CNN Architecture

Calicia Perea

Hw 8

- First Convolutional Layer: kernel size 3 × 3, strides 1 × 1, valid padding mode, output channel size 4.
- First Pooling Layer: max pooling with pooling size 2 × 2 and strides 2 × 2.
- Second Convolutional Layer: kernel size 3 × 3, strides 3 × 3, valid padding mode, output channel size 2.
- Second Pooling Layer: max pooling with pooling size 4 × 4, and strides 4 × 4.
- Fully Connected Layer: output channel size 10

Here is a diagram of the CNN architecture:

```
Input
    |
Convolutional Layer (4 channels, kernel size 3x3)
    |
Max Pooling Layer (pooling size 2x2)
    |
Convolutional Layer (2 channels, kernel size 3x3)
    |
Max Pooling Layer (pooling size 4x4)
    |
Fully Connected Layer (output size 10)
```

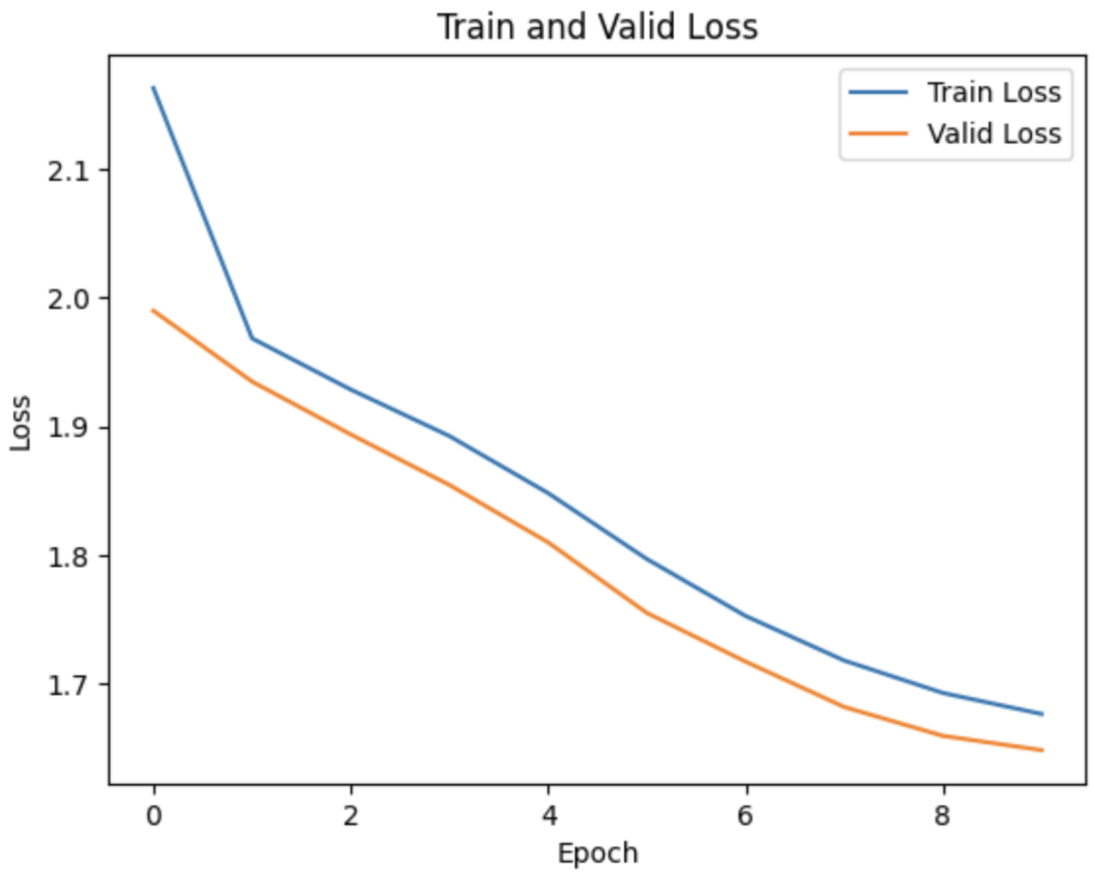
```
Input
    |
+---------------------+
| Convolutional Layer  |
| (4 channels,        |
|  kernel size 3x3)   |
+---------------------+
    |
+---------------------+
| Max Pooling Layer    |
| (pooling size 2x2)   |
+---------------------+
    |
+---------------------+
| Convolutional Layer  |
| (2 channels,        |
|  kernel size 3x3)   |
+---------------------+
    |
+---------------------+
| Max Pooling Layer    |
| (pooling size 4x4)   |
+---------------------+
    |
+---------------------+
| Fully Connected Layer|
| (output size 10)     |
+---------------------+
```

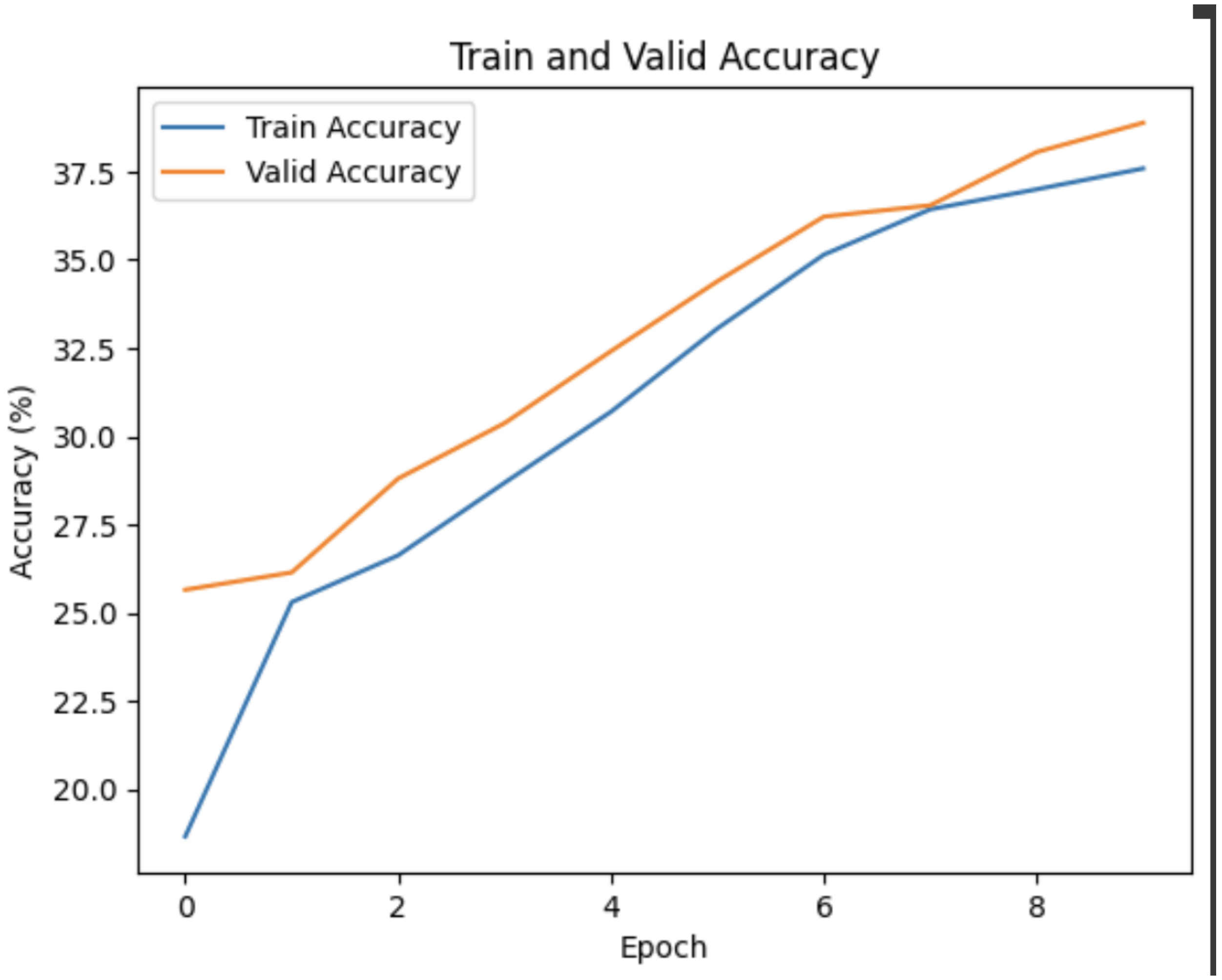
```
import torch
import torch.nn as nn
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import matplotlib.pyplot as plt


train_loss_list = []
valid_loss_list = []
train_accuracy_list = []
valid_accuracy_list = []

class MyCNN(nn.Module):
    def __init__(self):
        super(MyCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, stride=1, padding=0)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=4, out_channels=2, kernel_size=3, stride=3, padding=0)
        self.pool2 = nn.MaxPool2d(kernel_size=4, stride=4)
        self.fc = nn.Linear(in_features=2, out_features=10)
        self.relu = nn.ReLU()

   def forward(self, x):
        x = self.conv1(x)
        output_size = (x.size()[2] - 3 + 1) // 1
        output_dimension = output_size * 4
        x = self.relu(x)
        x = self.pool1(x)
        output_size = (x.size()[2] - 2 + 1) // 2
        output_dimension = output_size * 4
        x = self.conv2(x)
        output_size = (x.size()[2] - 3 + 1) // 3
        output_dimension = output_size * 2 *2
        x = self.relu(x)
        x = self.pool2(x)
        output_size = (x.size()[2] - 4 + 1) // 4
        output_dimension = output_size * 2
        x = x.view(-1, 2)
        x = self.fc(x)
        return x

# Load the MNIST dataset
train_dataset = datasets.MNIST(root='./data', train=True, transform=transforms.ToTensor(), download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transforms.ToTensor())

# Create data loaders
```

```
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=64, shuffle=False)

# Create the model
model = MyCNN()

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Train the model
num_epochs = 10
for epoch in range(num_epochs):
    train_loss = 0.0
    valid_loss = 0.0
    train_total = 0
    valid_total = 0
    train_correct = 0
    valid_correct = 0

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

        # Accumulate the loss and accuracy for this batch
        train_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()

    # Compute the epoch loss and accuracy
    train_loss /= len(train_loader.dataset)
    train_accuracy = 100.0 * train_correct / train_total

    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Accumulate the loss and accuracy for this batch
            valid_loss += loss.item() * images.size(0)
            _, predicted = torch.max(outputs.data, 1)
            valid_total += labels.size(0)
            valid_correct += (predicted == labels).sum().item()


    # Compute the epoch loss and accuracy
    valid_loss /= len(test_loader.dataset)
    valid_accuracy = 100.0 * valid_correct / valid_total

    # Print the accuracy on the test set
    print('Accuracy on the test set: {:.2f}%'.format(valid_accuracy))

    # Update the lists
    train_loss_list.append(train_loss)
    valid_loss_list.append(valid_loss)
    train_accuracy_list.append(train_accuracy)
    valid_accuracy_list.append(valid_accuracy)
    # Print the loss and accuracy for this epoch
    print('Epoch [{}/{}], Train Loss: {:.4f}, Valid Loss: {:.4f}, Train Acc: {:.2f}%, Valid Acc: {:.2f}%'
          .format(epoch+1, num_epochs, train_loss, valid_loss, train_accuracy, valid_accuracy))
# Plot train and validation loss
plt.plot(train_loss_list, label='Train Loss')
plt.plot(valid_loss_list, label='Valid Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Train and Valid Loss')
plt.show()

# Plot train and validation accuracy
plt.plot(train_accuracy_list, label='Train Accuracy')
plt.plot(valid_accuracy_list, label='Valid Accuracy')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Train and Valid Accuracy')
plt.show()
```

Output:

```
Accuracy on the test set: 25.65%
Epoch [1/10], Train Loss: 2.1632, Valid Loss: 1.9899, Train Acc: 18.66%, Valid Acc: 25.65%
Accuracy on the test set: 26.14%
Epoch [2/10], Train Loss: 1.9685, Valid Loss: 1.9349, Train Acc: 25.30%, Valid Acc: 26.14%
Accuracy on the test set: 28.80%
Epoch [3/10], Train Loss: 1.9286, Valid Loss: 1.8937, Train Acc: 26.62%, Valid Acc: 28.80%
Accuracy on the test set: 30.37%
Epoch [4/10], Train Loss: 1.8925, Valid Loss: 1.8545, Train Acc: 28.68%, Valid Acc: 30.37%
Accuracy on the test set: 32.41%
Epoch [5/10], Train Loss: 1.8481, Valid Loss: 1.8098, Train Acc: 30.69%, Valid Acc: 32.41%
Accuracy on the test set: 34.39%
Epoch [6/10], Train Loss: 1.7968, Valid Loss: 1.7549, Train Acc: 33.05%, Valid Acc: 34.39%
Accuracy on the test set: 36.22%
Epoch [7/10], Train Loss: 1.7526, Valid Loss: 1.7171, Train Acc: 35.14%, Valid Acc: 36.22%
Accuracy on the test set: 36.54%
Epoch [8/10], Train Loss: 1.7179, Valid Loss: 1.6818, Train Acc: 36.43%, Valid Acc: 36.54%
Accuracy on the test set: 38.05%
Epoch [9/10], Train Loss: 1.6927, Valid Loss: 1.6593, Train Acc: 36.99%, Valid Acc: 38.05%
Accuracy on the test set: 38.88%
Epoch [10/10], Train Loss: 1.6763, Valid Loss: 1.6482, Train Acc: 37.59%, Valid Acc: 38.88%
```


Train and Valid Loss
Train Loss
Valid Loss
Loss
Epoch
2.1
2.0
1.9
1.8
1.7
0
2
4
6
8



This code implements a Convolutional Neural Network (CNN) model named MyCNN, which is trained on the MNIST dataset. The architecture of the CNN contains two convolutional layers, two max pooling layers, and a fully connected layer. The first convolutional layer has a kernel size of 3x3, a stride of 1x1, and a valid padding mode. It outputs 4 channels. The first pooling layer performs max pooling with a pooling size of 2x2 and strides of 2x2. The second convolutional layer has a kernel size of 3x3, a stride of 3x3, and a valid padding mode. It outputs 2 channels. The second pooling layer performs max pooling with a pooling size of 4x4 and strides of 4x4. The fully connected layer has an output size of 10.

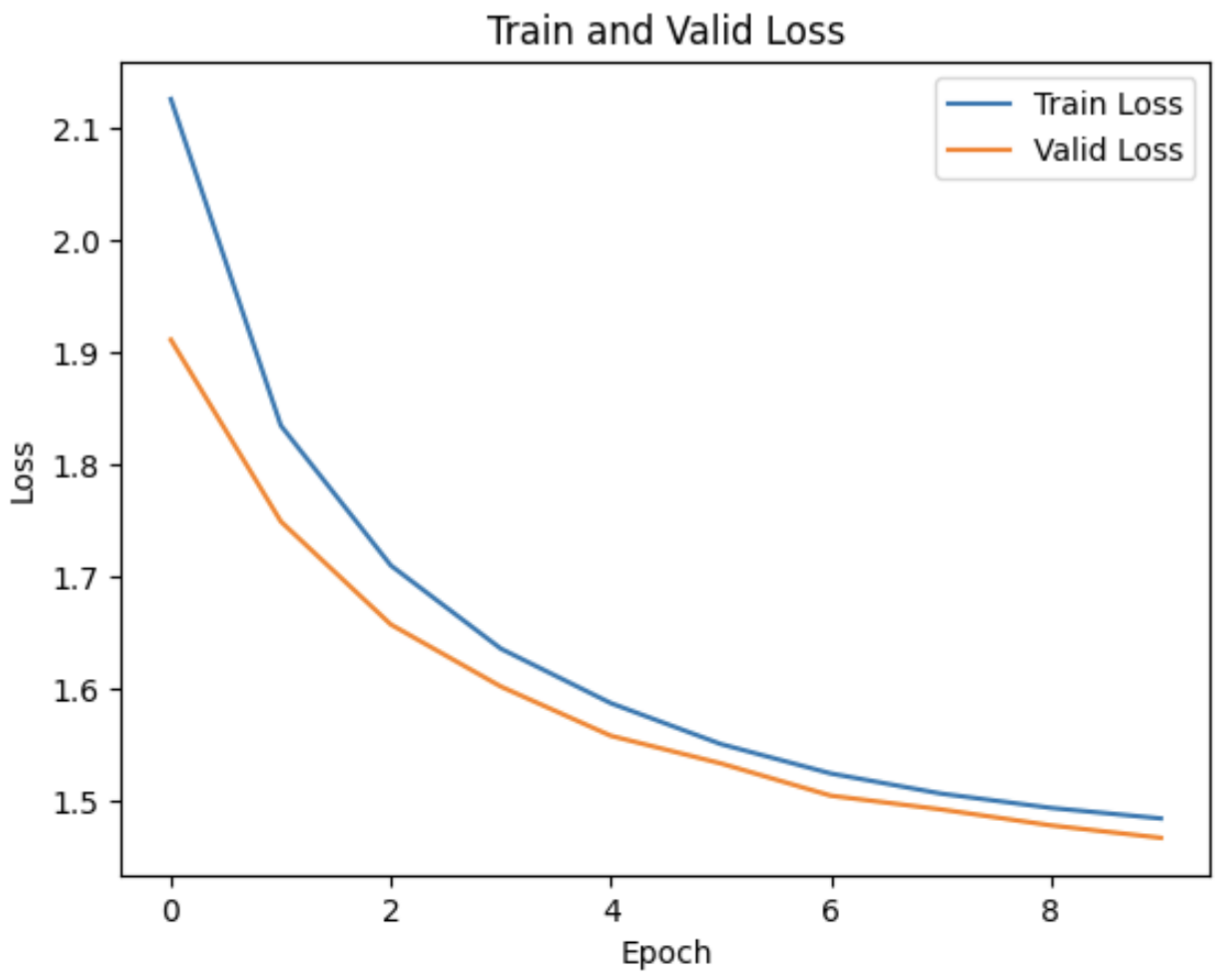
The model is trained for 10 epochs using the Adam optimizer and cross-entropy loss function. During training, the accuracy on the test set gradually increases from around 25% to nearly 39%. The accuracy on the test set is computed after each epoch. The loss and accuracy for each epoch are printed, including the train loss, validation loss, train accuracy, and validation accuracy.The code also includes two plots that show the train and validation loss and accuracy over the 10 epochs. The first plot shows the train and validation loss, while the second plot shows the train and validation accuracy. Both plots demonstrate that the model's performance improves with more epochs of training.

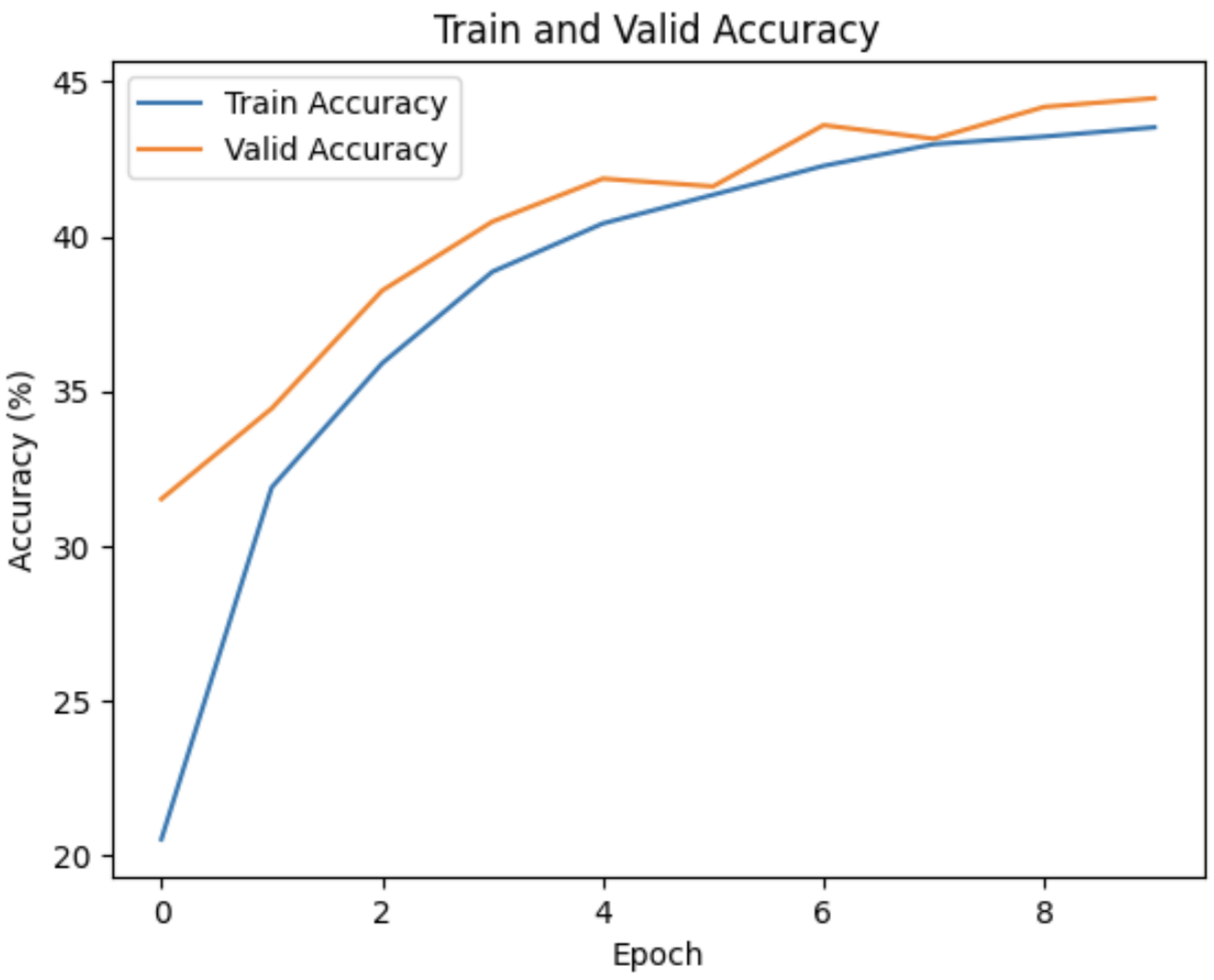
To test the code to change the kernel size of the first convolutional layer to 5x5 I modified this part of the code :

```
class MyCNN(nn.Module):
    def __init__(self):
        super(MyCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=4, kernel_size=5, stride=1, padding=0)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=4, out_channels=2, kernel_size=3, stride=3, padding=0)
        self.pool2 = nn.MaxPool2d(kernel_size=4, stride=4)
        self.fc = nn.Linear(in_features=2, out_features=10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv1(x)
        output_size = (x.size()[2] - 5 + 1) // 1
        output_dimension = output_size * 4
        x = self.relu(x)
        x = self.pool1(x)
        output_size = (x.size()[2] - 2 + 1) // 2
        output_dimension = output_size * 4
        x = self.conv2(x)
        output_size = (x.size()[2] - 3 + 1) // 3
        output_dimension = output_size * 2 *2
        x = self.relu(x)
        x = self.pool2(x)
        output_size = (x.size()[2] - 4 + 1) // 4
        output_dimension = output_size * 2
        x = x.view(-1, 2)
        x = self.fc(x)
        return x
```

Output:

```
Accuracy on the test set: 31.51%
Epoch [1/10], Train Loss: 2.1266, Valid Loss: 1.9120, Train Acc: 20.51%, Valid Acc: 31.51%
Accuracy on the test set: 34.44%
Epoch [2/10], Train Loss: 1.8354, Valid Loss: 1.7498, Train Acc: 31.89%, Valid Acc: 34.44%
Accuracy on the test set: 38.25%
Epoch [3/10], Train Loss: 1.7106, Valid Loss: 1.6580, Train Acc: 35.90%, Valid Acc: 38.25%
Accuracy on the test set: 40.48%
Epoch [4/10], Train Loss: 1.6364, Valid Loss: 1.6026, Train Acc: 38.86%, Valid Acc: 40.48%
Accuracy on the test set: 41.87%
Epoch [5/10], Train Loss: 1.5878, Valid Loss: 1.5587, Train Acc: 40.42%, Valid Acc: 41.87%
Accuracy on the test set: 41.62%
Epoch [6/10], Train Loss: 1.5513, Valid Loss: 1.5340, Train Acc: 41.35%, Valid Acc: 41.62%
Accuracy on the test set: 43.60%
Epoch [7/10], Train Loss: 1.5251, Valid Loss: 1.5053, Train Acc: 42.28%, Valid Acc: 43.60%
Accuracy on the test set: 43.16%
Epoch [8/10], Train Loss: 1.5071, Valid Loss: 1.4930, Train Acc: 42.98%, Valid Acc: 43.16%
Accuracy on the test set: 44.19%
Epoch [9/10], Train Loss: 1.4944, Valid Loss: 1.4788, Train Acc: 43.23%, Valid Acc: 44.19%
Accuracy on the test set: 44.47%
Epoch [10/10], Train Loss: 1.4849, Valid Loss: 1.4677, Train Acc: 43.53%, Valid Acc: 44.47%
```


Train and Valid Loss
Train Loss
Valid Loss
Loss
Epoch
2.1
2.0
1.9
1.8
1.7
1.6
1.5
0
2
4
6
8



The CNN model with a kernel size of 3x3 achieves a final accuracy of around 39% on the MNIST dataset after 10 epochs of training. The first convolutional layer has a kernel size of 3x3, a stride of 1x1, and a valid padding mode. It outputs 4 channels. The first pooling layer performs max pooling with a pooling size of 2x2 and strides of 2x2. The second convolutional layer has a kernel size of 3x3, a stride of 3x3, and a valid padding mode. It outputs 2 channels. The second pooling layer performs max pooling with a pooling size of 4x4 and strides of 4x4.

The CNN model with a kernel size of 5x5 achieves a final accuracy of around 44% on the MNIST dataset after 10 epochs of training. The first convolutional layer has a kernel size of 5x5, a stride of 1x1, and a valid padding mode. It outputs 4 channels. The first pooling layer performs max pooling with a pooling size of 2x2 and strides of 2x2. The second convolutional layer has a kernel size of 3x3, a stride of 3x3, and a valid padding mode. It outputs 2 channels. The second pooling layer performs max pooling with a pooling size of 4x4 and strides of 4x4.

As we can see, the CNN model with a kernel size of 5x5 achieves a slightly higher accuracy on the MNIST dataset than the CNN model with a kernel size of 3x3. However, the difference in accuracy is relatively small.