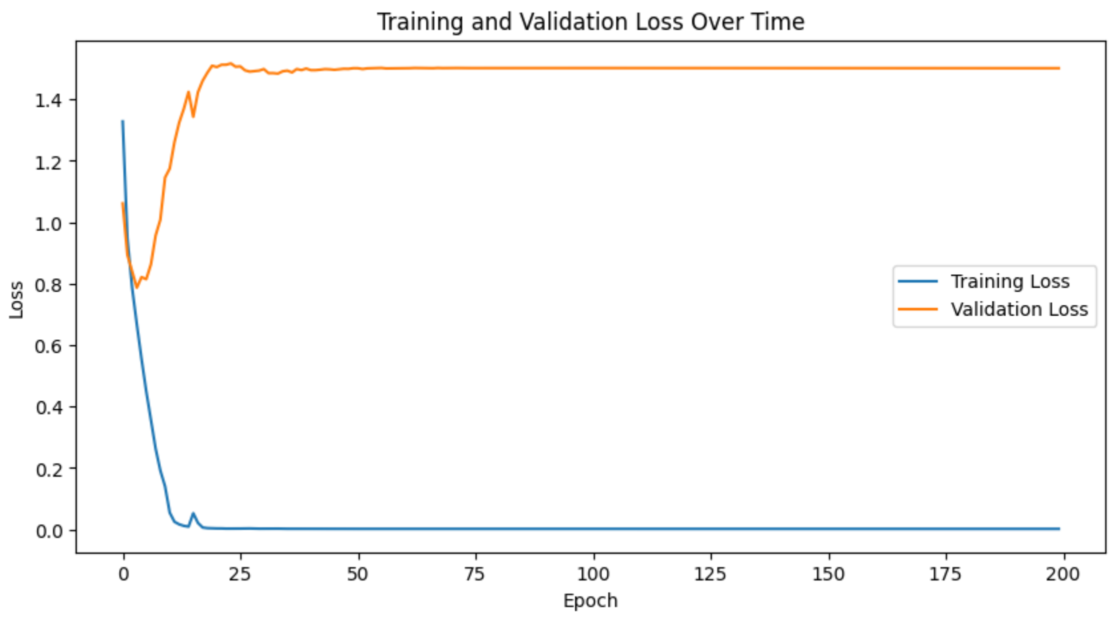
homework7

December 8, 2024

[13]:
```
##Homework 7
##1a

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import time
import matplotlib.pyplot as plt

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Data preprocessing
transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
        ])

# Load CIFAR-10 dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                      download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,
                                        shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                     download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=128,
                                       shuffle=False, num_workers=2)

# Define the CNN architecture
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # Convolutional layers
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        # Fully connected layers
        self.fc_layers = nn.Sequential(
            nn.Linear(64 * 8 * 8, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(-1, 64 * 8 * 8)  # Flatten the output
        x = self.fc_layers(x)
        return x

# Initialize model, loss function, and optimizer
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4)

# Add learning rate scheduling
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=5,␣
 ↪factor=0.5)

# Training loop
train_losses = []
val_losses = []  # New list for validation losses
start_time = time.time()

for epoch in range(200):
    # Training phase
    running_loss = 0.0
    model.train()

    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    epoch_loss = running_loss / len(trainloader)
    train_losses.append(epoch_loss)

    # Validation phase
    model.eval()
    val_running_loss = 0.0
    with torch.no_grad():
        for data in testloader:
            images, labels = data[0].to(device), data[1].to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_running_loss += loss.item()

    val_epoch_loss = val_running_loss / len(testloader)
    val_losses.append(val_epoch_loss)

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/200], Train Loss: {epoch_loss:.4f}, Val Loss:␣
 ↪{val_epoch_loss:.4f}')

    # Update learning rate
    scheduler.step(val_epoch_loss)

training_time = time.time() - start_time
print(f'Training finished in {training_time/60:.2f} minutes')

# Evaluation
model.eval()
correct = 0
total = 0

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

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

```python
# Plot training loss
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title('Training and Validation Loss Over Time')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch [10/200], Train Loss: 0.1402, Val Loss: 1.1454
Epoch [20/200], Train Loss: 0.0036, Val Loss: 1.5094
Epoch [30/200], Train Loss: 0.0023, Val Loss: 1.4935
Epoch [40/200], Train Loss: 0.0021, Val Loss: 1.5003
Epoch [50/200], Train Loss: 0.0018, Val Loss: 1.5013
Epoch [60/200], Train Loss: 0.0017, Val Loss: 1.5013
Epoch [70/200], Train Loss: 0.0017, Val Loss: 1.5020
Epoch [80/200], Train Loss: 0.0017, Val Loss: 1.5020
Epoch [90/200], Train Loss: 0.0017, Val Loss: 1.5019
Epoch [100/200], Train Loss: 0.0017, Val Loss: 1.5019
Epoch [110/200], Train Loss: 0.0017, Val Loss: 1.5019
Epoch [120/200], Train Loss: 0.0017, Val Loss: 1.5018
Epoch [130/200], Train Loss: 0.0017, Val Loss: 1.5018
Epoch [140/200], Train Loss: 0.0017, Val Loss: 1.5017
Epoch [150/200], Train Loss: 0.0017, Val Loss: 1.5016
Epoch [160/200], Train Loss: 0.0017, Val Loss: 1.5016
Epoch [170/200], Train Loss: 0.0017, Val Loss: 1.5015
Epoch [180/200], Train Loss: 0.0017, Val Loss: 1.5015
Epoch [190/200], Train Loss: 0.0017, Val Loss: 1.5014
Epoch [200/200], Train Loss: 0.0017, Val Loss: 1.5014
Training finished in 14.83 minutes
Accuracy on test set: 74.27%
```



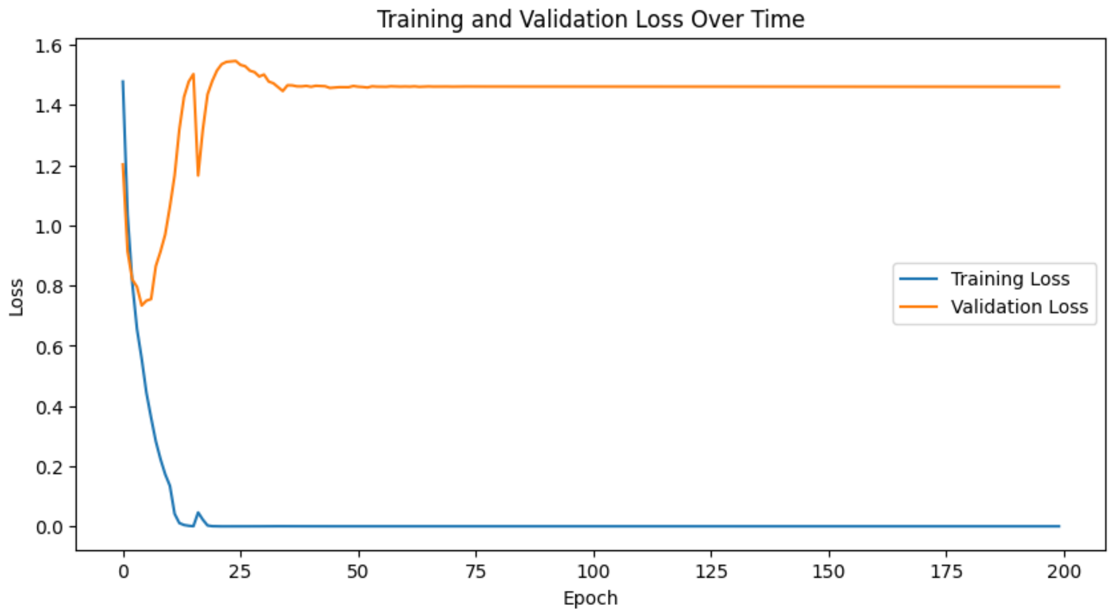
[16]:
```
##1b
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import time
import matplotlib.pyplot as plt

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Data preprocessing
transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
        ])

# Load CIFAR-10 dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                      download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,
                                        shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                     download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=128,
                                       shuffle=False, num_workers=2)

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # Convolutional layers remain the same
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        # Update the input size of the first fully connected layer
        self.fc_layers = nn.Sequential(
            nn.Linear(256 * 2 * 2, 256),  # This is correct now
            nn.ReLU(),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(-1, 256 * 2 * 2)  # Update this line to match the new␣
↪dimensions
        x = self.fc_layers(x)
        return x

# Initialize model, loss function, and optimizer
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4)

# Add learning rate scheduling
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=5,␣
↪factor=0.5)

# Training loop
```

```
train_losses = []
val_losses = []  # New list for validation losses
start_time = time.time()

for epoch in range(200):
    # Training phase
    running_loss = 0.0
    model.train()

    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    epoch_loss = running_loss / len(trainloader)
    train_losses.append(epoch_loss)

    # Validation phase
    model.eval()
    val_running_loss = 0.0
    with torch.no_grad():
        for data in testloader:
            images, labels = data[0].to(device), data[1].to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_running_loss += loss.item()

    val_epoch_loss = val_running_loss / len(testloader)
    val_losses.append(val_epoch_loss)

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/200], Train Loss: {epoch_loss:.4f}, Val Loss:␣
 ↪{val_epoch_loss:.4f}')

    # Update learning rate
    scheduler.step(val_epoch_loss)

training_time = time.time() - start_time
print(f'Training finished in {training_time/60:.2f} minutes')

# Evaluation
```

```python
model.eval()
correct = 0
total = 0

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

accuracy = 100 * correct / total
print(f'Accuracy on test set: {accuracy:.2f}%')

# Plot training loss
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title('Training and Validation Loss Over Time')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch [10/200], Train Loss: 0.1736, Val Loss: 0.9701
Epoch [20/200], Train Loss: 0.0014, Val Loss: 1.4795
Epoch [30/200], Train Loss: 0.0009, Val Loss: 1.4946
Epoch [40/200], Train Loss: 0.0010, Val Loss: 1.4634
Epoch [50/200], Train Loss: 0.0009, Val Loss: 1.4631
Epoch [60/200], Train Loss: 0.0009, Val Loss: 1.4610
Epoch [70/200], Train Loss: 0.0009, Val Loss: 1.4612
Epoch [80/200], Train Loss: 0.0009, Val Loss: 1.4612
Epoch [90/200], Train Loss: 0.0009, Val Loss: 1.4612
Epoch [100/200], Train Loss: 0.0009, Val Loss: 1.4612
Epoch [110/200], Train Loss: 0.0009, Val Loss: 1.4611
Epoch [120/200], Train Loss: 0.0009, Val Loss: 1.4610
Epoch [130/200], Train Loss: 0.0009, Val Loss: 1.4610
Epoch [140/200], Train Loss: 0.0009, Val Loss: 1.4609
Epoch [150/200], Train Loss: 0.0009, Val Loss: 1.4609
Epoch [160/200], Train Loss: 0.0009, Val Loss: 1.4608
Epoch [170/200], Train Loss: 0.0009, Val Loss: 1.4607
Epoch [180/200], Train Loss: 0.0009, Val Loss: 1.4607
Epoch [190/200], Train Loss: 0.0009, Val Loss: 1.4606
Epoch [200/200], Train Loss: 0.0009, Val Loss: 1.4606
Training finished in 14.52 minutes
```

```
Accuracy on test set: 77.81%
```

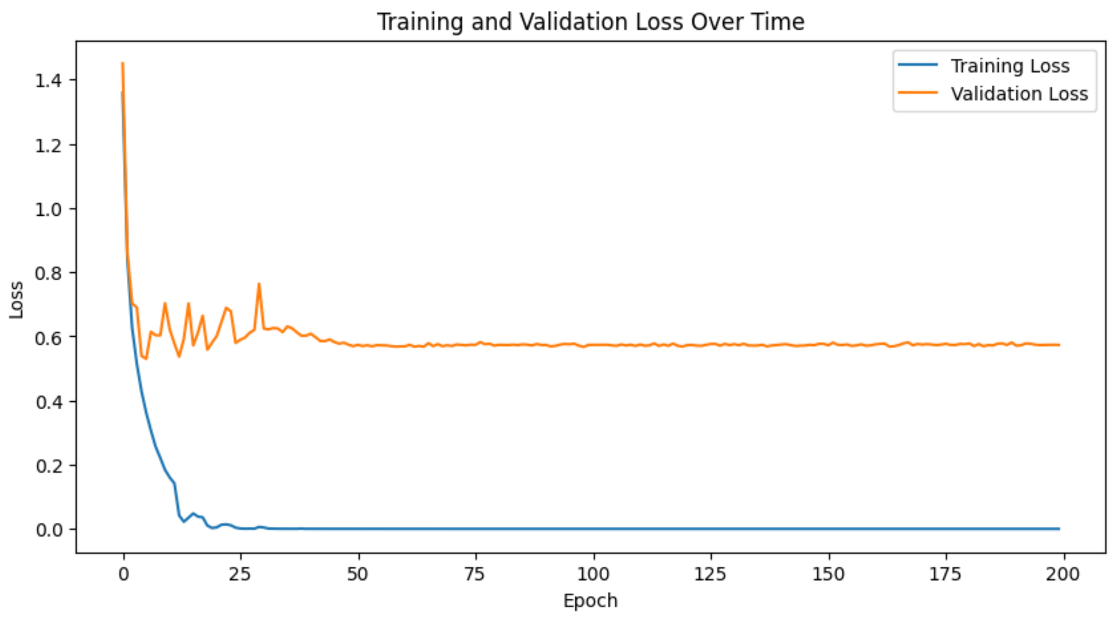


[18]:
```
##2
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import time
import matplotlib.pyplot as plt

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Data preprocessing
transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
        ])

# Load CIFAR-10 dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                      download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,
                                        shuffle=True, num_workers=2)
```

```
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=128,
                                         shuffle=False, num_workers=2)

# Define the ResNet block
class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResBlock, self).__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):
        out = torch.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = torch.relu(out)
        return out

# Replace CNN with ResNet10
class ResNet10(nn.Module):
    def __init__(self):
        super(ResNet10, self).__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)

        # Create 10 ResNet blocks across different scales
        self.layer1 = self.make_layer(64, 2, stride=1)    # 2 blocks
        self.layer2 = self.make_layer(128, 2, stride=2)   # 2 blocks
        self.layer3 = self.make_layer(256, 3, stride=2)   # 3 blocks
        self.layer4 = self.make_layer(512, 3, stride=2)   # 3 blocks
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, 10)

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

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

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

        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out

# Initialize model, loss function, and optimizer
model = ResNet10().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4)

# Add learning rate scheduling
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=5,␣
 ↪factor=0.5)

# Training loop
train_losses = []
val_losses = []  # New list for validation losses
start_time = time.time()

for epoch in range(200):
    # Training phase
    running_loss = 0.0
    model.train()

    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    epoch_loss = running_loss / len(trainloader)
    train_losses.append(epoch_loss)

    # Validation phase
    model.eval()
    val_running_loss = 0.0
    with torch.no_grad():
        for data in testloader:
            images, labels = data[0].to(device), data[1].to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_running_loss += loss.item()

    val_epoch_loss = val_running_loss / len(testloader)
    val_losses.append(val_epoch_loss)

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/200], Train Loss: {epoch_loss:.4f}, Val Loss:␣
 ↪{val_epoch_loss:.4f}')

    # Update learning rate
    scheduler.step(val_epoch_loss)

training_time = time.time() - start_time
print(f'Training finished in {training_time/60:.2f} minutes')

# Evaluation
model.eval()
correct = 0
total = 0

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

accuracy = 100 * correct / total
```

```python
print(f'Accuracy on test set: {accuracy:.2f}%')

# Plot training loss
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title('Training and Validation Loss Over Time')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch [10/200], Train Loss: 0.1841, Val Loss: 0.7032
Epoch [20/200], Train Loss: 0.0028, Val Loss: 0.5806
Epoch [30/200], Train Loss: 0.0062, Val Loss: 0.7637
Epoch [40/200], Train Loss: 0.0004, Val Loss: 0.6024
Epoch [50/200], Train Loss: 0.0003, Val Loss: 0.5695
Epoch [60/200], Train Loss: 0.0002, Val Loss: 0.5687
Epoch [70/200], Train Loss: 0.0002, Val Loss: 0.5723
Epoch [80/200], Train Loss: 0.0002, Val Loss: 0.5712
Epoch [90/200], Train Loss: 0.0002, Val Loss: 0.5731
Epoch [100/200], Train Loss: 0.0002, Val Loss: 0.5734
Epoch [110/200], Train Loss: 0.0002, Val Loss: 0.5701
Epoch [120/200], Train Loss: 0.0002, Val Loss: 0.5683
Epoch [130/200], Train Loss: 0.0002, Val Loss: 0.5726
Epoch [140/200], Train Loss: 0.0002, Val Loss: 0.5729
Epoch [150/200], Train Loss: 0.0002, Val Loss: 0.5768
Epoch [160/200], Train Loss: 0.0002, Val Loss: 0.5720
Epoch [170/200], Train Loss: 0.0002, Val Loss: 0.5764
Epoch [180/200], Train Loss: 0.0002, Val Loss: 0.5757
Epoch [190/200], Train Loss: 0.0002, Val Loss: 0.5810
Epoch [200/200], Train Loss: 0.0002, Val Loss: 0.5732
Training finished in 57.86 minutes
Accuracy on test set: 87.62%
```


Training and Validation Loss Over Time
Training Loss
Validation Loss
Loss
Epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200

```
[ ]:
```