochs = 20
```

```
for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {running_loss / len(train_loader)}")
```

```
Epoch 1, Loss: 1.9170854744094108
Epoch 2, Loss: 1.504779476338945
Epoch 3, Loss: 1.3007136578755
Epoch 4, Loss: 1.1566980427793225
Epoch 5, Loss: 1.0343817435871914
Epoch 6, Loss: 0.9384298397756904
Epoch 7, Loss: 0.858494293506798
Epoch 8, Loss: 0.7984940441673064
Epoch 9, Loss: 0.742782989121459
Epoch 10, Loss: 0.70002162708041
Epoch 11, Loss: 0.6573862092726676
Epoch 12, Loss: 0.6344001712396626
Epoch 13, Loss: 0.5937278773016332
Epoch 14, Loss: 0.5725595268904401
Epoch 15, Loss: 0.5431025224878355
Epoch 16, Loss: 0.5225233110168096
Epoch 17, Loss: 0.4975406737888561
Epoch 18, Loss: 0.4794955088964204
Epoch 19, Loss: 0.4615834606882861
Epoch 20, Loss: 0.4391456308877072
```

In [15]:
```
# By using the test dataset we have to evaluate the model  and print the accuracy scores a
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        inputs, labels = data
        outputs = net(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"Accuracy on the test dataset: {accuracy:.2f}%")
```

```
Accuracy on the test dataset: 81.80%
```

In [ ]:
```

```