digit-recognizer-1

November 18, 2023

[ ]:
```python
import tensorflow as tf
from tensorflow.keras import datasets
import matplotlib.pyplot as plt
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader,TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import torch.optim as optim
import torch.nn as nn
from torch.optim import SGD
from sklearn.metrics import accuracy_score
```

This is the function that will enable us to print the image and its label of index(index)

[ ]:
```python
def show_image(images, labels, index):
    image = images[index].view(28, 28)  # Reshape the flattened image
    label = labels[index].item()
    figsize=(2, 2)
    plt.figure(figsize=figsize)
    plt.imshow(image, cmap='gray')
    plt.title(f"Label: {label}")
    plt.axis('off')
    plt.show()
```

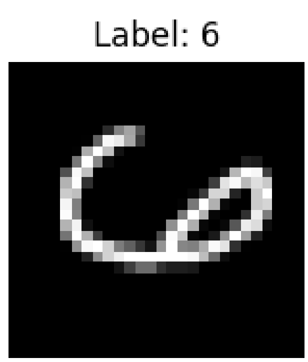
Here we will define the transform that will convert images to .tensors instead of ndarray,also it normalizes the values to [-1,1]

[ ]:
```python
# Define a transform to normalize the data
transform = transforms.Compose([
    transforms.ToTensor(),  # Converts PIL Image or numpy.ndarray to torch.
 ↪Tensor
    transforms.Normalize((0.5,), (0.5,))  # Normalize to range [-1, 1]
])
```

In this cell we will download the data and use the transform we init in the last cell also we will split the images and the labels then we will split the training data into validation (0.2) and training (0.8)

[ ]:
```
# Download and load the training data
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform,␣
↪download=True)


# Download and load the test data
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform,␣
↪download=True)


images, labels = train_dataset.data.float() , train_dataset.targets.long()

# images, labels = train_dataset.data, train_dataset.targets
test_images, test_labels = test_dataset.data.float(), test_dataset.targets.
↪long()



train_images, val_images, train_labels, val_labels = train_test_split(
    images, labels, test_size=0.2, random_state=42, stratify=labels)

train_images, train_labels = shuffle(train_images, train_labels,␣
↪random_state=42)
val_images, val_labels = shuffle(val_images, val_labels, random_state=42)
```

[ ]:
```
show_image(val_images,val_labels,10)
```



Here is the main class that does all the work:input_size, output_size, hidden_layers, activation function dropout_prob

[ ]:
```
class FlexibleNN(nn.Module):
```

```
    def __init__(self, input_size, output_size, hidden_layers=[128, 64],␣
↪activation=nn.ReLU(), dropout_prob=0.5):
        super(FlexibleNN, self).__init__()
        layers = []
        for i in range(len(hidden_layers) - 1):
            layers.extend([
                nn.Linear(hidden_layers[i], hidden_layers[i + 1]),
                nn.LayerNorm(hidden_layers[i + 1]),
                activation,
                nn.Dropout(p=dropout_prob)
            ])
        # Input layer
        layers.insert(0, nn.Linear(input_size, hidden_layers[0]))

        # Output layer
        layers.append(nn.Linear(hidden_layers[-1], output_size))

        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)
```

here is the custom training loop that trains the model using sgd

[ ]:
```
def train_model(model, train_loader, criterion, optimizer):

        total_loss = 0.0
        total_samples=0
        correct_predictions=0

        for batch in train_loader:
            images, labels = batch

            # Flatten the images
            images = images.view(images.size(0), -1)

            # Zero the gradients
            optimizer.zero_grad()

            # Forward pass
            outputs = model(images)

            # Calculate the loss
            loss = criterion(outputs, labels)

            # Backward pass
            loss.backward()
```

```
            # Update weights
            optimizer.step()

            # Accumulate the total loss for the epoch
            total_loss+=loss.item()
              # Calculate accuracy
            _, predicted = torch.max(outputs.data, 1)
            total_samples += labels.size(0)
            correct_predictions += (predicted == labels).sum().item()

        accuracy = correct_predictions / total_samples
        return total_loss / len(train_loader),accuracy
```

This is the validate fn that tests the total loss and accuracy of a provided data

[ ]:
```
def validate(model, val_loader, criterion):
    model.eval()
    total_loss = 0.0
    all_preds = []
    all_labels = []
    with torch.no_grad():
      for inputs, labels in val_loader:
        inputs = inputs.view(inputs.size(0), -1)
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        total_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())
    accuracy = accuracy_score(all_labels, all_preds)

    return total_loss / len(val_loader), accuracy
```

here we choose all our hyerparameters and created tendordatasets from our images/labels also we defined our criterion that specifies the loss

[ ]:
```
input_size = 28 * 28
output_size = 10
hidden_layers = [128, 64, 32]
learning_rate = 0.01
num_of_epochs=10

# Create a TensorDataset
dataset = TensorDataset(train_images, train_labels)
val_data = TensorDataset(val_images, val_labels)
test_data = TensorDataset(test_images,test_labels)
# Specify batch size
```

```python
batch_size = 64

# Create a DataLoader
train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_data, batch_size=64, shuffle=False)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)
model = FlexibleNN(input_size, output_size, hidden_layers)
print(model)
# Define the criterion
criterion = nn.CrossEntropyLoss()

# Create an SGD optimizer
optimizer = SGD(model.parameters(), lr=learning_rate)
```

```
FlexibleNN(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): Linear(in_features=128, out_features=64, bias=True)
    (2): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
    (3): ReLU()
    (4): Dropout(p=0.5, inplace=False)
    (5): Linear(in_features=64, out_features=32, bias=True)
    (6): LayerNorm((32,), eps=1e-05, elementwise_affine=True)
    (7): ReLU()
    (8): Dropout(p=0.5, inplace=False)
    (9): Linear(in_features=32, out_features=10, bias=True)
  )
)
```

here is the real training that happens

[ ]:
```python
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []
# Train the model
for epoch in range(num_of_epochs):
      train_loss, train_accuracy = train_model(model, train_loader, criterion,␣
 ↪optimizer)
      val_loss, val_accuracy = validate(model, val_loader, criterion)
      train_accuracies.append(train_accuracy)
      train_losses.append(train_loss)
      val_losses.append(val_loss)
      val_accuracies.append(val_accuracy)

      print("Epoch {}/{}: Train Loss: {:.4f}, Validation Loss: {:.4f},␣
 ↪Validation Accuracy: {:.4f}".format(
          epoch + 1, num_of_epochs, train_loss, val_loss, val_accuracy))
```

```
# train_model(model, train_loader, criterion, optimizer,␣
 ↪num_epochs=num_of_epochs)
```

```
Epoch 1/10: Train Loss: 1.4143, Validation Loss: 0.5956, Validation Accuracy:
0.8719
Epoch 2/10: Train Loss: 0.3835, Validation Loss: 0.2732, Validation Accuracy:
0.9285
Epoch 3/10: Train Loss: 0.2340, Validation Loss: 0.2070, Validation Accuracy:
0.9427
Epoch 4/10: Train Loss: 0.1732, Validation Loss: 0.1654, Validation Accuracy:
0.9544
Epoch 5/10: Train Loss: 0.1393, Validation Loss: 0.1455, Validation Accuracy:
0.9589
Epoch 6/10: Train Loss: 0.1167, Validation Loss: 0.1261, Validation Accuracy:
0.9633
Epoch 7/10: Train Loss: 0.0995, Validation Loss: 0.1197, Validation Accuracy:
0.9660
Epoch 8/10: Train Loss: 0.0875, Validation Loss: 0.1119, Validation Accuracy:
0.9682
Epoch 9/10: Train Loss: 0.0769, Validation Loss: 0.1073, Validation Accuracy:
0.9690
Epoch 10/10: Train Loss: 0.0699, Validation Loss: 0.1039, Validation Accuracy:
0.9695
```

now we will test our model on test set to get final scores

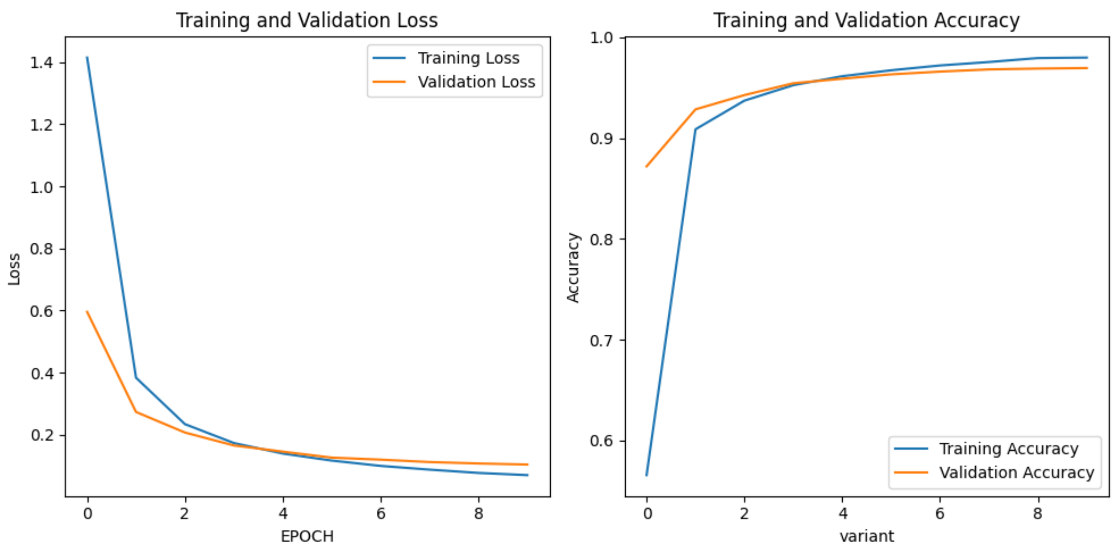
[ ]:
```
test_loss, test_accuracy = validate(model, test_loader, criterion)
print(test_loss)
print(test_accuracy)
```

```
0.09942378488274374
0.9693
```

A fn to plot:

[ ]:
```
def Plotting (train_losses,train_accuracies,val_losses,val_accuracies,variant):
  # Plotting training and validation loss
      plt.figure(figsize=(10, 5))
      plt.subplot(1, 2, 1)
      plt.plot(train_losses, label='Training Loss')
      plt.plot(val_losses, label='Validation Loss')
      plt.xlabel(variant)
      plt.ylabel('Loss')
      plt.title('Training and Validation Loss')
      plt.legend()

      # Plotting training and validation accuracy
      plt.subplot(1, 2, 2)
      plt.plot(train_accuracies, label='Training Accuracy')
```

```
    plt.plot(val_accuracies, label='Validation Accuracy')
    plt.xlabel('variant')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()
    plt.tight_layout()
    plt.show()


Plotting(train_losses=train_losses,train_accuracies=train_accuracies,val_losses=val_losses,val
```



as we se as epochs pass our model is learning more and the loss is decreasing for both train and val sets also accuracy is increasing for both

here we will validate the learning rate by trying different rates and catching the acc and loss to plot later

[ ]:
```
learning_rates = [0.1, 0.01, 0.001, 0.0001, 0.00001]
all_train_losses = []
all_train_accuracies = []
all_val_losses = []
all_val_accuracies = []

for lr in learning_rates:
    model = FlexibleNN(input_size, output_size, hidden_layers)
    optimizer = SGD(model.parameters(), lr=lr)
    train_losses = []
    val_losses = []
    train_accuracies = []
```

```
    val_accuracies = []
    for epoch in range(num_of_epochs):
        train_loss,train_accuracy = train_model(model, train_loader, criterion,␣
↪optimizer)
        val_loss, val_accuracy = validate(model, val_loader, criterion)
        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)
        val_losses.append(val_loss)
        val_accuracies.append(val_accuracy)



        print("Epoch {}/{} - Learning Rate: {:.5f}: Train Loss: {:.4f},␣
↪Validation Loss: {:.4f}, Validation Accuracy: {:.4f}".format(
            epoch + 1, num_of_epochs, lr, train_loss, val_loss, val_accuracy))
              # Append results for the current learning rate to the overall␣
↪lists
    all_train_losses.append(train_losses)
    all_train_accuracies.append(train_accuracies)
    all_val_losses.append(val_losses)
    all_val_accuracies.append(val_accuracies)
```

```
Epoch 1/10 - Learning Rate: 0.10000: Train Loss: 0.9273, Validation Loss:
0.3711, Validation Accuracy: 0.8923
Epoch 2/10 - Learning Rate: 0.10000: Train Loss: 0.2427, Validation Loss:
0.1953, Validation Accuracy: 0.9387
Epoch 3/10 - Learning Rate: 0.10000: Train Loss: 0.1652, Validation Loss:
0.1466, Validation Accuracy: 0.9553
Epoch 4/10 - Learning Rate: 0.10000: Train Loss: 0.1316, Validation Loss:
0.1255, Validation Accuracy: 0.9633
Epoch 5/10 - Learning Rate: 0.10000: Train Loss: 0.1096, Validation Loss:
0.1304, Validation Accuracy: 0.9612
Epoch 6/10 - Learning Rate: 0.10000: Train Loss: 0.0939, Validation Loss:
0.1125, Validation Accuracy: 0.9665
Epoch 7/10 - Learning Rate: 0.10000: Train Loss: 0.0822, Validation Loss:
0.1113, Validation Accuracy: 0.9684
Epoch 8/10 - Learning Rate: 0.10000: Train Loss: 0.0744, Validation Loss:
0.0969, Validation Accuracy: 0.9717
Epoch 9/10 - Learning Rate: 0.10000: Train Loss: 0.0661, Validation Loss:
0.1011, Validation Accuracy: 0.9711
Epoch 10/10 - Learning Rate: 0.10000: Train Loss: 0.0603, Validation Loss:
0.0989, Validation Accuracy: 0.9711
Epoch 1/10 - Learning Rate: 0.01000: Train Loss: 1.4183, Validation Loss:
0.5664, Validation Accuracy: 0.8849
Epoch 2/10 - Learning Rate: 0.01000: Train Loss: 0.3719, Validation Loss:
0.2642, Validation Accuracy: 0.9327
Epoch 3/10 - Learning Rate: 0.01000: Train Loss: 0.2304, Validation Loss:
```

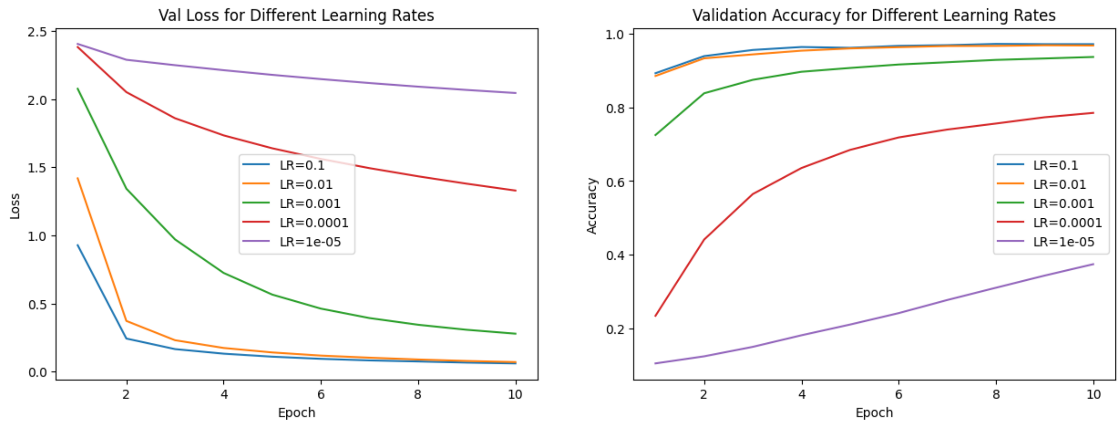
```
0.2036, Validation Accuracy: 0.9433
Epoch 4/10 - Learning Rate: 0.01000: Train Loss: 0.1736, Validation Loss:
0.1653, Validation Accuracy: 0.9534
Epoch 5/10 - Learning Rate: 0.01000: Train Loss: 0.1406, Validation Loss:
0.1477, Validation Accuracy: 0.9595
Epoch 6/10 - Learning Rate: 0.01000: Train Loss: 0.1174, Validation Loss:
0.1308, Validation Accuracy: 0.9628
Epoch 7/10 - Learning Rate: 0.01000: Train Loss: 0.1021, Validation Loss:
0.1233, Validation Accuracy: 0.9660
Epoch 8/10 - Learning Rate: 0.01000: Train Loss: 0.0890, Validation Loss:
0.1180, Validation Accuracy: 0.9663
Epoch 9/10 - Learning Rate: 0.01000: Train Loss: 0.0782, Validation Loss:
0.1144, Validation Accuracy: 0.9680
Epoch 10/10 - Learning Rate: 0.01000: Train Loss: 0.0700, Validation Loss:
0.1117, Validation Accuracy: 0.9673
Epoch 1/10 - Learning Rate: 0.00100: Train Loss: 2.0755, Validation Loss:
1.6099, Validation Accuracy: 0.7248
Epoch 2/10 - Learning Rate: 0.00100: Train Loss: 1.3425, Validation Loss:
1.1307, Validation Accuracy: 0.8379
Epoch 3/10 - Learning Rate: 0.00100: Train Loss: 0.9706, Validation Loss:
0.8315, Validation Accuracy: 0.8743
Epoch 4/10 - Learning Rate: 0.00100: Train Loss: 0.7243, Validation Loss:
0.6348, Validation Accuracy: 0.8962
Epoch 5/10 - Learning Rate: 0.00100: Train Loss: 0.5652, Validation Loss:
0.5085, Validation Accuracy: 0.9065
Epoch 6/10 - Learning Rate: 0.00100: Train Loss: 0.4622, Validation Loss:
0.4273, Validation Accuracy: 0.9159
Epoch 7/10 - Learning Rate: 0.00100: Train Loss: 0.3929, Validation Loss:
0.3692, Validation Accuracy: 0.9221
Epoch 8/10 - Learning Rate: 0.00100: Train Loss: 0.3437, Validation Loss:
0.3286, Validation Accuracy: 0.9284
Epoch 9/10 - Learning Rate: 0.00100: Train Loss: 0.3070, Validation Loss:
0.2980, Validation Accuracy: 0.9323
Epoch 10/10 - Learning Rate: 0.00100: Train Loss: 0.2784, Validation Loss:
0.2742, Validation Accuracy: 0.9363
Epoch 1/10 - Learning Rate: 0.00010: Train Loss: 2.3816, Validation Loss:
2.1749, Validation Accuracy: 0.2347
Epoch 2/10 - Learning Rate: 0.00010: Train Loss: 2.0512, Validation Loss:
1.9393, Validation Accuracy: 0.4413
Epoch 3/10 - Learning Rate: 0.00010: Train Loss: 1.8595, Validation Loss:
1.7886, Validation Accuracy: 0.5647
Epoch 4/10 - Learning Rate: 0.00010: Train Loss: 1.7337, Validation Loss:
1.6820, Validation Accuracy: 0.6352
Epoch 5/10 - Learning Rate: 0.00010: Train Loss: 1.6390, Validation Loss:
1.5969, Validation Accuracy: 0.6844
Epoch 6/10 - Learning Rate: 0.00010: Train Loss: 1.5608, Validation Loss:
1.5246, Validation Accuracy: 0.7183
Epoch 7/10 - Learning Rate: 0.00010: Train Loss: 1.4933, Validation Loss:
```

```
1.4611, Validation Accuracy: 0.7396
Epoch 8/10 - Learning Rate: 0.00010: Train Loss: 1.4333, Validation Loss:
1.4041, Validation Accuracy: 0.7559
Epoch 9/10 - Learning Rate: 0.00010: Train Loss: 1.3789, Validation Loss:
1.3520, Validation Accuracy: 0.7729
Epoch 10/10 - Learning Rate: 0.00010: Train Loss: 1.3289, Validation Loss:
1.3039, Validation Accuracy: 0.7847
Epoch 1/10 - Learning Rate: 0.00001: Train Loss: 2.4047, Validation Loss:
2.3106, Validation Accuracy: 0.1057
Epoch 2/10 - Learning Rate: 0.00001: Train Loss: 2.2884, Validation Loss:
2.2700, Validation Accuracy: 0.1248
Epoch 3/10 - Learning Rate: 0.00001: Train Loss: 2.2490, Validation Loss:
2.2318, Validation Accuracy: 0.1505
Epoch 4/10 - Learning Rate: 0.00001: Train Loss: 2.2121, Validation Loss:
2.1964, Validation Accuracy: 0.1818
Epoch 5/10 - Learning Rate: 0.00001: Train Loss: 2.1780, Validation Loss:
2.1635, Validation Accuracy: 0.2109
Epoch 6/10 - Learning Rate: 0.00001: Train Loss: 2.1466, Validation Loss:
2.1333, Validation Accuracy: 0.2419
Epoch 7/10 - Learning Rate: 0.00001: Train Loss: 2.1177, Validation Loss:
2.1056, Validation Accuracy: 0.2775
Epoch 8/10 - Learning Rate: 0.00001: Train Loss: 2.0913, Validation Loss:
2.0803, Validation Accuracy: 0.3108
Epoch 9/10 - Learning Rate: 0.00001: Train Loss: 2.0670, Validation Loss:
2.0571, Validation Accuracy: 0.3437
Epoch 10/10 - Learning Rate: 0.00001: Train Loss: 2.0447, Validation Loss:
2.0357, Validation Accuracy: 0.3746
```

[ ]:

```python
plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
for i, lr in enumerate(learning_rates):
    plt.plot(range(1, num_of_epochs + 1), all_train_losses[i], label=f'LR={lr}')
plt.title('Val Loss for Different Learning Rates')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plotting training and validation accuracy
plt.subplot(1, 2, 2)
for i, lr in enumerate(learning_rates):
    plt.plot(range(1, num_of_epochs + 1), all_val_accuracies[i],␣
 ↪label=f'LR={lr}')
plt.title('Validation Accuracy for Different Learning Rates')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.show()
```



-Higher Learning Rates (e.g., 0.1):

Training Loss: May decrease rapidly initially but may oscillate or diverge. Validation Accuracy: May fluctuate, and the model might not converge to an optimal solution. Overfitting: Higher risk of divergence and poor generalization. Computational Efficiency: Faster initial updates but may require careful tuning. Convergence Speed: Faster initial convergence, but stability is a concern.

-Moderate Learning Rates (e.g., 0.0001):

Training Loss: Decreases steadily without oscillations or divergence. Validation Accuracy: Shows stable improvement, and the model converges well. Overfitting: Moderate risk; better generalization compared to high learning rates. Computational Efficiency: Efficient updates with a balance between speed and stability. Convergence Speed: Moderate convergence speed with good stability.

-Lower Learning Rates (e.g., 0.000001):

Training Loss: Decreases more slowly but steadily. Validation Accuracy: Improves steadily with a higher likelihood of convergence. Overfitting: Lower risk, tends to generalize well. Computational Efficiency: Slower updates, but stability is prioritized. Convergence Speed: Slower but more stable convergence.

as we can see the optimal lr is 0.1

now we will try different batch sizes

[ ]:
```
batch_sizes = [32, 64, 128, 256, 512]
all_train_losses = []
all_train_accuracies = []
all_val_losses = []
all_val_accuracies = []

for batch_size in batch_sizes:
    model = FlexibleNN(input_size, output_size, hidden_layers)
    train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

```
    val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=False)
    optimizer = SGD(model.parameters(), lr=learning_rate)
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(num_of_epochs):
        train_loss, train_accuracy = train_model(model, train_loader,␣
↪criterion, optimizer)
        val_loss, val_accuracy = validate(model, val_loader, criterion)
        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)
        val_losses.append(val_loss)
        val_accuracies.append(val_accuracy)

        print("Epoch {}/{} - Batch Size: {}: Train Loss: {:.4f}, Validation␣
↪Loss: {:.4f}, Validation Accuracy: {:.4f}".format(
            epoch + 1, num_of_epochs, batch_size, train_loss, val_loss,␣
↪val_accuracy))
    all_train_losses.append(train_losses)
    all_train_accuracies.append(train_accuracies)
    all_val_losses.append(val_losses)
    all_val_accuracies.append(val_accuracies)
```

```
Epoch 1/10 - Batch Size: 32: Train Loss: 1.1597, Validation Loss: 0.4261,
Validation Accuracy: 0.8963
Epoch 2/10 - Batch Size: 32: Train Loss: 0.2819, Validation Loss: 0.2081,
Validation Accuracy: 0.9403
Epoch 3/10 - Batch Size: 32: Train Loss: 0.1765, Validation Loss: 0.1543,
Validation Accuracy: 0.9559
Epoch 4/10 - Batch Size: 32: Train Loss: 0.1351, Validation Loss: 0.1403,
Validation Accuracy: 0.9611
Epoch 5/10 - Batch Size: 32: Train Loss: 0.1089, Validation Loss: 0.1251,
Validation Accuracy: 0.9638
Epoch 6/10 - Batch Size: 32: Train Loss: 0.0935, Validation Loss: 0.1142,
Validation Accuracy: 0.9666
Epoch 7/10 - Batch Size: 32: Train Loss: 0.0809, Validation Loss: 0.1064,
Validation Accuracy: 0.9689
Epoch 8/10 - Batch Size: 32: Train Loss: 0.0722, Validation Loss: 0.1056,
Validation Accuracy: 0.9680
Epoch 9/10 - Batch Size: 32: Train Loss: 0.0640, Validation Loss: 0.0967,
Validation Accuracy: 0.9712
Epoch 10/10 - Batch Size: 32: Train Loss: 0.0574, Validation Loss: 0.1016,
Validation Accuracy: 0.9699
Epoch 1/10 - Batch Size: 64: Train Loss: 1.4080, Validation Loss: 0.5613,
Validation Accuracy: 0.8877
```

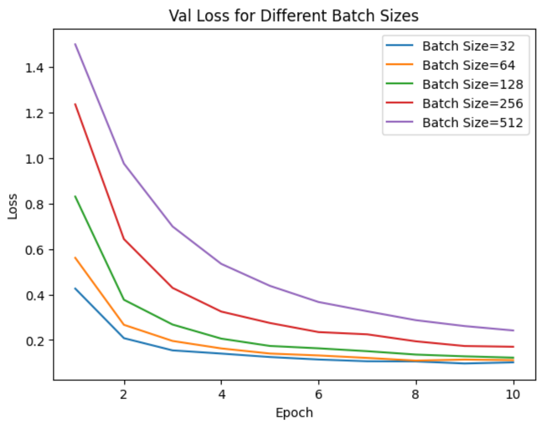
```
Epoch 2/10 - Batch Size: 64: Train Loss: 0.3624, Validation Loss: 0.2668,
Validation Accuracy: 0.9301
Epoch 3/10 - Batch Size: 64: Train Loss: 0.2244, Validation Loss: 0.1958,
Validation Accuracy: 0.9462
Epoch 4/10 - Batch Size: 64: Train Loss: 0.1687, Validation Loss: 0.1629,
Validation Accuracy: 0.9543
Epoch 5/10 - Batch Size: 64: Train Loss: 0.1335, Validation Loss: 0.1404,
Validation Accuracy: 0.9609
Epoch 6/10 - Batch Size: 64: Train Loss: 0.1107, Validation Loss: 0.1320,
Validation Accuracy: 0.9610
Epoch 7/10 - Batch Size: 64: Train Loss: 0.0967, Validation Loss: 0.1212,
Validation Accuracy: 0.9647
Epoch 8/10 - Batch Size: 64: Train Loss: 0.0842, Validation Loss: 0.1094,
Validation Accuracy: 0.9687
Epoch 9/10 - Batch Size: 64: Train Loss: 0.0752, Validation Loss: 0.1139,
Validation Accuracy: 0.9670
Epoch 10/10 - Batch Size: 64: Train Loss: 0.0672, Validation Loss: 0.1114,
Validation Accuracy: 0.9673
Epoch 1/10 - Batch Size: 128: Train Loss: 1.6558, Validation Loss: 0.8308,
Validation Accuracy: 0.8538
Epoch 2/10 - Batch Size: 128: Train Loss: 0.5376, Validation Loss: 0.3770,
Validation Accuracy: 0.9151
Epoch 3/10 - Batch Size: 128: Train Loss: 0.3079, Validation Loss: 0.2680,
Validation Accuracy: 0.9337
Epoch 4/10 - Batch Size: 128: Train Loss: 0.2291, Validation Loss: 0.2060,
Validation Accuracy: 0.9469
Epoch 5/10 - Batch Size: 128: Train Loss: 0.1839, Validation Loss: 0.1738,
Validation Accuracy: 0.9532
Epoch 6/10 - Batch Size: 128: Train Loss: 0.1541, Validation Loss: 0.1633,
Validation Accuracy: 0.9560
Epoch 7/10 - Batch Size: 128: Train Loss: 0.1342, Validation Loss: 0.1507,
Validation Accuracy: 0.9570
Epoch 8/10 - Batch Size: 128: Train Loss: 0.1175, Validation Loss: 0.1359,
Validation Accuracy: 0.9623
Epoch 9/10 - Batch Size: 128: Train Loss: 0.1062, Validation Loss: 0.1282,
Validation Accuracy: 0.9646
Epoch 10/10 - Batch Size: 128: Train Loss: 0.0961, Validation Loss: 0.1223,
Validation Accuracy: 0.9659
Epoch 1/10 - Batch Size: 256: Train Loss: 1.8998, Validation Loss: 1.2363,
Validation Accuracy: 0.8145
Epoch 2/10 - Batch Size: 256: Train Loss: 0.8813, Validation Loss: 0.6439,
Validation Accuracy: 0.8893
Epoch 3/10 - Batch Size: 256: Train Loss: 0.5151, Validation Loss: 0.4292,
Validation Accuracy: 0.9107
Epoch 4/10 - Batch Size: 256: Train Loss: 0.3671, Validation Loss: 0.3250,
Validation Accuracy: 0.9266
Epoch 5/10 - Batch Size: 256: Train Loss: 0.2921, Validation Loss: 0.2748,
Validation Accuracy: 0.9326
```

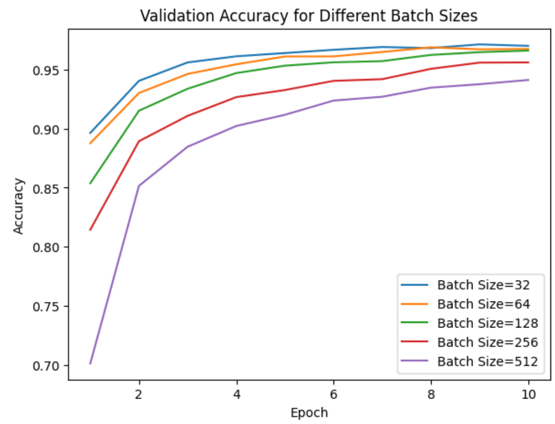
```
Epoch 6/10 - Batch Size: 256: Train Loss: 0.2452, Validation Loss: 0.2348,
Validation Accuracy: 0.9403
Epoch 7/10 - Batch Size: 256: Train Loss: 0.2114, Validation Loss: 0.2249,
Validation Accuracy: 0.9417
Epoch 8/10 - Batch Size: 256: Train Loss: 0.1867, Validation Loss: 0.1942,
Validation Accuracy: 0.9505
Epoch 9/10 - Batch Size: 256: Train Loss: 0.1675, Validation Loss: 0.1735,
Validation Accuracy: 0.9557
Epoch 10/10 - Batch Size: 256: Train Loss: 0.1512, Validation Loss: 0.1702,
Validation Accuracy: 0.9559
Epoch 1/10 - Batch Size: 512: Train Loss: 2.0277, Validation Loss: 1.5002,
Validation Accuracy: 0.7015
Epoch 2/10 - Batch Size: 512: Train Loss: 1.2016, Validation Loss: 0.9758,
Validation Accuracy: 0.8516
Epoch 3/10 - Batch Size: 512: Train Loss: 0.8251, Validation Loss: 0.6993,
Validation Accuracy: 0.8847
Epoch 4/10 - Batch Size: 512: Train Loss: 0.6087, Validation Loss: 0.5349,
Validation Accuracy: 0.9022
Epoch 5/10 - Batch Size: 512: Train Loss: 0.4783, Validation Loss: 0.4382,
Validation Accuracy: 0.9117
Epoch 6/10 - Batch Size: 512: Train Loss: 0.3957, Validation Loss: 0.3670,
Validation Accuracy: 0.9237
Epoch 7/10 - Batch Size: 512: Train Loss: 0.3391, Validation Loss: 0.3263,
Validation Accuracy: 0.9269
Epoch 8/10 - Batch Size: 512: Train Loss: 0.2993, Validation Loss: 0.2872,
Validation Accuracy: 0.9346
Epoch 9/10 - Batch Size: 512: Train Loss: 0.2682, Validation Loss: 0.2613,
Validation Accuracy: 0.9375
Epoch 10/10 - Batch Size: 512: Train Loss: 0.2443, Validation Loss: 0.2420,
Validation Accuracy: 0.9411
```

[ ]:

```
plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
for i, batch_size in enumerate(batch_sizes):
    plt.plot(range(1, num_of_epochs + 1), all_val_losses[i], label=f'Batch␣
↪Size={batch_size}')
plt.title('Val Loss for Different Batch Sizes')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plotting training and validation accuracy
plt.subplot(1, 2, 2)
for i, batch_size in enumerate(batch_sizes):
    plt.plot(range(1, num_of_epochs + 1), all_val_accuracies[i], label=f'Batch␣
↪Size={batch_size}')
plt.title('Validation Accuracy for Different Batch Sizes')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```





-Smaller Batch Sizes (e.g., 32, 64):

Training Loss: Decreases more erratically due to noisy updates. Validation Accuracy: May achieve higher accuracy initially but has a risk of overfitting. Overfitting: Higher risk due to quicker adaptation to the training data. Computational Efficiency: Faster updates but potentially less GPU utilization. Convergence Speed: Faster convergence but may converge to a suboptimal solution quickly.

-Larger Batch Sizes (e.g., 256, 512):

Training Loss: Decreases more steadily, resulting in smoother convergence. Validation Accuracy: May increase more steadily and generalize better. Overfitting: Lower risk due to a more representative sample in each update. Computational Efficiency: More GPU utilization, potentially faster overall training time. Convergence Speed: Slower convergence in terms of updates per epoch but may converge to a more stable solution.