DATA_598_HW_2

January 20, 2022

1

Homework 2: Auto Differentiation and Data Augmentation

Apoorv Sharma

<center> DATA 598 (Winter 2022), University of Washington </center>

[3]:
```
import torch
import numpy as np

from torchvision.datasets import FashionMNIST
from torch.nn.functional import cross_entropy
import torchvision.transforms as transforms

import pickle

import matplotlib.pyplot as plt
%matplotlib inline
```

1.1 1. Edge cases of automatic differentiation

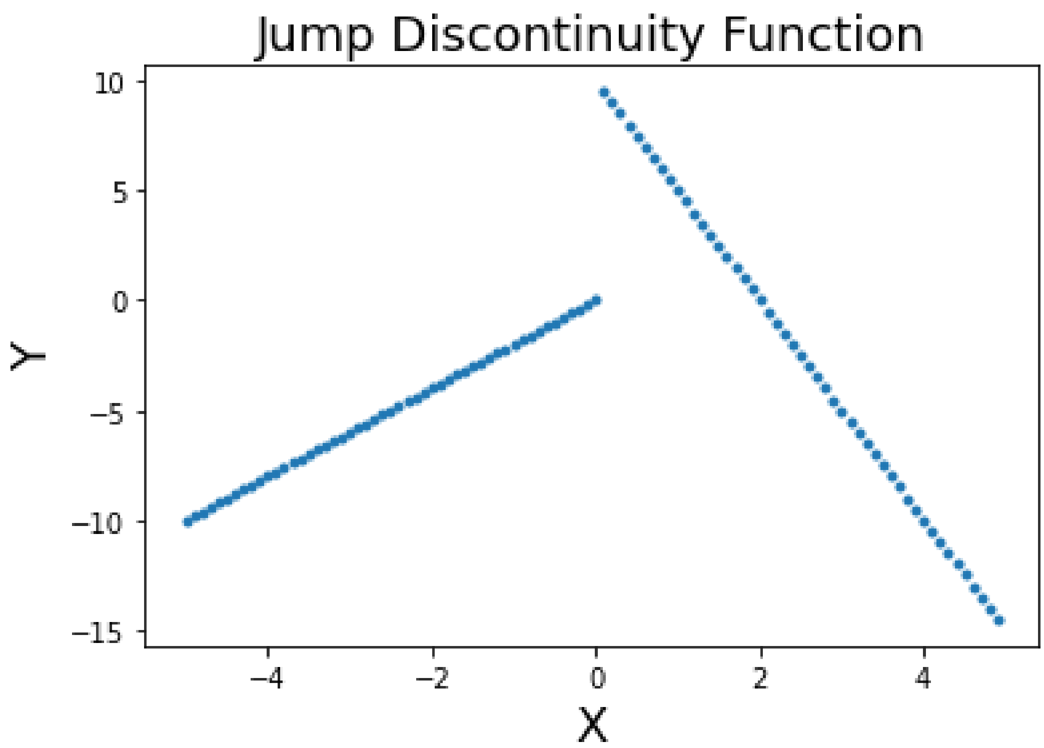
1.1.1 1.1 Derivatives of a discontinuous function

Define and plot a (mathematical) function f : R → R which is discontinuous at $\hat{x}$ with a jump discontinuity. Clearly show the point at which f is discontinuous and indicate whether it is right continuous or left continuous.

[4]:
```
def my_discontinious_fn(x):
    x_hat = 0
    if x <= x_hat:
        return 2*x
    else:
        return -5*x + 10

x_list = np.arange(-5, 5, 0.1)
y_list = [my_discontinious_fn(x) for x in x_list]

f = plt.figure()
ax = f.gca()
```

```
ax.plot(x_list, y_list, '.')
ax.set_title('Jump Discontinuity Function', fontsize=18)
ax.set_ylabel('Y', fontsize=18)
ax.set_xlabel('X', fontsize=18)

print('This function is *left* continuous')
```

```
This function is *left* continuous
```



Implement f as a DiffProg function in PyTorch so that PyTorch returns a derivative of 0 at $\hat{x}$, our point of discontinuity

[5]:
```
class MyDiscontiniousFn(torch.autograd.Function): # subclass `torch.autograd.
 ↪Function`

    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)  # save the result
        x_hat = 0
        if x <= x_hat:
            return 2*x
        else:
            return -5*x + 10
```

```python
    @staticmethod
    def backward(ctx, z):
        x = ctx.saved_tensors[0]
        x_hat = 0
        if x == x_hat:
            fprime = 0
        elif x > x_hat:
            fprime = -5
        else:
            fprime = 2

        return z * fprime
```

[6]:
```python
x = torch.rand(1, requires_grad=True)
y = MyDiscontiniousFn.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = -1 * torch.ones(1, requires_grad=True)
y = MyDiscontiniousFn.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = torch.zeros(1, requires_grad=True)
y = MyDiscontiniousFn.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')
```

```
y: tensor([7.0149], grad_fn=<MyDiscontiniousFnBackward>), y_prime: tensor([-5.])
y: tensor([-2.], grad_fn=<MyDiscontiniousFnBackward>), y_prime: tensor([2.])
y: tensor([0.], grad_fn=<MyDiscontiniousFnBackward>), y_prime: tensor([0.])
```

Implement f again in DiffProg so that PyTorch now returns a derivative of -1728 at exactly the same point $\hat{x}$.

[7]:
```python
class MyDiscontiniousFn(torch.autograd.Function): # subclass `torch.autograd.
 ↪Function`

    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)  # save the result
        x_hat = 0
        if x <= x_hat:
            return 2*x
        else:
            return -5*x + 10
```

```python
    @staticmethod
    def backward(ctx, z):
        x = ctx.saved_tensors[0]
        x_hat = 0
        if x == x_hat:
            fprime = -1728
        elif x > x_hat:
            fprime = -5
        else:
            fprime = 2

        return z * fprime
```

[8]:
```python
x = torch.rand(1, requires_grad=True)
y = MyDiscontiniousFn.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = -1 * torch.ones(1, requires_grad=True)
y = MyDiscontiniousFn.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = torch.zeros(1, requires_grad=True)
y = MyDiscontiniousFn.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')
```

```
y: tensor([6.7683], grad_fn=<MyDiscontiniousFnBackward>), y_prime: tensor([-5.])
y: tensor([-2.], grad_fn=<MyDiscontiniousFnBackward>), y_prime: tensor([2.])
y: tensor([0.], grad_fn=<MyDiscontiniousFnBackward>), y_prime: tensor([-1728.])
```

1.1.2 1.2 Inconsistent derivatives of a differentiable function

Consider the (mathematical) function g(x) = x^2. Clearly, g is differentiable everywhere.

Implement g as a DiffProg function in PyTorch so that PyTorch returns a derivative of 0 at $\hat{x} = 0$.

[9]:
```python
class XPowerTwo(torch.autograd.Function): # subclass `torch.autograd.Function`

    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)  # save the result
        return x**2

    @staticmethod
    def backward(ctx, z):
        x = ctx.saved_tensors[0]
```

```
        fprime = 2 * x
        return z * fprime
```

[10]:
```
x = torch.rand(1, requires_grad=True)
y = XPowerTwo.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = -1 * torch.rand(1, requires_grad=True)
y = XPowerTwo.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = torch.zeros(1, requires_grad=True)
y = XPowerTwo.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')
```

```
y: tensor([0.3109], grad_fn=<XPowerTwoBackward>), y_prime: tensor([1.1152])
y: tensor([0.0779], grad_fn=<XPowerTwoBackward>), y_prime: tensor([-0.5582])
y: tensor([0.], grad_fn=<XPowerTwoBackward>), y_prime: tensor([0.])
```

Implement g again in DiffProg so that PyTorch now returns a derivative of 897 at exactly the same point $\hat{x} = 0$.

[11]:
```
class XPowerTwo(torch.autograd.Function): # subclass `torch.autograd.Function`

    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)  # save the result
        return x**2

    @staticmethod
    def backward(ctx, z):
        x = ctx.saved_tensors[0]
        if x == 0:
           fprime =  897
        else:
            fprime = 2 * x
        return z * fprime
```

[12]:
```
x = torch.rand(1, requires_grad=True)
y = XPowerTwo.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = -1 * torch.rand(1, requires_grad=True)
y = XPowerTwo.apply(x)
```

```
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = torch.zeros(1, requires_grad=True)
y = XPowerTwo.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')
```

```
y: tensor([0.3479], grad_fn=<XPowerTwoBackward>), y_prime: tensor([1.1797])
y: tensor([0.3503], grad_fn=<XPowerTwoBackward>), y_prime: tensor([-1.1838])
y: tensor([0.], grad_fn=<XPowerTwoBackward>), y_prime: tensor([897.])
```

1.1.3 1.3 Derivatives with loops: When is it valid?

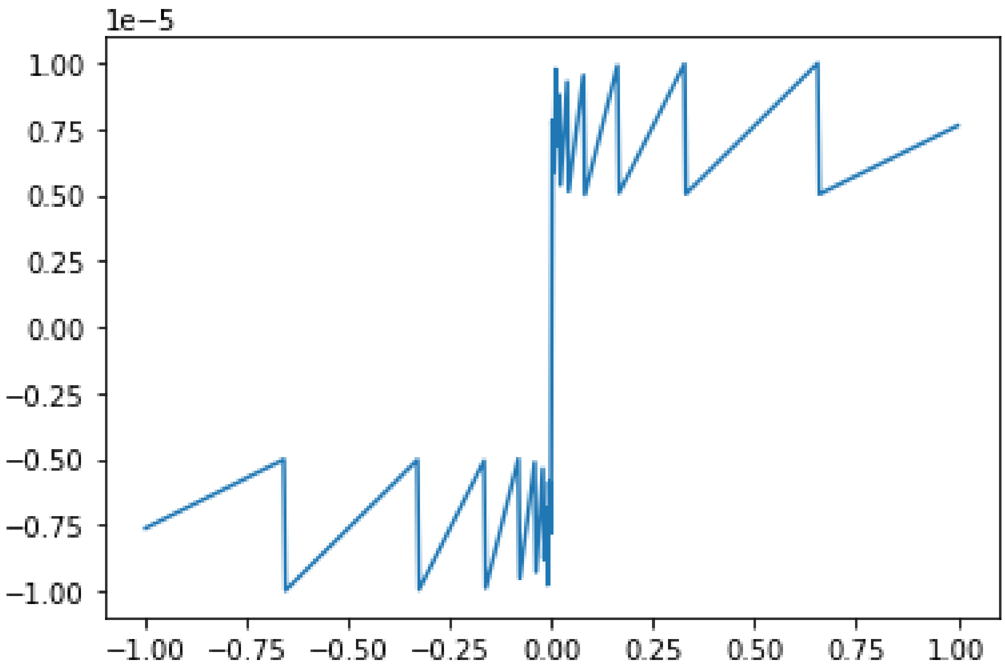
Write a DiffProg function in PyTorch which takes an input x_0 and iteratively updates $x_{t+1} \leftarrow \frac{x_t}{2}$ until a stopping criterion $|x_t| < 10^{-6}$ is satisfied.

[13]:
```
class IterativeUpdate(torch.autograd.Function): # subclass `torch.autograd.
 ↪Function`

    @staticmethod
    def forward(ctx, x):
        n = torch.zeros(1, requires_grad=False)
        while abs(x) >= 10e-6:
            x = x/2
            n[0] += 1

        ctx.save_for_backward(n)  # save the result
        return x

    @staticmethod
    def backward(ctx, z):
        n = ctx.saved_tensors[0]
        fprime = 0.5 ** n
        return z * fprime
```

Plot this function in the range $[-1, 1]$. Are the derivatives of this function well-defined everywhere?

[14]:
```
x_list = np.linspace(-1, 1, 500)
y_list = [IterativeUpdate.apply(x) for x in x_list]
plt.plot(x_list, y_list, '-')
```

[14]:
```
[<matplotlib.lines.Line2D at 0x7f3d905c3370>]
```



[15]:
```
x = torch.rand(1, requires_grad=True)
y = IterativeUpdate.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = -1 * torch.rand(1, requires_grad=True)
y = IterativeUpdate.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')

x = torch.zeros(1, requires_grad=True)
y = IterativeUpdate.apply(x)
y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(f'y: {y}, y_prime: {y_prime}')
```

```
y: tensor([9.8362e-06], grad_fn=<IterativeUpdateBackward>), y_prime:
tensor([1.5259e-05])
y: tensor([-8.6255e-06], grad_fn=<IterativeUpdateBackward>), y_prime:
tensor([0.0002])
y: tensor([0.], grad_fn=<IterativeUpdateBackward>), y_prime: tensor([1.])
```

Find a point $\hat{x}$ such that implementing the stopping criterion as $|x_t| < 10^{-6}$ or $|x_t| \quad 10^{-6}$ changes the value of the derivative returned by PyTorch. Is the derivative mathematically well-defined at $\hat{x}$.

[16]:
```
def stopping_criterion_greater_or_equal(x):
    while abs(x) >= 10e-6:
        x = x/2
    return x

def stopping_criterion_greater(x):
    while abs(x) > 10e-6:
        x = x/2
    return x

n = 3
x = 10e-6 * 2 ** n * torch.ones(1, requires_grad=True)
y_greater_or_equal = stopping_criterion_greater_or_equal(x)
y_greater = stopping_criterion_greater(x)
y_prime_greater_or_equal = torch.autograd.grad(outputs=y_greater_or_equal,␣
 ↪inputs=[x])[0]
y_prime_greater = torch.autograd.grad(outputs=y_greater, inputs=[x])[0]

print(f'x: {x}, y_prime_greater_or_equal: {y_prime_greater_or_equal}')
print(f'x: {x}, y_prime_greater: {y_prime_greater}')
```

```
x: tensor([8.0000e-05], grad_fn=<MulBackward0>), y_prime_greater_or_equal:
tensor([0.0625])
x: tensor([8.0000e-05], grad_fn=<MulBackward0>), y_prime_greater:
tensor([0.1250])
```

Write out the (mathematical) function : R → R which is implemented by this DiffProg function.

Let x_0 be the initial starting value and x_t be the value after the tth iteration. We define the function above as follows:

$$x_t = \begin{cases} \mid x_t \mid, & \text{if } \mid x_t \mid \leq 10^{-6} \\ \frac{x_t}{2}, & \text{otherwise} \end{cases}$$

1.1.4 1.4 When can we not use branches in differentiable programs?

Plot this function. Is it continuous? Is it differentiable almost everywhere?

[17]:
```
def my_binary_function(x):
    if x >= 0:
        return 1

    return 0
```

Implement this in PyTorch. Try to compute its derivatives. What do we get?

This code has been commented out on purpose. We get an error when this code is run

[18]:
```
# x = torch.rand(1, requires_grad=True)
# y = my_binary_function(x)
# y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
# print(f'y: {y}, y_prime: {y_prime}')

# x = -1 * torch.rand(1, requires_grad=True)
# y = my_binary_function(x)
# y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
# print(f'y: {y}, y_prime: {y_prime}')

# x = torch.zeros(1, requires_grad=True)
# y = my_binary_function(x)
# y_prime = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
# print(f'y: {y}, y_prime: {y_prime}')
```

Can we train a differentiable program containing this function as a component using stochastic gradient descent? Why or why not? Justify your answer in words

No, we cannot train a differentiable program containing this function as a component using SGD. The main reason is because SGD computes the gradient so it can move in the direction with the steepest descent.

However, this function always returns a derivative of 0. As a result, the algorithm will fail to converge and move. Consequently, no learning will occur

1.2 2. Data Augmentation

Data augmentation can be applied at training time or testing time.

- Training time: in each iteration, we sample a minibatch, take one transformation per-image and use those instead to compute the minibatch stochastic gradient. The rest of the training loop continues as usual.
- Test time: we predict an output for an image x as follows. Take augmentations x1, x2, · · · , xT of x. For each augmented image xi, obtain prediction yi. The combined prediction y for image x is obtained by taking a majority vote from y1, … , yT. Note that the augmentations can only be used to compute the accuracy but not the loss.

In this exercise, we will try four combinations: 1. No data augmentation for training or testing 2. Use data augmentation for training but not for testing 3. Use data augmentation for testing but not for training 4. Use data augmentation for both training and testing

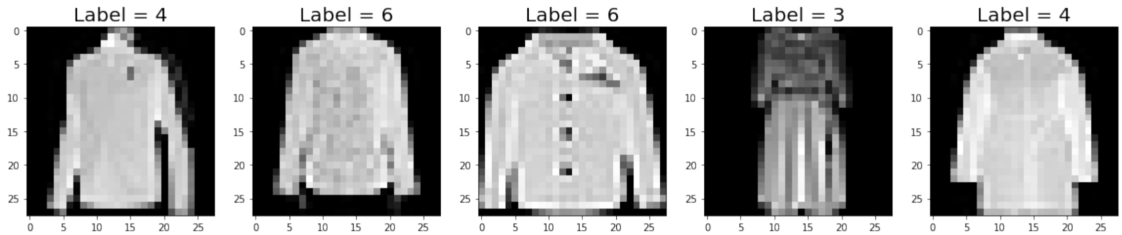
Here are the details: * The setup is identical to the lab. Take the FashionMNIST dataset and randomly subsample 10% of its training set to work with. As a test set, we will use the full test set of FashionMNIST. * We will use a convolutional neural network defined in the lab. * Use a batch size of 16 and a learning rate of 0.04. * Train the model for 100 passes through the data or until you observe perfect interpolation of the training data (i.e., the training accuracy is 100%). * We will use a random crop and a random rotation as our transformations. * For testing time, use T = 8 augmentations for each image.

The deliverables are: 1. Report the final test accuracy for each of the 4 settings considered above. 2. Make 4 plots, one each for the train loss, train accuracy, test loss and test accuracy over the course of training (i.e., the metric on the y-axis and number of effective passes on the x-axis). Plot all 4 lines on the same plot.

Hint: You may use the function `transform_selected_data` defined in this week's demo to perform the data augmentations.

[19]:
```
# download dataset (~117M in size)
train_dataset = FashionMNIST('./data', train=True, download=True)
X_train = train_dataset.data # torch tensor of type uint8
y_train = train_dataset.targets # torch tensor of type Long
test_dataset = FashionMNIST('./data', train=False, download=True)
X_test = test_dataset.data
y_test = test_dataset.targets

# choose a subsample of 10% of the data:
idxs_train = torch.from_numpy(
    np.random.choice(X_train.shape[0], replace=False, size=X_train.shape[0]//
 ↪10)).long()
X_train, y_train = X_train[idxs_train], y_train[idxs_train]

print(f'X_train.shape = {X_train.shape}')
print(f'n_train: {X_train.shape[0]}, n_test: {X_test.shape[0]}')
print(f'Image size: {X_train.shape[1:]}')

f, ax = plt.subplots(1, 5, figsize=(20, 4))
for i, idx in enumerate(np.random.choice(X_train.shape[0], 5)):
    ax[i].imshow(X_train[idx], cmap='gray', vmin=0, vmax=255)
    ax[i].set_title(f'Label = {y_train[idx]}', fontsize=20)

# Normalize dataset: pixel values lie between 0 and 255
# Normalize them so the pixelwise mean is zero and standard deviation is 1

X_train = X_train.float()  # convert to float32
X_train = X_train.view(-1, 784)
mean, std = X_train.mean(axis=0), X_train.std(axis=0)
X_train = (X_train - mean[None, :]) / (std[None, :] + 1e-6)  # avoid divide by␣
 ↪zero

X_test = X_test.float()
X_test = X_test.view(-1, 784)
X_test = (X_test - mean[None, :]) / (std[None, :] + 1e-6)

n_class = np.unique(y_train).shape[0]
```

```
X_train.shape = torch.Size([6000, 28, 28])
n_train: 6000, n_test: 10000
Image size: torch.Size([28, 28])
```



[20]:

```python
def transform_selected_data(X):
    # X is of shape (B, 784), where B is the batch_size
    X = X.view(-1, 28, 28)  # reshape to 28x28
    transform1 = transforms.RandomResizedCrop((28, 28), scale=(0.75, 1.0), ratio=(0.75, 1.33), interpolation=2)
    transform2 = transforms.RandomRotation((-10, 10))
    X_transformed = transform2(transform1(X))
    return X_transformed.view(-1, 784) # reshape into a vector
```

[21]:

```python
def compute_objective(net, X, y):
    """ Compute the multinomial logistic loss.
        net is a module
        X of shape (n, d) and y of shape (n,)
    """
    # send
    score = net(X)
    # PyTorch's function cross_entropy computes the multinomial logistic loss
    return cross_entropy(input=score, target=y, reduction='mean')

@torch.no_grad()
def compute_accuracy(net, X, y, num_augmentations=0, augment_data=False):
    """ Compute the classification accuracy
        ws is a list of tensors of consistent shapes
        X of shape (n, d) and y of shape (n,)
    """
    if augment_data:
        aug_predictions = torch.empty(num_augmentations, X.shape[0])
        for i in range(num_augmentations):
            score = net(transform_selected_data(X)) #get prob for current aug
            curr_aug_predictions = torch.argmax(score, axis=1).reshape(1,-1) #class with highest score is predicted
            aug_predictions[i] = curr_aug_predictions #assign the curr aug prediction to each row

        predictions = torch.mode(aug_predictions, 0).values #take the column-wise mode
```

```python
    else:
        score = net(X)
        predictions = torch.argmax(score, axis=1)  # class with highest score is predicted

    return (predictions == y).sum() * 1.0 / y.shape[0]

@torch.no_grad()
def compute_logs(net, verbose=False, num_augmentations=0, augment_data=False):
    train_loss = compute_objective(net, X_train, y_train)
    test_loss = compute_objective(net, X_test, y_test)
    train_accuracy = compute_accuracy(net, X_train, y_train, num_augmentations, augment_data)
    test_accuracy = compute_accuracy(net, X_test, y_test, num_augmentations, augment_data)
    if verbose:
        print(('Train Loss = {:.3f}, Train Accuracy = {:.3f}, ' +
               'Test Loss = {:.3f}, Test Accuracy = {:.3f}').format(
                train_loss.item(), train_accuracy.item(),
                test_loss.item(), test_accuracy.item())
    )
    return (train_loss, train_accuracy, test_loss, test_accuracy)

def minibatch_sgd_one_pass(net, X, y, learning_rate, batch_size, verbose=False, num_augmentations=0, augment_data=False):
    num_examples = X.shape[0]
    average_loss = 0.0
    num_updates = int(round(num_examples / batch_size))
    for i in range(num_updates):
        idxs = np.random.choice(num_examples, size=(batch_size,))
        X_data = X[idxs]

        if augment_data:
            X_data = transform_selected_data(X_data)

        # compute the objective.
        objective = compute_objective(net, X_data, y[idxs])
        average_loss = 0.99 * average_loss + 0.01 * objective.item()
        if verbose and (i+1) % 100 == 0:
            print(average_loss)

        gradients = torch.autograd.grad(outputs=objective, inputs=net.parameters())

        with torch.no_grad():
            for (w, g) in zip(net.parameters(), gradients):
                w -= learning_rate * g
```

```
    return net
```

[22]:
```
class MyConvNet(torch.nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.conv_ensemble_1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 16, kernel_size=5, padding=2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv_ensemble_2 = torch.nn.Sequential(
            torch.nn.Conv2d(16, 32, kernel_size=5, padding=2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.fully_connected_layer = torch.nn.Linear(7*7*32, 10)

    def forward(self, x):
        x = x.view(-1, 1, 28, 28)  # reshape input; convolutions need a channel
        out = self.conv_ensemble_1(x)  # first convolution + relu + pooling
        out = self.conv_ensemble_2(out) # second convolution + relu + pooling
        out = out.view(out.shape[0], -1)  # flatten output
        out = self.fully_connected_layer(out)  # output layer
        return out
```

[23]:
```
learning_rate = 0.04
num_augmentations = 8
verbose = False
cases_list = []
batch_size = 16

aug_data_rules = [(False, False), (True, False), (False, True), (True, True)]
for i, aug_data in enumerate(aug_data_rules):
    augment_train, augment_test = aug_data
    logs = []

    print(f'Starting Case {i+1} with Augment Train={augment_train} and Augment␣
↪Test={augment_test}')

    model = MyConvNet(num_classes=10)
    print('Iteration 0', end=', ')
    logs.append(compute_logs(model, True, num_augmentations, augment_test))

    # Compute 100 iterations OR until complete interpolation
    for j in range(100):
        model = minibatch_sgd_one_pass(model, X_train, y_train, learning_rate,␣
↪batch_size, verbose, num_augmentations, augment_train)
        print(f'Iteration {j+1}', end=', ')
        log = compute_logs(model, True, num_augmentations, augment_test)
```

```python
        logs.append(log)
        if log[1].item() == 1: #check if complete interpolation has been reached
            break

    torch.save(model.state_dict(), f'./models/model_parms_case_{i+1}.pt')

    cases_list.append(logs)
    print('\n')

with open('./models/logs.pkl', 'wb') as f:
    pickle.dump(cases_list, f)
```

```
Starting Case 1 with Augment Train=False and Augment Test=False
Iteration 0, Train Loss = 2.317, Train Accuracy = 0.110, Test Loss = 2.316, Test
Accuracy = 0.120
Iteration 1, Train Loss = 0.467, Train Accuracy = 0.827, Test Loss = 0.530, Test
Accuracy = 0.807
Iteration 2, Train Loss = 0.370, Train Accuracy = 0.873, Test Loss = 0.462, Test
Accuracy = 0.841
Iteration 3, Train Loss = 0.343, Train Accuracy = 0.880, Test Loss = 0.458, Test
Accuracy = 0.844
Iteration 4, Train Loss = 0.322, Train Accuracy = 0.881, Test Loss = 0.464, Test
Accuracy = 0.837
Iteration 5, Train Loss = 0.280, Train Accuracy = 0.897, Test Loss = 0.455, Test
Accuracy = 0.846
Iteration 6, Train Loss = 0.232, Train Accuracy = 0.912, Test Loss = 0.420, Test
Accuracy = 0.861
Iteration 7, Train Loss = 0.210, Train Accuracy = 0.923, Test Loss = 0.415, Test
Accuracy = 0.864
Iteration 8, Train Loss = 0.198, Train Accuracy = 0.930, Test Loss = 0.440, Test
Accuracy = 0.860
Iteration 9, Train Loss = 0.173, Train Accuracy = 0.942, Test Loss = 0.438, Test
Accuracy = 0.871
Iteration 10, Train Loss = 0.159, Train Accuracy = 0.946, Test Loss = 0.436,
Test Accuracy = 0.868
Iteration 11, Train Loss = 0.149, Train Accuracy = 0.945, Test Loss = 0.481,
Test Accuracy = 0.864
Iteration 12, Train Loss = 0.113, Train Accuracy = 0.961, Test Loss = 0.454,
Test Accuracy = 0.869
Iteration 13, Train Loss = 0.123, Train Accuracy = 0.957, Test Loss = 0.460,
Test Accuracy = 0.863
Iteration 14, Train Loss = 0.103, Train Accuracy = 0.965, Test Loss = 0.517,
Test Accuracy = 0.866
Iteration 15, Train Loss = 0.133, Train Accuracy = 0.953, Test Loss = 0.558,
Test Accuracy = 0.858
Iteration 16, Train Loss = 0.071, Train Accuracy = 0.978, Test Loss = 0.512,
Test Accuracy = 0.869
```

```
Iteration 17, Train Loss = 0.068, Train Accuracy = 0.978, Test Loss = 0.540,
Test Accuracy = 0.867
Iteration 18, Train Loss = 0.047, Train Accuracy = 0.987, Test Loss = 0.537,
Test Accuracy = 0.875
Iteration 19, Train Loss = 0.043, Train Accuracy = 0.988, Test Loss = 0.553,
Test Accuracy = 0.872
Iteration 20, Train Loss = 0.047, Train Accuracy = 0.985, Test Loss = 0.583,
Test Accuracy = 0.873
Iteration 21, Train Loss = 0.035, Train Accuracy = 0.991, Test Loss = 0.589,
Test Accuracy = 0.872
Iteration 22, Train Loss = 0.039, Train Accuracy = 0.988, Test Loss = 0.623,
Test Accuracy = 0.866
Iteration 23, Train Loss = 0.028, Train Accuracy = 0.992, Test Loss = 0.648,
Test Accuracy = 0.871
Iteration 24, Train Loss = 0.034, Train Accuracy = 0.989, Test Loss = 0.642,
Test Accuracy = 0.869
Iteration 25, Train Loss = 0.018, Train Accuracy = 0.996, Test Loss = 0.685,
Test Accuracy = 0.874
Iteration 26, Train Loss = 0.019, Train Accuracy = 0.996, Test Loss = 0.699,
Test Accuracy = 0.871
Iteration 27, Train Loss = 0.018, Train Accuracy = 0.996, Test Loss = 0.721,
Test Accuracy = 0.867
Iteration 28, Train Loss = 0.046, Train Accuracy = 0.986, Test Loss = 0.744,
Test Accuracy = 0.860
Iteration 29, Train Loss = 0.023, Train Accuracy = 0.995, Test Loss = 0.701,
Test Accuracy = 0.872
Iteration 30, Train Loss = 0.018, Train Accuracy = 0.995, Test Loss = 0.721,
Test Accuracy = 0.870
Iteration 31, Train Loss = 0.008, Train Accuracy = 0.999, Test Loss = 0.742,
Test Accuracy = 0.873
Iteration 32, Train Loss = 0.005, Train Accuracy = 0.999, Test Loss = 0.755,
Test Accuracy = 0.873
Iteration 33, Train Loss = 0.004, Train Accuracy = 1.000, Test Loss = 0.764,
Test Accuracy = 0.872
Iteration 34, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 0.793,
Test Accuracy = 0.874


Starting Case 2 with Augment Train=True and Augment Test=False
Iteration 0, Train Loss = 2.286, Train Accuracy = 0.075, Test Loss = 2.288, Test
Accuracy = 0.077
```

```
/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/torchvision/transforms/transforms.py:852: UserWarning: Argument
interpolation should be of type InterpolationMode instead of int. Please, use
InterpolationMode enum.
  warnings.warn(
```

```
Iteration 1, Train Loss = 0.562, Train Accuracy = 0.790, Test Loss = 0.628, Test
```

```
Accuracy = 0.772
Iteration 2, Train Loss = 0.493, Train Accuracy = 0.821, Test Loss = 0.571, Test
Accuracy = 0.791
Iteration 3, Train Loss = 0.427, Train Accuracy = 0.844, Test Loss = 0.506, Test
Accuracy = 0.819
Iteration 4, Train Loss = 0.404, Train Accuracy = 0.854, Test Loss = 0.494, Test
Accuracy = 0.831
Iteration 5, Train Loss = 0.430, Train Accuracy = 0.840, Test Loss = 0.528, Test
Accuracy = 0.815
Iteration 6, Train Loss = 0.372, Train Accuracy = 0.869, Test Loss = 0.449, Test
Accuracy = 0.841
Iteration 7, Train Loss = 0.374, Train Accuracy = 0.867, Test Loss = 0.483, Test
Accuracy = 0.839
Iteration 8, Train Loss = 0.354, Train Accuracy = 0.867, Test Loss = 0.458, Test
Accuracy = 0.833
Iteration 9, Train Loss = 0.334, Train Accuracy = 0.878, Test Loss = 0.428, Test
Accuracy = 0.848
Iteration 10, Train Loss = 0.316, Train Accuracy = 0.882, Test Loss = 0.425,
Test Accuracy = 0.845
Iteration 11, Train Loss = 0.309, Train Accuracy = 0.888, Test Loss = 0.438,
Test Accuracy = 0.855
Iteration 12, Train Loss = 0.321, Train Accuracy = 0.878, Test Loss = 0.464,
Test Accuracy = 0.845
Iteration 13, Train Loss = 0.295, Train Accuracy = 0.895, Test Loss = 0.419,
Test Accuracy = 0.857
Iteration 14, Train Loss = 0.303, Train Accuracy = 0.887, Test Loss = 0.456,
Test Accuracy = 0.847
Iteration 15, Train Loss = 0.264, Train Accuracy = 0.900, Test Loss = 0.416,
Test Accuracy = 0.860
Iteration 16, Train Loss = 0.283, Train Accuracy = 0.894, Test Loss = 0.440,
Test Accuracy = 0.856
Iteration 17, Train Loss = 0.259, Train Accuracy = 0.903, Test Loss = 0.390,
Test Accuracy = 0.868
Iteration 18, Train Loss = 0.283, Train Accuracy = 0.897, Test Loss = 0.426,
Test Accuracy = 0.859
Iteration 19, Train Loss = 0.278, Train Accuracy = 0.898, Test Loss = 0.431,
Test Accuracy = 0.852
Iteration 20, Train Loss = 0.237, Train Accuracy = 0.915, Test Loss = 0.384,
Test Accuracy = 0.871
Iteration 21, Train Loss = 0.296, Train Accuracy = 0.892, Test Loss = 0.451,
Test Accuracy = 0.851
Iteration 22, Train Loss = 0.266, Train Accuracy = 0.896, Test Loss = 0.414,
Test Accuracy = 0.853
Iteration 23, Train Loss = 0.245, Train Accuracy = 0.909, Test Loss = 0.408,
Test Accuracy = 0.861
Iteration 24, Train Loss = 0.239, Train Accuracy = 0.912, Test Loss = 0.394,
Test Accuracy = 0.865
Iteration 25, Train Loss = 0.265, Train Accuracy = 0.900, Test Loss = 0.450,
```

Test Accuracy = 0.848
Iteration 26, Train Loss = 0.237, Train Accuracy = 0.913, Test Loss = 0.408, Test Accuracy = 0.868
Iteration 27, Train Loss = 0.248, Train Accuracy = 0.908, Test Loss = 0.438, Test Accuracy = 0.856
Iteration 28, Train Loss = 0.211, Train Accuracy = 0.924, Test Loss = 0.394, Test Accuracy = 0.868
Iteration 29, Train Loss = 0.277, Train Accuracy = 0.900, Test Loss = 0.444, Test Accuracy = 0.840
Iteration 30, Train Loss = 0.222, Train Accuracy = 0.919, Test Loss = 0.393, Test Accuracy = 0.870
Iteration 31, Train Loss = 0.258, Train Accuracy = 0.904, Test Loss = 0.457, Test Accuracy = 0.848
Iteration 32, Train Loss = 0.213, Train Accuracy = 0.919, Test Loss = 0.408, Test Accuracy = 0.868
Iteration 33, Train Loss = 0.269, Train Accuracy = 0.899, Test Loss = 0.492, Test Accuracy = 0.843
Iteration 34, Train Loss = 0.204, Train Accuracy = 0.926, Test Loss = 0.418, Test Accuracy = 0.870
Iteration 35, Train Loss = 0.211, Train Accuracy = 0.922, Test Loss = 0.438, Test Accuracy = 0.863
Iteration 36, Train Loss = 0.218, Train Accuracy = 0.922, Test Loss = 0.441, Test Accuracy = 0.862
Iteration 37, Train Loss = 0.220, Train Accuracy = 0.919, Test Loss = 0.437, Test Accuracy = 0.863
Iteration 38, Train Loss = 0.213, Train Accuracy = 0.923, Test Loss = 0.442, Test Accuracy = 0.860
Iteration 39, Train Loss = 0.238, Train Accuracy = 0.912, Test Loss = 0.454, Test Accuracy = 0.853
Iteration 40, Train Loss = 0.224, Train Accuracy = 0.916, Test Loss = 0.452, Test Accuracy = 0.857
Iteration 41, Train Loss = 0.251, Train Accuracy = 0.902, Test Loss = 0.475, Test Accuracy = 0.845
Iteration 42, Train Loss = 0.210, Train Accuracy = 0.922, Test Loss = 0.455, Test Accuracy = 0.858
Iteration 43, Train Loss = 0.198, Train Accuracy = 0.926, Test Loss = 0.440, Test Accuracy = 0.873
Iteration 44, Train Loss = 0.190, Train Accuracy = 0.929, Test Loss = 0.431, Test Accuracy = 0.870
Iteration 45, Train Loss = 0.190, Train Accuracy = 0.930, Test Loss = 0.423, Test Accuracy = 0.867
Iteration 46, Train Loss = 0.181, Train Accuracy = 0.932, Test Loss = 0.429, Test Accuracy = 0.865
Iteration 47, Train Loss = 0.218, Train Accuracy = 0.920, Test Loss = 0.472, Test Accuracy = 0.854
Iteration 48, Train Loss = 0.179, Train Accuracy = 0.936, Test Loss = 0.417, Test Accuracy = 0.870
Iteration 49, Train Loss = 0.190, Train Accuracy = 0.929, Test Loss = 0.428,

```
Test Accuracy = 0.864
Iteration 50, Train Loss = 0.183, Train Accuracy = 0.934, Test Loss = 0.432,
Test Accuracy = 0.871
Iteration 51, Train Loss = 0.188, Train Accuracy = 0.934, Test Loss = 0.417,
Test Accuracy = 0.877
Iteration 52, Train Loss = 0.185, Train Accuracy = 0.933, Test Loss = 0.445,
Test Accuracy = 0.869
Iteration 53, Train Loss = 0.174, Train Accuracy = 0.934, Test Loss = 0.416,
Test Accuracy = 0.875
Iteration 54, Train Loss = 0.198, Train Accuracy = 0.927, Test Loss = 0.439,
Test Accuracy = 0.859
Iteration 55, Train Loss = 0.175, Train Accuracy = 0.936, Test Loss = 0.423,
Test Accuracy = 0.867
Iteration 56, Train Loss = 0.172, Train Accuracy = 0.938, Test Loss = 0.449,
Test Accuracy = 0.865
Iteration 57, Train Loss = 0.191, Train Accuracy = 0.930, Test Loss = 0.470,
Test Accuracy = 0.867
Iteration 58, Train Loss = 0.193, Train Accuracy = 0.932, Test Loss = 0.441,
Test Accuracy = 0.863
Iteration 59, Train Loss = 0.168, Train Accuracy = 0.941, Test Loss = 0.404,
Test Accuracy = 0.870
Iteration 60, Train Loss = 0.174, Train Accuracy = 0.936, Test Loss = 0.406,
Test Accuracy = 0.871
Iteration 61, Train Loss = 0.166, Train Accuracy = 0.940, Test Loss = 0.431,
Test Accuracy = 0.871
Iteration 62, Train Loss = 0.172, Train Accuracy = 0.937, Test Loss = 0.437,
Test Accuracy = 0.862
Iteration 63, Train Loss = 0.152, Train Accuracy = 0.945, Test Loss = 0.415,
Test Accuracy = 0.875
Iteration 64, Train Loss = 0.167, Train Accuracy = 0.938, Test Loss = 0.449,
Test Accuracy = 0.870
Iteration 65, Train Loss = 0.175, Train Accuracy = 0.938, Test Loss = 0.466,
Test Accuracy = 0.864
Iteration 66, Train Loss = 0.213, Train Accuracy = 0.918, Test Loss = 0.498,
Test Accuracy = 0.848
Iteration 67, Train Loss = 0.162, Train Accuracy = 0.942, Test Loss = 0.466,
Test Accuracy = 0.875
Iteration 68, Train Loss = 0.197, Train Accuracy = 0.923, Test Loss = 0.461,
Test Accuracy = 0.851
Iteration 69, Train Loss = 0.162, Train Accuracy = 0.938, Test Loss = 0.453,
Test Accuracy = 0.870
Iteration 70, Train Loss = 0.154, Train Accuracy = 0.945, Test Loss = 0.436,
Test Accuracy = 0.875
Iteration 71, Train Loss = 0.171, Train Accuracy = 0.937, Test Loss = 0.452,
Test Accuracy = 0.862
Iteration 72, Train Loss = 0.162, Train Accuracy = 0.941, Test Loss = 0.459,
Test Accuracy = 0.870
Iteration 73, Train Loss = 0.154, Train Accuracy = 0.945, Test Loss = 0.433,
```

Test Accuracy = 0.870
Iteration 74, Train Loss = 0.175, Train Accuracy = 0.932, Test Loss = 0.498, Test Accuracy = 0.865
Iteration 75, Train Loss = 0.157, Train Accuracy = 0.939, Test Loss = 0.467, Test Accuracy = 0.866
Iteration 76, Train Loss = 0.162, Train Accuracy = 0.942, Test Loss = 0.466, Test Accuracy = 0.865
Iteration 77, Train Loss = 0.171, Train Accuracy = 0.937, Test Loss = 0.436, Test Accuracy = 0.859
Iteration 78, Train Loss = 0.141, Train Accuracy = 0.948, Test Loss = 0.443, Test Accuracy = 0.872
Iteration 79, Train Loss = 0.173, Train Accuracy = 0.936, Test Loss = 0.460, Test Accuracy = 0.868
Iteration 80, Train Loss = 0.155, Train Accuracy = 0.946, Test Loss = 0.444, Test Accuracy = 0.871
Iteration 81, Train Loss = 0.178, Train Accuracy = 0.936, Test Loss = 0.475, Test Accuracy = 0.864
Iteration 82, Train Loss = 0.153, Train Accuracy = 0.946, Test Loss = 0.455, Test Accuracy = 0.866
Iteration 83, Train Loss = 0.162, Train Accuracy = 0.941, Test Loss = 0.480, Test Accuracy = 0.862
Iteration 84, Train Loss = 0.153, Train Accuracy = 0.944, Test Loss = 0.480, Test Accuracy = 0.870
Iteration 85, Train Loss = 0.141, Train Accuracy = 0.948, Test Loss = 0.429, Test Accuracy = 0.872
Iteration 86, Train Loss = 0.134, Train Accuracy = 0.951, Test Loss = 0.457, Test Accuracy = 0.875
Iteration 87, Train Loss = 0.155, Train Accuracy = 0.943, Test Loss = 0.517, Test Accuracy = 0.869
Iteration 88, Train Loss = 0.145, Train Accuracy = 0.948, Test Loss = 0.481, Test Accuracy = 0.866
Iteration 89, Train Loss = 0.149, Train Accuracy = 0.950, Test Loss = 0.470, Test Accuracy = 0.868
Iteration 90, Train Loss = 0.162, Train Accuracy = 0.940, Test Loss = 0.489, Test Accuracy = 0.863
Iteration 91, Train Loss = 0.157, Train Accuracy = 0.942, Test Loss = 0.513, Test Accuracy = 0.864
Iteration 92, Train Loss = 0.145, Train Accuracy = 0.946, Test Loss = 0.442, Test Accuracy = 0.874
Iteration 93, Train Loss = 0.157, Train Accuracy = 0.939, Test Loss = 0.466, Test Accuracy = 0.863
Iteration 94, Train Loss = 0.147, Train Accuracy = 0.949, Test Loss = 0.475, Test Accuracy = 0.867
Iteration 95, Train Loss = 0.156, Train Accuracy = 0.943, Test Loss = 0.474, Test Accuracy = 0.865
Iteration 96, Train Loss = 0.155, Train Accuracy = 0.943, Test Loss = 0.469, Test Accuracy = 0.863
Iteration 97, Train Loss = 0.140, Train Accuracy = 0.951, Test Loss = 0.437,

Test Accuracy = 0.874
Iteration 98, Train Loss = 0.166, Train Accuracy = 0.937, Test Loss = 0.492, Test Accuracy = 0.857
Iteration 99, Train Loss = 0.148, Train Accuracy = 0.944, Test Loss = 0.498, Test Accuracy = 0.869
Iteration 100, Train Loss = 0.163, Train Accuracy = 0.940, Test Loss = 0.480, Test Accuracy = 0.857

Starting Case 3 with Augment Train=False and Augment Test=True
Iteration 0, Train Loss = 2.303, Train Accuracy = 0.082, Test Loss = 2.306, Test Accuracy = 0.094
Iteration 1, Train Loss = 0.505, Train Accuracy = 0.788, Test Loss = 0.570, Test Accuracy = 0.775
Iteration 2, Train Loss = 0.382, Train Accuracy = 0.815, Test Loss = 0.492, Test Accuracy = 0.796
Iteration 3, Train Loss = 0.328, Train Accuracy = 0.821, Test Loss = 0.449, Test Accuracy = 0.793
Iteration 4, Train Loss = 0.325, Train Accuracy = 0.853, Test Loss = 0.485, Test Accuracy = 0.800
Iteration 5, Train Loss = 0.301, Train Accuracy = 0.803, Test Loss = 0.484, Test Accuracy = 0.801
Iteration 6, Train Loss = 0.262, Train Accuracy = 0.822, Test Loss = 0.472, Test Accuracy = 0.791
Iteration 7, Train Loss = 0.230, Train Accuracy = 0.848, Test Loss = 0.460, Test Accuracy = 0.830
Iteration 8, Train Loss = 0.191, Train Accuracy = 0.875, Test Loss = 0.433, Test Accuracy = 0.833
Iteration 9, Train Loss = 0.174, Train Accuracy = 0.877, Test Loss = 0.439, Test Accuracy = 0.826
Iteration 10, Train Loss = 0.182, Train Accuracy = 0.805, Test Loss = 0.500, Test Accuracy = 0.770
Iteration 11, Train Loss = 0.138, Train Accuracy = 0.869, Test Loss = 0.436, Test Accuracy = 0.812
Iteration 12, Train Loss = 0.110, Train Accuracy = 0.901, Test Loss = 0.464, Test Accuracy = 0.836
Iteration 13, Train Loss = 0.106, Train Accuracy = 0.889, Test Loss = 0.503, Test Accuracy = 0.843
Iteration 14, Train Loss = 0.111, Train Accuracy = 0.861, Test Loss = 0.543, Test Accuracy = 0.823
Iteration 15, Train Loss = 0.067, Train Accuracy = 0.903, Test Loss = 0.492, Test Accuracy = 0.834
Iteration 16, Train Loss = 0.084, Train Accuracy = 0.874, Test Loss = 0.552, Test Accuracy = 0.815
Iteration 17, Train Loss = 0.206, Train Accuracy = 0.845, Test Loss = 0.727, Test Accuracy = 0.778
Iteration 18, Train Loss = 0.074, Train Accuracy = 0.885, Test Loss = 0.583, Test Accuracy = 0.830

```
Iteration 19, Train Loss = 0.044, Train Accuracy = 0.890, Test Loss = 0.594,
Test Accuracy = 0.789
Iteration 20, Train Loss = 0.034, Train Accuracy = 0.899, Test Loss = 0.583,
Test Accuracy = 0.845
Iteration 21, Train Loss = 0.073, Train Accuracy = 0.897, Test Loss = 0.710,
Test Accuracy = 0.833
Iteration 22, Train Loss = 0.026, Train Accuracy = 0.873, Test Loss = 0.606,
Test Accuracy = 0.839
Iteration 23, Train Loss = 0.017, Train Accuracy = 0.882, Test Loss = 0.635,
Test Accuracy = 0.846
Iteration 24, Train Loss = 0.018, Train Accuracy = 0.889, Test Loss = 0.651,
Test Accuracy = 0.842
Iteration 25, Train Loss = 0.030, Train Accuracy = 0.886, Test Loss = 0.677,
Test Accuracy = 0.804
Iteration 26, Train Loss = 0.014, Train Accuracy = 0.855, Test Loss = 0.701,
Test Accuracy = 0.822
Iteration 27, Train Loss = 0.015, Train Accuracy = 0.900, Test Loss = 0.694,
Test Accuracy = 0.839
Iteration 28, Train Loss = 0.007, Train Accuracy = 0.854, Test Loss = 0.723,
Test Accuracy = 0.825
Iteration 29, Train Loss = 0.005, Train Accuracy = 0.929, Test Loss = 0.707,
Test Accuracy = 0.823
Iteration 30, Train Loss = 0.003, Train Accuracy = 0.878, Test Loss = 0.728,
Test Accuracy = 0.841
Iteration 31, Train Loss = 0.003, Train Accuracy = 0.878, Test Loss = 0.745,
Test Accuracy = 0.826
Iteration 32, Train Loss = 0.002, Train Accuracy = 0.875, Test Loss = 0.758,
Test Accuracy = 0.839
Iteration 33, Train Loss = 0.002, Train Accuracy = 0.877, Test Loss = 0.777,
Test Accuracy = 0.839
Iteration 34, Train Loss = 0.002, Train Accuracy = 0.867, Test Loss = 0.791,
Test Accuracy = 0.816
Iteration 35, Train Loss = 0.002, Train Accuracy = 0.899, Test Loss = 0.799,
Test Accuracy = 0.806
Iteration 36, Train Loss = 0.002, Train Accuracy = 0.890, Test Loss = 0.802,
Test Accuracy = 0.814
Iteration 37, Train Loss = 0.001, Train Accuracy = 0.917, Test Loss = 0.805,
Test Accuracy = 0.836
Iteration 38, Train Loss = 0.001, Train Accuracy = 0.826, Test Loss = 0.815,
Test Accuracy = 0.821
Iteration 39, Train Loss = 0.001, Train Accuracy = 0.890, Test Loss = 0.818,
Test Accuracy = 0.818
Iteration 40, Train Loss = 0.001, Train Accuracy = 0.891, Test Loss = 0.827,
Test Accuracy = 0.837
Iteration 41, Train Loss = 0.001, Train Accuracy = 0.899, Test Loss = 0.835,
Test Accuracy = 0.828
Iteration 42, Train Loss = 0.001, Train Accuracy = 0.893, Test Loss = 0.835,
Test Accuracy = 0.823
```

```
Iteration 43, Train Loss = 0.001, Train Accuracy = 0.868, Test Loss = 0.845,
Test Accuracy = 0.831
Iteration 44, Train Loss = 0.001, Train Accuracy = 0.904, Test Loss = 0.852,
Test Accuracy = 0.831
Iteration 45, Train Loss = 0.001, Train Accuracy = 0.854, Test Loss = 0.858,
Test Accuracy = 0.814
Iteration 46, Train Loss = 0.001, Train Accuracy = 0.881, Test Loss = 0.859,
Test Accuracy = 0.809
Iteration 47, Train Loss = 0.001, Train Accuracy = 0.848, Test Loss = 0.865,
Test Accuracy = 0.842
Iteration 48, Train Loss = 0.001, Train Accuracy = 0.895, Test Loss = 0.866,
Test Accuracy = 0.827
Iteration 49, Train Loss = 0.001, Train Accuracy = 0.908, Test Loss = 0.867,
Test Accuracy = 0.828
Iteration 50, Train Loss = 0.001, Train Accuracy = 0.898, Test Loss = 0.873,
Test Accuracy = 0.825
Iteration 51, Train Loss = 0.001, Train Accuracy = 0.865, Test Loss = 0.881,
Test Accuracy = 0.848
Iteration 52, Train Loss = 0.001, Train Accuracy = 0.909, Test Loss = 0.885,
Test Accuracy = 0.848
Iteration 53, Train Loss = 0.001, Train Accuracy = 0.885, Test Loss = 0.888,
Test Accuracy = 0.838
Iteration 54, Train Loss = 0.001, Train Accuracy = 0.871, Test Loss = 0.889,
Test Accuracy = 0.815
Iteration 55, Train Loss = 0.001, Train Accuracy = 0.882, Test Loss = 0.891,
Test Accuracy = 0.834
Iteration 56, Train Loss = 0.001, Train Accuracy = 0.918, Test Loss = 0.896,
Test Accuracy = 0.828
Iteration 57, Train Loss = 0.001, Train Accuracy = 0.881, Test Loss = 0.901,
Test Accuracy = 0.823
Iteration 58, Train Loss = 0.001, Train Accuracy = 0.879, Test Loss = 0.905,
Test Accuracy = 0.820
Iteration 59, Train Loss = 0.001, Train Accuracy = 0.855, Test Loss = 0.907,
Test Accuracy = 0.816
Iteration 60, Train Loss = 0.001, Train Accuracy = 0.891, Test Loss = 0.911,
Test Accuracy = 0.834
Iteration 61, Train Loss = 0.001, Train Accuracy = 0.895, Test Loss = 0.910,
Test Accuracy = 0.829
Iteration 62, Train Loss = 0.001, Train Accuracy = 0.886, Test Loss = 0.918,
Test Accuracy = 0.834
Iteration 63, Train Loss = 0.001, Train Accuracy = 0.906, Test Loss = 0.917,
Test Accuracy = 0.851
Iteration 64, Train Loss = 0.001, Train Accuracy = 0.866, Test Loss = 0.921,
Test Accuracy = 0.834
Iteration 65, Train Loss = 0.000, Train Accuracy = 0.892, Test Loss = 0.924,
Test Accuracy = 0.854
Iteration 66, Train Loss = 0.000, Train Accuracy = 0.905, Test Loss = 0.926,
Test Accuracy = 0.851
```

Iteration 67, Train Loss = 0.000, Train Accuracy = 0.887, Test Loss = 0.929, Test Accuracy = 0.815
Iteration 68, Train Loss = 0.000, Train Accuracy = 0.877, Test Loss = 0.931, Test Accuracy = 0.839
Iteration 69, Train Loss = 0.000, Train Accuracy = 0.917, Test Loss = 0.932, Test Accuracy = 0.838
Iteration 70, Train Loss = 0.000, Train Accuracy = 0.870, Test Loss = 0.935, Test Accuracy = 0.833
Iteration 71, Train Loss = 0.000, Train Accuracy = 0.885, Test Loss = 0.939, Test Accuracy = 0.827
Iteration 72, Train Loss = 0.000, Train Accuracy = 0.887, Test Loss = 0.939, Test Accuracy = 0.838
Iteration 73, Train Loss = 0.000, Train Accuracy = 0.898, Test Loss = 0.942, Test Accuracy = 0.822
Iteration 74, Train Loss = 0.000, Train Accuracy = 0.886, Test Loss = 0.943, Test Accuracy = 0.828
Iteration 75, Train Loss = 0.000, Train Accuracy = 0.887, Test Loss = 0.948, Test Accuracy = 0.848
Iteration 76, Train Loss = 0.000, Train Accuracy = 0.886, Test Loss = 0.950, Test Accuracy = 0.831
Iteration 77, Train Loss = 0.000, Train Accuracy = 0.855, Test Loss = 0.951, Test Accuracy = 0.829
Iteration 78, Train Loss = 0.000, Train Accuracy = 0.878, Test Loss = 0.952, Test Accuracy = 0.841
Iteration 79, Train Loss = 0.000, Train Accuracy = 0.855, Test Loss = 0.955, Test Accuracy = 0.831
Iteration 80, Train Loss = 0.000, Train Accuracy = 0.855, Test Loss = 0.959, Test Accuracy = 0.818
Iteration 81, Train Loss = 0.000, Train Accuracy = 0.830, Test Loss = 0.958, Test Accuracy = 0.818
Iteration 82, Train Loss = 0.000, Train Accuracy = 0.893, Test Loss = 0.960, Test Accuracy = 0.823
Iteration 83, Train Loss = 0.000, Train Accuracy = 0.917, Test Loss = 0.962, Test Accuracy = 0.857
Iteration 84, Train Loss = 0.000, Train Accuracy = 0.877, Test Loss = 0.963, Test Accuracy = 0.831
Iteration 85, Train Loss = 0.000, Train Accuracy = 0.896, Test Loss = 0.968, Test Accuracy = 0.802
Iteration 86, Train Loss = 0.000, Train Accuracy = 0.806, Test Loss = 0.970, Test Accuracy = 0.847
Iteration 87, Train Loss = 0.000, Train Accuracy = 0.880, Test Loss = 0.971, Test Accuracy = 0.836
Iteration 88, Train Loss = 0.000, Train Accuracy = 0.875, Test Loss = 0.972, Test Accuracy = 0.825
Iteration 89, Train Loss = 0.000, Train Accuracy = 0.914, Test Loss = 0.974, Test Accuracy = 0.824
Iteration 90, Train Loss = 0.000, Train Accuracy = 0.893, Test Loss = 0.978, Test Accuracy = 0.821

```
Iteration 91, Train Loss = 0.000, Train Accuracy = 0.893, Test Loss = 0.976,
Test Accuracy = 0.836
Iteration 92, Train Loss = 0.000, Train Accuracy = 0.890, Test Loss = 0.979,
Test Accuracy = 0.845
Iteration 93, Train Loss = 0.000, Train Accuracy = 0.917, Test Loss = 0.980,
Test Accuracy = 0.845
Iteration 94, Train Loss = 0.000, Train Accuracy = 0.875, Test Loss = 0.982,
Test Accuracy = 0.835
Iteration 95, Train Loss = 0.000, Train Accuracy = 0.852, Test Loss = 0.984,
Test Accuracy = 0.831
Iteration 96, Train Loss = 0.000, Train Accuracy = 0.883, Test Loss = 0.986,
Test Accuracy = 0.840
Iteration 97, Train Loss = 0.000, Train Accuracy = 0.871, Test Loss = 0.986,
Test Accuracy = 0.829
Iteration 98, Train Loss = 0.000, Train Accuracy = 0.901, Test Loss = 0.987,
Test Accuracy = 0.815
Iteration 99, Train Loss = 0.000, Train Accuracy = 0.839, Test Loss = 0.988,
Test Accuracy = 0.825
Iteration 100, Train Loss = 0.000, Train Accuracy = 0.877, Test Loss = 0.990,
Test Accuracy = 0.789


Starting Case 4 with Augment Train=True and Augment Test=True
Iteration 0, Train Loss = 2.283, Train Accuracy = 0.235, Test Loss = 2.284, Test
Accuracy = 0.220
Iteration 1, Train Loss = 0.566, Train Accuracy = 0.781, Test Loss = 0.629, Test
Accuracy = 0.753
Iteration 2, Train Loss = 0.518, Train Accuracy = 0.798, Test Loss = 0.591, Test
Accuracy = 0.786
Iteration 3, Train Loss = 0.450, Train Accuracy = 0.823, Test Loss = 0.528, Test
Accuracy = 0.809
Iteration 4, Train Loss = 0.407, Train Accuracy = 0.834, Test Loss = 0.485, Test
Accuracy = 0.818
Iteration 5, Train Loss = 0.393, Train Accuracy = 0.844, Test Loss = 0.472, Test
Accuracy = 0.821
Iteration 6, Train Loss = 0.384, Train Accuracy = 0.850, Test Loss = 0.470, Test
Accuracy = 0.821
Iteration 7, Train Loss = 0.332, Train Accuracy = 0.866, Test Loss = 0.425, Test
Accuracy = 0.837
Iteration 8, Train Loss = 0.332, Train Accuracy = 0.868, Test Loss = 0.435, Test
Accuracy = 0.840
Iteration 9, Train Loss = 0.323, Train Accuracy = 0.862, Test Loss = 0.440, Test
Accuracy = 0.838
Iteration 10, Train Loss = 0.366, Train Accuracy = 0.867, Test Loss = 0.474,
Test Accuracy = 0.824
Iteration 11, Train Loss = 0.285, Train Accuracy = 0.882, Test Loss = 0.407,
Test Accuracy = 0.860
Iteration 12, Train Loss = 0.336, Train Accuracy = 0.870, Test Loss = 0.467,
```

```
Test Accuracy = 0.840
Iteration 13, Train Loss = 0.283, Train Accuracy = 0.891, Test Loss = 0.407,
Test Accuracy = 0.845
Iteration 14, Train Loss = 0.312, Train Accuracy = 0.873, Test Loss = 0.445,
Test Accuracy = 0.840
Iteration 15, Train Loss = 0.277, Train Accuracy = 0.876, Test Loss = 0.428,
Test Accuracy = 0.849
Iteration 16, Train Loss = 0.298, Train Accuracy = 0.885, Test Loss = 0.434,
Test Accuracy = 0.844
Iteration 17, Train Loss = 0.281, Train Accuracy = 0.890, Test Loss = 0.431,
Test Accuracy = 0.849
Iteration 18, Train Loss = 0.259, Train Accuracy = 0.897, Test Loss = 0.415,
Test Accuracy = 0.856
Iteration 19, Train Loss = 0.271, Train Accuracy = 0.883, Test Loss = 0.431,
Test Accuracy = 0.852
Iteration 20, Train Loss = 0.260, Train Accuracy = 0.902, Test Loss = 0.417,
Test Accuracy = 0.852
Iteration 21, Train Loss = 0.249, Train Accuracy = 0.897, Test Loss = 0.413,
Test Accuracy = 0.859
Iteration 22, Train Loss = 0.276, Train Accuracy = 0.884, Test Loss = 0.449,
Test Accuracy = 0.850
Iteration 23, Train Loss = 0.260, Train Accuracy = 0.897, Test Loss = 0.432,
Test Accuracy = 0.862
Iteration 24, Train Loss = 0.257, Train Accuracy = 0.890, Test Loss = 0.423,
Test Accuracy = 0.846
Iteration 25, Train Loss = 0.253, Train Accuracy = 0.887, Test Loss = 0.436,
Test Accuracy = 0.852
Iteration 26, Train Loss = 0.235, Train Accuracy = 0.892, Test Loss = 0.421,
Test Accuracy = 0.843
Iteration 27, Train Loss = 0.250, Train Accuracy = 0.899, Test Loss = 0.427,
Test Accuracy = 0.857
Iteration 28, Train Loss = 0.234, Train Accuracy = 0.896, Test Loss = 0.423,
Test Accuracy = 0.854
Iteration 29, Train Loss = 0.234, Train Accuracy = 0.910, Test Loss = 0.416,
Test Accuracy = 0.867
Iteration 30, Train Loss = 0.224, Train Accuracy = 0.916, Test Loss = 0.429,
Test Accuracy = 0.864
Iteration 31, Train Loss = 0.229, Train Accuracy = 0.903, Test Loss = 0.416,
Test Accuracy = 0.857
Iteration 32, Train Loss = 0.201, Train Accuracy = 0.919, Test Loss = 0.400,
Test Accuracy = 0.867
Iteration 33, Train Loss = 0.231, Train Accuracy = 0.912, Test Loss = 0.434,
Test Accuracy = 0.855
Iteration 34, Train Loss = 0.237, Train Accuracy = 0.910, Test Loss = 0.444,
Test Accuracy = 0.861
Iteration 35, Train Loss = 0.225, Train Accuracy = 0.902, Test Loss = 0.433,
Test Accuracy = 0.863
Iteration 36, Train Loss = 0.219, Train Accuracy = 0.922, Test Loss = 0.437,
```

Test Accuracy = 0.858
Iteration 37, Train Loss = 0.212, Train Accuracy = 0.916, Test Loss = 0.431,
Test Accuracy = 0.859
Iteration 38, Train Loss = 0.226, Train Accuracy = 0.910, Test Loss = 0.445,
Test Accuracy = 0.861
Iteration 39, Train Loss = 0.204, Train Accuracy = 0.918, Test Loss = 0.411,
Test Accuracy = 0.865
Iteration 40, Train Loss = 0.179, Train Accuracy = 0.921, Test Loss = 0.412,
Test Accuracy = 0.861
Iteration 41, Train Loss = 0.190, Train Accuracy = 0.920, Test Loss = 0.450,
Test Accuracy = 0.865
Iteration 42, Train Loss = 0.216, Train Accuracy = 0.922, Test Loss = 0.435,
Test Accuracy = 0.861
Iteration 43, Train Loss = 0.175, Train Accuracy = 0.929, Test Loss = 0.415,
Test Accuracy = 0.873
Iteration 44, Train Loss = 0.224, Train Accuracy = 0.918, Test Loss = 0.464,
Test Accuracy = 0.845
Iteration 45, Train Loss = 0.195, Train Accuracy = 0.924, Test Loss = 0.426,
Test Accuracy = 0.863
Iteration 46, Train Loss = 0.200, Train Accuracy = 0.909, Test Loss = 0.458,
Test Accuracy = 0.865
Iteration 47, Train Loss = 0.177, Train Accuracy = 0.918, Test Loss = 0.428,
Test Accuracy = 0.861
Iteration 48, Train Loss = 0.197, Train Accuracy = 0.916, Test Loss = 0.472,
Test Accuracy = 0.857
Iteration 49, Train Loss = 0.198, Train Accuracy = 0.925, Test Loss = 0.444,
Test Accuracy = 0.864
Iteration 50, Train Loss = 0.166, Train Accuracy = 0.921, Test Loss = 0.434,
Test Accuracy = 0.878
Iteration 51, Train Loss = 0.186, Train Accuracy = 0.923, Test Loss = 0.451,
Test Accuracy = 0.867
Iteration 52, Train Loss = 0.193, Train Accuracy = 0.924, Test Loss = 0.447,
Test Accuracy = 0.860
Iteration 53, Train Loss = 0.175, Train Accuracy = 0.920, Test Loss = 0.437,
Test Accuracy = 0.874
Iteration 54, Train Loss = 0.197, Train Accuracy = 0.926, Test Loss = 0.449,
Test Accuracy = 0.861
Iteration 55, Train Loss = 0.195, Train Accuracy = 0.923, Test Loss = 0.429,
Test Accuracy = 0.871
Iteration 56, Train Loss = 0.175, Train Accuracy = 0.937, Test Loss = 0.451,
Test Accuracy = 0.869
Iteration 57, Train Loss = 0.180, Train Accuracy = 0.915, Test Loss = 0.440,
Test Accuracy = 0.863
Iteration 58, Train Loss = 0.163, Train Accuracy = 0.933, Test Loss = 0.430,
Test Accuracy = 0.866
Iteration 59, Train Loss = 0.174, Train Accuracy = 0.921, Test Loss = 0.452,
Test Accuracy = 0.853
Iteration 60, Train Loss = 0.169, Train Accuracy = 0.922, Test Loss = 0.432,

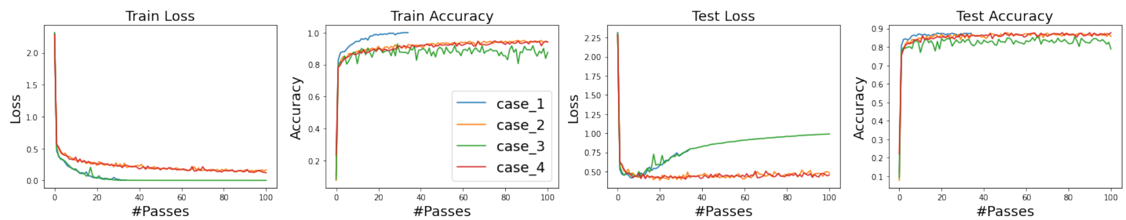
```
Test Accuracy = 0.858
Iteration 61, Train Loss = 0.182, Train Accuracy = 0.928, Test Loss = 0.472,
Test Accuracy = 0.870
Iteration 62, Train Loss = 0.158, Train Accuracy = 0.936, Test Loss = 0.429,
Test Accuracy = 0.872
Iteration 63, Train Loss = 0.158, Train Accuracy = 0.929, Test Loss = 0.451,
Test Accuracy = 0.873
Iteration 64, Train Loss = 0.158, Train Accuracy = 0.932, Test Loss = 0.447,
Test Accuracy = 0.876
Iteration 65, Train Loss = 0.217, Train Accuracy = 0.915, Test Loss = 0.505,
Test Accuracy = 0.858
Iteration 66, Train Loss = 0.146, Train Accuracy = 0.916, Test Loss = 0.418,
Test Accuracy = 0.862
Iteration 67, Train Loss = 0.149, Train Accuracy = 0.946, Test Loss = 0.421,
Test Accuracy = 0.874
Iteration 68, Train Loss = 0.169, Train Accuracy = 0.932, Test Loss = 0.431,
Test Accuracy = 0.866
Iteration 69, Train Loss = 0.155, Train Accuracy = 0.923, Test Loss = 0.443,
Test Accuracy = 0.868
Iteration 70, Train Loss = 0.214, Train Accuracy = 0.925, Test Loss = 0.506,
Test Accuracy = 0.858
Iteration 71, Train Loss = 0.167, Train Accuracy = 0.922, Test Loss = 0.469,
Test Accuracy = 0.865
Iteration 72, Train Loss = 0.149, Train Accuracy = 0.925, Test Loss = 0.466,
Test Accuracy = 0.869
Iteration 73, Train Loss = 0.165, Train Accuracy = 0.930, Test Loss = 0.443,
Test Accuracy = 0.873
Iteration 74, Train Loss = 0.147, Train Accuracy = 0.941, Test Loss = 0.428,
Test Accuracy = 0.871
Iteration 75, Train Loss = 0.170, Train Accuracy = 0.948, Test Loss = 0.447,
Test Accuracy = 0.872
Iteration 76, Train Loss = 0.153, Train Accuracy = 0.940, Test Loss = 0.477,
Test Accuracy = 0.868
Iteration 77, Train Loss = 0.138, Train Accuracy = 0.941, Test Loss = 0.443,
Test Accuracy = 0.878
Iteration 78, Train Loss = 0.146, Train Accuracy = 0.940, Test Loss = 0.437,
Test Accuracy = 0.862
Iteration 79, Train Loss = 0.132, Train Accuracy = 0.933, Test Loss = 0.431,
Test Accuracy = 0.875
Iteration 80, Train Loss = 0.154, Train Accuracy = 0.925, Test Loss = 0.461,
Test Accuracy = 0.865
Iteration 81, Train Loss = 0.150, Train Accuracy = 0.939, Test Loss = 0.441,
Test Accuracy = 0.868
Iteration 82, Train Loss = 0.149, Train Accuracy = 0.928, Test Loss = 0.467,
Test Accuracy = 0.870
Iteration 83, Train Loss = 0.152, Train Accuracy = 0.939, Test Loss = 0.448,
Test Accuracy = 0.865
Iteration 84, Train Loss = 0.166, Train Accuracy = 0.932, Test Loss = 0.473,
```

```
Test Accuracy = 0.854
Iteration 85, Train Loss = 0.136, Train Accuracy = 0.933, Test Loss = 0.436,
Test Accuracy = 0.872
Iteration 86, Train Loss = 0.146, Train Accuracy = 0.947, Test Loss = 0.459,
Test Accuracy = 0.874
Iteration 87, Train Loss = 0.150, Train Accuracy = 0.932, Test Loss = 0.431,
Test Accuracy = 0.861
Iteration 88, Train Loss = 0.155, Train Accuracy = 0.933, Test Loss = 0.480,
Test Accuracy = 0.862
Iteration 89, Train Loss = 0.136, Train Accuracy = 0.938, Test Loss = 0.463,
Test Accuracy = 0.868
Iteration 90, Train Loss = 0.138, Train Accuracy = 0.941, Test Loss = 0.449,
Test Accuracy = 0.878
Iteration 91, Train Loss = 0.171, Train Accuracy = 0.929, Test Loss = 0.471,
Test Accuracy = 0.857
Iteration 92, Train Loss = 0.131, Train Accuracy = 0.944, Test Loss = 0.422,
Test Accuracy = 0.875
Iteration 93, Train Loss = 0.132, Train Accuracy = 0.942, Test Loss = 0.460,
Test Accuracy = 0.868
Iteration 94, Train Loss = 0.124, Train Accuracy = 0.943, Test Loss = 0.437,
Test Accuracy = 0.862
Iteration 95, Train Loss = 0.133, Train Accuracy = 0.939, Test Loss = 0.428,
Test Accuracy = 0.872
Iteration 96, Train Loss = 0.152, Train Accuracy = 0.945, Test Loss = 0.468,
Test Accuracy = 0.870
Iteration 97, Train Loss = 0.158, Train Accuracy = 0.917, Test Loss = 0.477,
Test Accuracy = 0.864
Iteration 98, Train Loss = 0.134, Train Accuracy = 0.938, Test Loss = 0.462,
Test Accuracy = 0.871
Iteration 99, Train Loss = 0.125, Train Accuracy = 0.945, Test Loss = 0.443,
Test Accuracy = 0.870
Iteration 100, Train Loss = 0.120, Train Accuracy = 0.942, Test Loss = 0.456,
Test Accuracy = 0.877
```

[24]:

```
f, ax = plt.subplots(1, 4, figsize=(20, 4))

ax[0].set_title('Train Loss', fontsize=18)
ax[1].set_title('Train Accuracy', fontsize=18)
ax[2].set_title('Test Loss', fontsize=18)
ax[3].set_title('Test Accuracy', fontsize=18)

for j, case in enumerate(cases_list):
    line_label = f'case_{j+1}'
    for i in range(4):
        ax[i].set_xlabel('#Passes', fontsize=18)
```

```
        ax[i].set_ylabel('Loss' if i%2==0 else 'Accuracy', fontsize=18)
        ax[i].plot(list(map(lambda x: x[i], case)), label=line_label)

ax[1].legend(fontsize=18)
plt.tight_layout()
```



[30]:
```
for j, case in enumerate(cases_list):
    print(f'The final test accuracy for case {j+1} is: {case[-1][-1].item()*100:
 ↪.2f}%')
```

```
The final test accuracy for case 1 is: 87.41%
The final test accuracy for case 2 is: 85.70%
The final test accuracy for case 3 is: 78.93%
The final test accuracy for case 4 is: 87.69%
```