Assignment 3

November 13, 2024

EN3160 Assignment 3 on Neural Networks

Instructed by Dr. Ranga Rodrigo

Done by Warren Jayakumar

Repository: Warren-SJ/Image-Processing-Exercises

Contents

Introduction

This assignment is focused on implementing neural networks for image classification. This is done by using: 1. Our own neural network implementation 2. An implementation of LeNet-5 3. An implementation of ResNet-18

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchinfo import summary
import matplotlib.pyplot as plt
import gc
```

[2]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

1 Our own architecture

[3]:
```
transform = transforms.Compose ([ transforms.ToTensor(), transforms.
 ↪Normalize((0.5, 0.5, 0.5) , (0.5, 0.5, 0.5))])
batch_size = 32
trainset = torchvision.datasets.CIFAR10(root= './data', train=True,␣
 ↪download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,␣
 ↪shuffle=True, num_workers=2)
testset = torchvision.datasets.CIFAR10(root= './data', train=False,␣
 ↪download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,␣
 ↪shuffle=False, num_workers=2)
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',␣
 ↪'ship', 'truck')
```

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

1.1 Single Layer

[4]:
```
Din = 3*32*32 # Input size (flattened CIFAR=10 image size)
K = 10 # Output size (number of classes in CIFAR=10)
std = 1e-5
# Initialize weights and biases
w = torch.randn(Din, K, device=device, dtype=torch.float, requires_grad=True) *␣
 ↪std
b = torch.randn(K, device=device, dtype=torch.float, requires_grad=True)
# Hyperparameters
iterations = 20
lr = 2e-6 # Learning rate
lr_decay = 0.9 # Learning rate decay
reg = 0 # Regularization
loss_history = [ ]
```

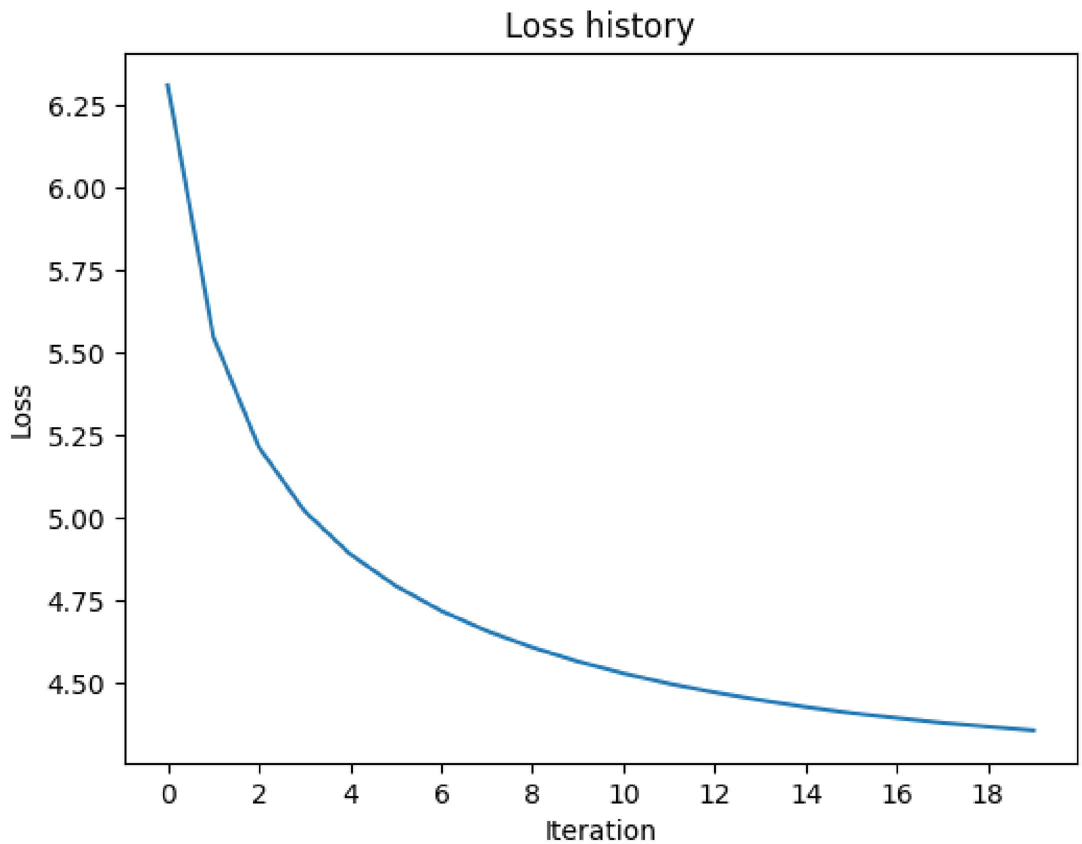
[5]:
```
for t in range(iterations):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # Get inputs and labels
        inputs, labels = data
        Ntr = inputs.shape[0]  # Batch size
        x_train = inputs.view(Ntr, -1).to(device)  # Flatten input to (Ntr, Din)
        y_train_onehot = nn.functional.one_hot(labels, K).float().to(device)  #␣
 ↪Convert labels to one-hot

        # Forward pass
        y_pred = x_train.mm(w) + b  # Output layer activation
```

```
        # Loss calculation (Mean Squared Error with regularization)
        loss = (1/Ntr) * torch.sum((y_pred - y_train_onehot) ** 2) + reg *␣
↪torch.sum(w ** 2)
        running_loss += loss.item()

        # Backpropagation
        dy_pred = (2.0 / Ntr) * (y_pred - y_train_onehot)
        dw = x_train.t().mm(dy_pred) + reg * w
        db = dy_pred.sum(dim=0)

        # Parameter update
        w = w - lr * dw
        b = b - lr * db

    loss_history.append(running_loss / len(trainloader))
    print(f"Epoch {t + 1} / {iterations}, Loss: {running_loss /␣
↪len(trainloader)}")

    # Learning rate decay
    lr *= lr_decay
```

```
Epoch 1 / 20, Loss: 6.309315636916109
Epoch 2 / 20, Loss: 5.549143583898124
Epoch 3 / 20, Loss: 5.214354223878583
Epoch 4 / 20, Loss: 5.0211618167806416
Epoch 5 / 20, Loss: 4.890955400405903
Epoch 6 / 20, Loss: 4.794787223111798
Epoch 7 / 20, Loss: 4.718993186798144
Epoch 8 / 20, Loss: 4.65812322838674
Epoch 9 / 20, Loss: 4.6082750625207645
Epoch 10 / 20, Loss: 4.56568484968355
Epoch 11 / 20, Loss: 4.529754013109116
Epoch 12 / 20, Loss: 4.498623353734812
Epoch 13 / 20, Loss: 4.472161533431373
Epoch 14 / 20, Loss: 4.448784878829009
Epoch 15 / 20, Loss: 4.427916666520229
Epoch 16 / 20, Loss: 4.409813799769621
Epoch 17 / 20, Loss: 4.394613589068978
Epoch 18 / 20, Loss: 4.380091812239956
Epoch 19 / 20, Loss: 4.368443265299879
Epoch 20 / 20, Loss: 4.356881305337028
```

[6]:
```
plt.plot(loss_history)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.xticks(range(0, iterations, 2))
```

```python
plt.title('Loss history')
plt.show()
```



[7]:
```python
def calculate_accuracy(dataloader: torch.utils.data.DataLoader, w: torch.
↪Tensor, b: torch.Tensor) -> float:
    correct = 0
    total = 0
    with torch.no_grad():
        for data in dataloader:
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)
            N = inputs.shape[0]
            x = inputs.view(N, -1)
            y = x.mm(w) + b
            predicted = torch.argmax(y, dim=1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return 100 * correct / total
```

[8]:
```
train_accuracy = calculate_accuracy(trainloader, w, b)
test_accuracy = calculate_accuracy(testloader, w, b)

print(f"Train accuracy: {train_accuracy:.2f}%")
print(f"Test accuracy: {test_accuracy:.2f}%")
```

```
Train accuracy: 16.13%
Test accuracy: 15.91%
```

We see above that the performance is extremely poor. This is because the model has no non-linearity. We will add a non-linearity to the model and see if the performance improves. This is added using a hidden layer with sigmoid activation

[9]:
```
del w, b, x_train, y_train_onehot, y_pred, loss, dy_pred, dw, db
gc.collect()
if torch.cuda.is_available():
    torch.cuda.empty_cache()
```

1.2 Adding Non-linearity

[10]:
```
# This implementation is not efficient and is only for educational purposes.␣
 ↪For real-world applications, use PyTorch's built-in functions and classes.␣
 ↪ThiS may fail
# as memory usage increases with the number of iterations.

Din = 3*32*32 # Input size (flattened CIFAR=10 image size)
K = 10 # Output size (number of classes in CIFAR=10)
std = 1e-5
# Initialize weights and biases
w1 = torch.randn(Din, 100, device=device, requires_grad=True)
b1 = torch.zeros(100, device=device, requires_grad=True)
w2 = torch.randn(100, K, device=device, requires_grad=True)
b2 = torch.zeros(K, device=device, requires_grad=True)
# Hyperparameters
iterations = 10 # Reduced as memory usage increases
lr = 2e-6 # Learning rate
lr_decay = 0.9 # Learning rate decay
reg = 0 # Regularization
loss_history = [ ]
```

[11]:
```
for t in range(iterations):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # Get inputs and labels
        inputs, labels = data
        Ntr = inputs.shape[0]  # Batch size
        x_train = inputs.view(Ntr, -1).to(device)  # Flatten input to (Ntr, Din)
```

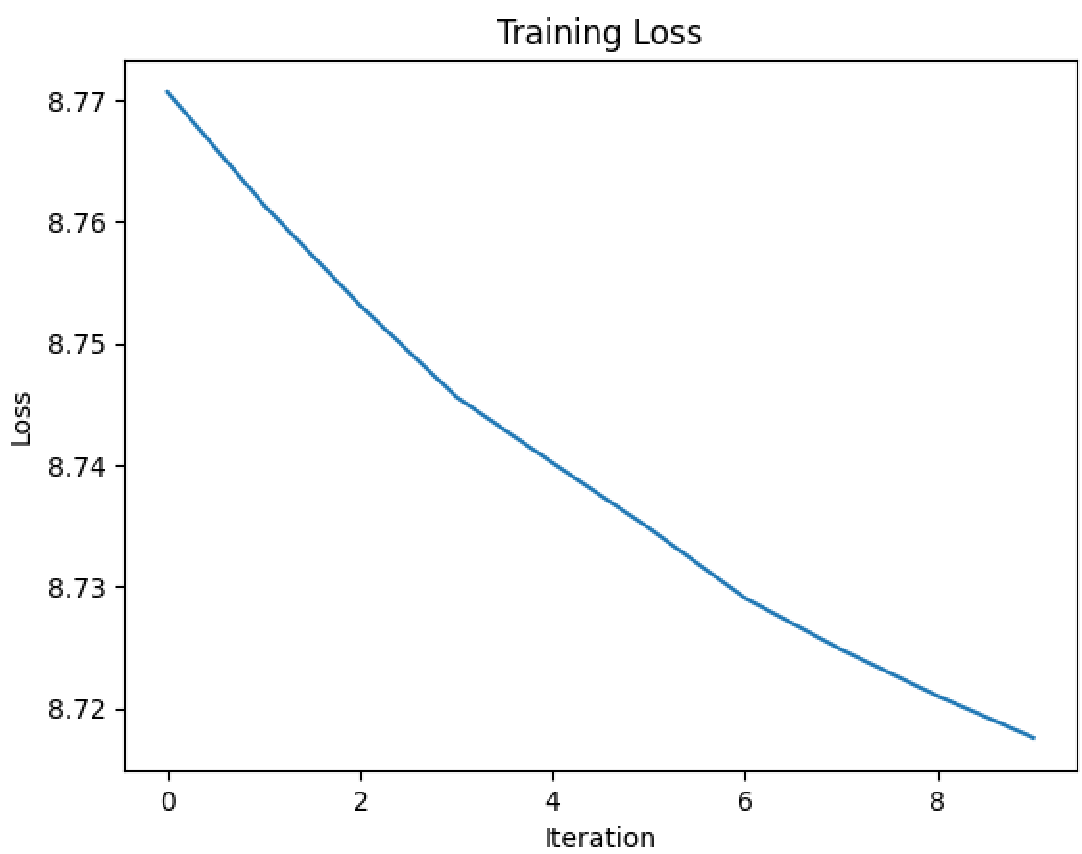
```
        y_train_onehot = nn.functional.one_hot(labels, K).float().to(device)  # One-hot labels

        # Forward pass
        hidden = x_train.mm(w1) + b1
        hidden_activation = torch.sigmoid(hidden)  # Sigmoid activation
        logits = hidden_activation.mm(w2) + b2  # Logits before softmax

        # Compute softmax probabilities
        max_logits = torch.max(logits, dim=1, keepdim=True)[0]
        exp_logits = torch.exp(logits - max_logits)
        probs = exp_logits / torch.sum(exp_logits, dim=1, keepdim=True)

        # Cross-Entropy Loss with L2 regularization
        epsilon = 1e-12  # Small value to prevent log(0)
        log_probs = torch.log(probs + epsilon)
        loss = -torch.sum(y_train_onehot * log_probs) / Ntr
        loss += reg * (torch.sum(w1 ** 2) + torch.sum(w2 ** 2))
        running_loss += loss.item()

        # Backpropagation
        dlogits = (probs - y_train_onehot) / Ntr

        # Gradients for parameters of the second layer
        dw2 = hidden_activation.t().mm(dlogits) + reg * w2
        db2 = dlogits.sum(dim=0)

        # Backpropagate through ReLU activation
        dhidden_activation = dlogits.mm(w2.t())
        dhidden = dhidden_activation * hidden_activation * (1 - hidden_activation)  # Derivative of sigmoid

        # Gradients for parameters of the first layer
        dw1 = x_train.t().mm(dhidden) + reg * w1
        db1 = dhidden.sum(dim=0)

        # Parameter updates
        w2 = w2 - lr * dw2
        b2 = b2 - lr * db2
        w1 = w1 - lr * dw1
        b1 = b1 - lr * db1

    loss_history.append(running_loss / len(trainloader))
    print(f"Epoch {t+1} / {iterations}, Loss: {running_loss / len(trainloader)}")

    # Learning rate decay
```

```
    lr *= lr_decay
```

```
Epoch 1 / 10, Loss: 8.77062671060983
Epoch 2 / 10, Loss: 8.7613635920441
Epoch 3 / 10, Loss: 8.753118033601318
Epoch 4 / 10, Loss: 8.745599119463412
Epoch 5 / 10, Loss: 8.740169318913651
Epoch 6 / 10, Loss: 8.734824682761673
Epoch 7 / 10, Loss: 8.729062423291149
Epoch 8 / 10, Loss: 8.724829464178397
Epoch 9 / 10, Loss: 8.721024848525522
Epoch 10 / 10, Loss: 8.717586632043371
```

It is observed that the loss values decrease on each iteration

[12]:
```
plt.plot(loss_history)
plt.xlabel('Iteration')
plt.xticks(range(0, iterations, 2))
plt.ylabel('Loss')
plt.title('Training Loss')
plt.show()
```



[13]:
```
def calculate_accuracy(dataloader: torch.utils.data.DataLoader, w1: torch.
↪Tensor, b1: torch.Tensor, w2: torch.Tensor, b2: torch.Tensor) -> float:
    correct = 0
    total = 0
    with torch.no_grad():
        for data in dataloader:
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)
            N = inputs.shape[0]
            x = inputs.view(N, -1)
            hidden = torch.sigmoid(x.mm(w1) + b1)
            y = hidden.mm(w2) + b2
            predicted = torch.argmax(y, dim=1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return 100 * correct / total
```

[14]:
```
train_accuracy = calculate_accuracy(trainloader, w1, b1, w2, b2)
test_accuracy = calculate_accuracy(testloader, w1, b1, w2, b2)

print(f"Train accuracy: {train_accuracy:.2f}%")
print(f"Test accuracy: {test_accuracy:.2f}%")
```

```
Train accuracy: 10.71%
Test accuracy: 10.24%
```

[15]:
```
del w1, b1, w2, b2, x_train, y_train_onehot, hidden, hidden_activation, logits,␣
↪probs, log_probs, loss, dlogits, dw2, db2, dhidden_activation, dhidden, dw1,␣
↪db1
gc.collect()
if torch.cuda.is_available():
    torch.cuda.empty_cache()
```

1.3 A more efficient implementation

[16]:
```
Din = 3*32*32 # Input size (flattened CIFAR=10 image size)
K = 10 # Output size (number of classes in CIFAR=10)
lr = 1e-3 # Learning rate
reg = 1e-5 # Regularization strength
```

[17]:
```
class NeuralNetwork(nn.Module):
    def __init__(self, Din, H, Dout):
        super(NeuralNetwork, self).__init__()
        self.linear1 = nn.Linear(Din, H)
        self.linear2 = nn.Linear(H, Dout)

    def forward(self, x):
```

```python
        x = torch.flatten(x, 1)
        x = torch.sigmoid(self.linear1(x))
        x = self.linear2(x)
        return x
```

We will define a function for training and testing the model

[18]:
```python
def train(model:nn.Module,
          trainloader:torch.utils.data.DataLoader,
          testloader:torch.utils.data.DataLoader,
          iterations:int,
          optimizer:torch.optim.Optimizer,
          loss_fn:torch.nn.Module,
          device: torch.device) -> tuple:
    train_accuracy_hist = [ ]
    test_accuracy_hist = [ ]
    train_loss_hist = [ ]
    test_loss_hist = [ ]
    for t in range(iterations):
        model.train()
        accuracy = 0
        running_loss = 0.0
        for _, data in enumerate(trainloader, 0):
            inputs, labels = data
            x_train, y_train = inputs.to(device), labels.to(device)
            y_pred = model(x_train)
            loss_val = loss_fn(y_pred, y_train)
            running_loss += loss_val.item()
            optimizer.zero_grad()
            loss_val.backward()
            optimizer.step()
            _, predicted = torch.max(y_pred, 1)
            accuracy += (predicted == y_train).sum().item()
        train_accuracy_hist.append(accuracy / len(trainloader.dataset))
        train_loss_hist.append(running_loss / len(trainloader))
        model.eval()
        with torch.inference_mode():
            accuracy = 0
            running_loss = 0.0
            for i, data in enumerate(testloader, 0):
                inputs, labels = data
                x_test, y_test = inputs.to(device), labels.to(device)
                y_pred = model(x_test)
                loss_val = loss_fn(y_pred, y_test)
                running_loss += loss_val.item()
                _, predicted = torch.max(y_pred, 1)
                accuracy += (predicted == y_test).sum().item()
```

```
            test_accuracy_hist.append(accuracy / len(testloader.dataset))
            test_loss_hist.append(running_loss / len(testloader))
        print(f"Epoch {t + 1} / {iterations}, Train Loss:␣
↪{train_loss_hist[-1]}, Test Loss: {test_loss_hist[-1]}, Train Accuracy:␣
↪{train_accuracy_hist[-1]}, Test Accuracy: {test_accuracy_hist[-1]}")
    return train_accuracy_hist, test_accuracy_hist, train_loss_hist,␣
↪test_loss_hist
```

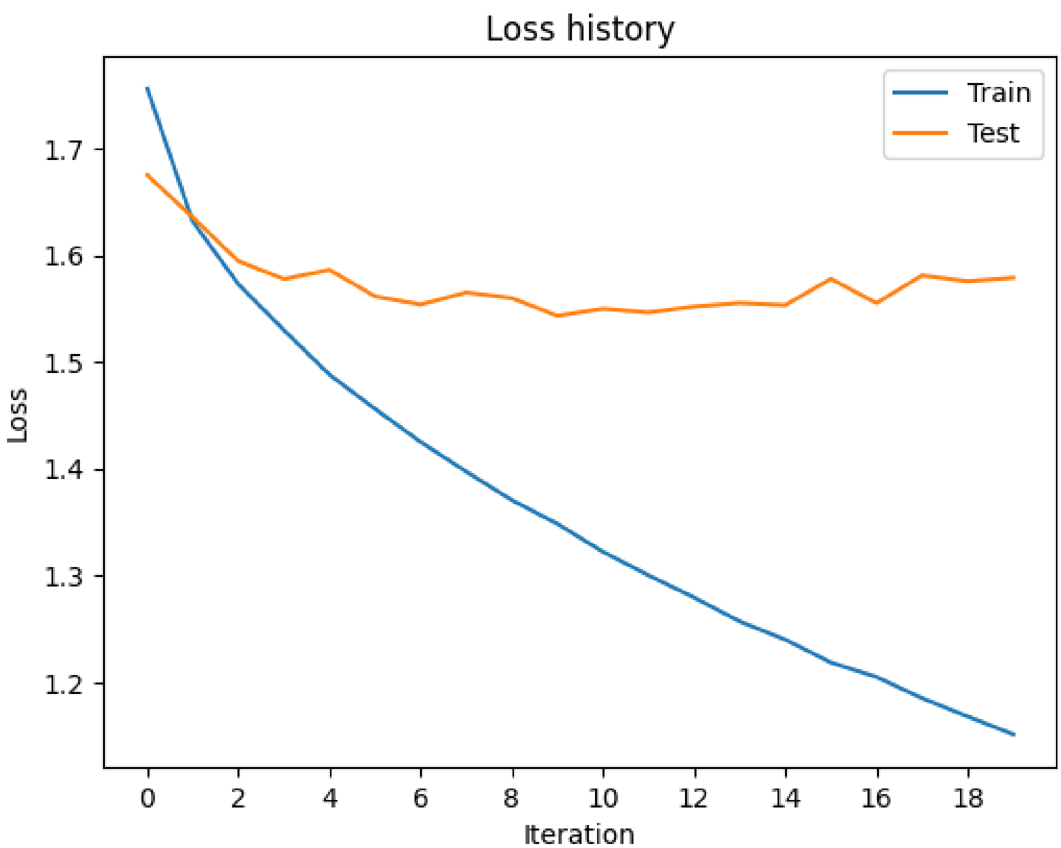
[19]:
```
model = NeuralNetwork(Din, 100, K).to(device)
loss = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=reg)
iterations = 20
```

[20]:
```
train_accuracy_hist, test_accuracy_hist, train_loss_hist, test_loss_hist =␣
↪train(model, trainloader, testloader, iterations, optimizer, loss, device)
```

```
Epoch 1 / 20, Train Loss: 1.756354550177366, Test Loss: 1.675218096556374, Train
Accuracy: 0.3876, Test Accuracy: 0.4225
Epoch 2 / 20, Train Loss: 1.6315957853142755, Test Loss: 1.6355849732987036,
Train Accuracy: 0.43504, Test Accuracy: 0.4294
Epoch 3 / 20, Train Loss: 1.5729634079960624, Test Loss: 1.5945421346841149,
Train Accuracy: 0.45582, Test Accuracy: 0.4499
Epoch 4 / 20, Train Loss: 1.5299214721488708, Test Loss: 1.577893341692111,
Train Accuracy: 0.47008, Test Accuracy: 0.4528
Epoch 5 / 20, Train Loss: 1.4881449486686111, Test Loss: 1.5864154199441782,
Train Accuracy: 0.48602, Test Accuracy: 0.4497
Epoch 6 / 20, Train Loss: 1.4560639447915729, Test Loss: 1.561577286583166,
Train Accuracy: 0.49772, Test Accuracy: 0.4648
Epoch 7 / 20, Train Loss: 1.425123206713378, Test Loss: 1.5540121016791835,
Train Accuracy: 0.50922, Test Accuracy: 0.4663
Epoch 8 / 20, Train Loss: 1.3972938078683839, Test Loss: 1.5651529760787282,
Train Accuracy: 0.51956, Test Accuracy: 0.4651
Epoch 9 / 20, Train Loss: 1.3705994560027535, Test Loss: 1.5600004447534823,
Train Accuracy: 0.52592, Test Accuracy: 0.4632
Epoch 10 / 20, Train Loss: 1.3484133568926644, Test Loss: 1.5433793650648464,
Train Accuracy: 0.53488, Test Accuracy: 0.4718
Epoch 11 / 20, Train Loss: 1.3221722872914676, Test Loss: 1.5499908836504903,
Train Accuracy: 0.54552, Test Accuracy: 0.4662
Epoch 12 / 20, Train Loss: 1.3001274806295384, Test Loss: 1.5465527403468904,
Train Accuracy: 0.5526, Test Accuracy: 0.4681
Epoch 13 / 20, Train Loss: 1.2793257157160391, Test Loss: 1.5520266684861228,
Train Accuracy: 0.5575, Test Accuracy: 0.4706
Epoch 14 / 20, Train Loss: 1.2569441592472148, Test Loss: 1.5554008283935035,
Train Accuracy: 0.56858, Test Accuracy: 0.4734
Epoch 15 / 20, Train Loss: 1.2398779309108634, Test Loss: 1.5534430233815226,
Train Accuracy: 0.57252, Test Accuracy: 0.4746
Epoch 16 / 20, Train Loss: 1.2184532880401733, Test Loss: 1.5779320538615267,
Train Accuracy: 0.58032, Test Accuracy: 0.4626
```

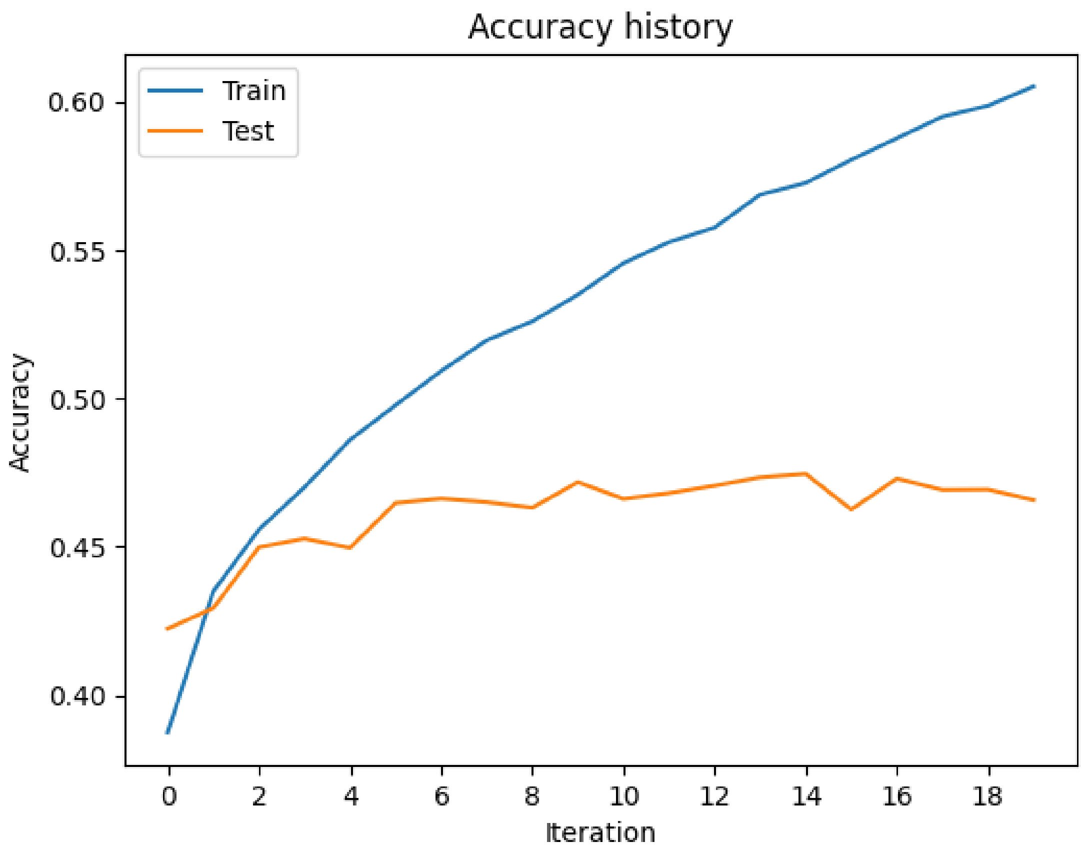
```
Epoch 17 / 20, Train Loss: 1.2049031911259345, Test Loss: 1.555306049962394,
Train Accuracy: 0.58754, Test Accuracy: 0.473
Epoch 18 / 20, Train Loss: 1.1850365490693735, Test Loss: 1.5812077326134752,
Train Accuracy: 0.59484, Test Accuracy: 0.4692
Epoch 19 / 20, Train Loss: 1.167856049667317, Test Loss: 1.5757547550308058,
Train Accuracy: 0.59852, Test Accuracy: 0.4693
Epoch 20 / 20, Train Loss: 1.1511781966541337, Test Loss: 1.578999431559834,
Train Accuracy: 0.60498, Test Accuracy: 0.4658
```

[21]:
```python
plt.plot(train_loss_hist, label='Train')
plt.plot(test_loss_hist, label='Test')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.xticks(range(0, iterations, 2))
plt.title('Loss history')
plt.legend()
plt.show()
```



[22]:
```python
plt.plot(train_accuracy_hist, label='Train')
plt.plot(test_accuracy_hist, label='Test')
```

```
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.xticks(range(0, iterations, 2))
plt.title('Accuracy history')
plt.legend()
plt.show()
```



[23]:
```
def calculate_accuracy(model: nn.Module, dataloader: torch.utils.data.
    DataLoader) -> float:
    correct = 0
    total = 0
    with torch.no_grad():
        for data in dataloader:
            inputs, labels = data
            x, y = inputs.to(device), labels.to(device)
            outputs = model(x)
            _, predicted = torch.max(outputs, 1)
            total += y.size(0)
            correct += (predicted == y).sum().item()
```

```
    return 100 * correct / total
```

[24]:
```
train_accuracy = calculate_accuracy(model, trainloader)
test_accuracy = calculate_accuracy(model, testloader)

print(f"Train accuracy: {train_accuracy:.2f}%")
print(f"Test accuracy: {test_accuracy:.2f}%")
```

```
Train accuracy: 62.63%
Test accuracy: 46.58%
```

We see that the accuracy is still very low as was in our custom implementation As at the time of writing, according to paperswithcode.com, the best accuracy on CIFAR-10 is 99.5%. This is achieved by a model called ViT-H/14 which is a vision transformer. Another thing to note is that the model is beginning to overfit after just 3 epochs. This is because the model is too simple and is not able to learn the complex patterns in the data.

[25]:
```
del model, trainloader, testloader, train_accuracy_hist, test_accuracy_hist,␣
 ↪train_loss_hist, test_loss_hist
gc.collect()
if torch.cuda.is_available():
    torch.cuda.empty_cache()
```

2 LeNet-5

Here we will be implementint LeNet-5 architecture for MNIST dataset.

[26]:
```
batch_size = 32
```

[27]:
```
trainset_mnist = torchvision.datasets.MNIST(root='./data', train=True,␣
 ↪download=True, transform=transforms.ToTensor())
trainloader_mnist = torch.utils.data.DataLoader(trainset_mnist,␣
 ↪batch_size=batch_size, shuffle=True)
testset_mnist = torchvision.datasets.MNIST(root='./data', train=False,␣
 ↪download=True, transform=transforms.ToTensor())
testloader_mnist = torch.utils.data.DataLoader(testset_mnist,␣
 ↪batch_size=batch_size, shuffle=False)
classes = tuple(str(i) for i in range(10))
```

Architecture

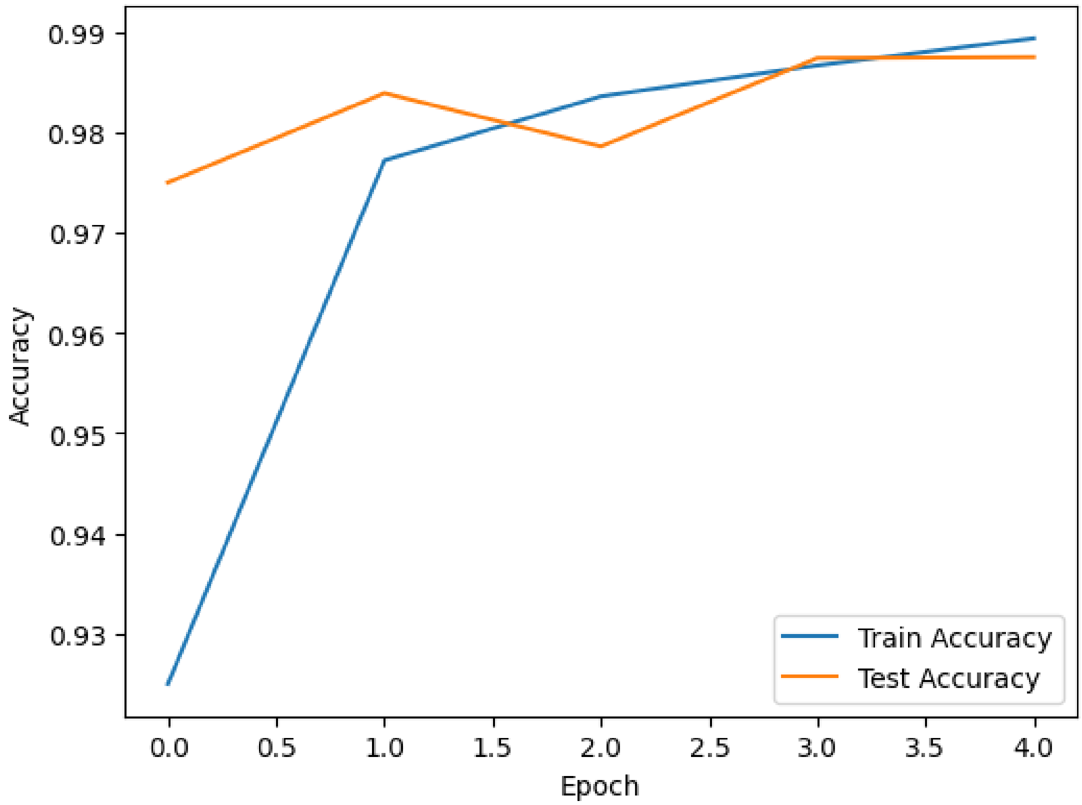
[28]:
```
class LeNet(nn.Module):
    def __init__(self, input_size, input_channels, output_size):
        super(LeNet, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(input_channels, 6, 5),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(6, 16, 5),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )

        conv_output_size = ((input_size - 4) // 2 - 4) // 2
        self.classifier = nn.Sequential(
            nn.Linear(16 * conv_output_size * conv_output_size, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, output_size)
        )

    def forward(self, x):
        y = self.conv1(x)
        y = self.conv2(y)
        y = y.view(y.size(0), -1)
        y = self.classifier(y)
        return y
```

[29]:
```
lenet_model = LeNet(input_size = 28, input_channels = 1, output_size = 10).
 ↪to(device)
loss = nn.CrossEntropyLoss()
optimizer = optim.Adam(lenet_model.parameters(), lr=0.001)
iterations = 5 # Sufficient since MNIST is a simple dataset
```

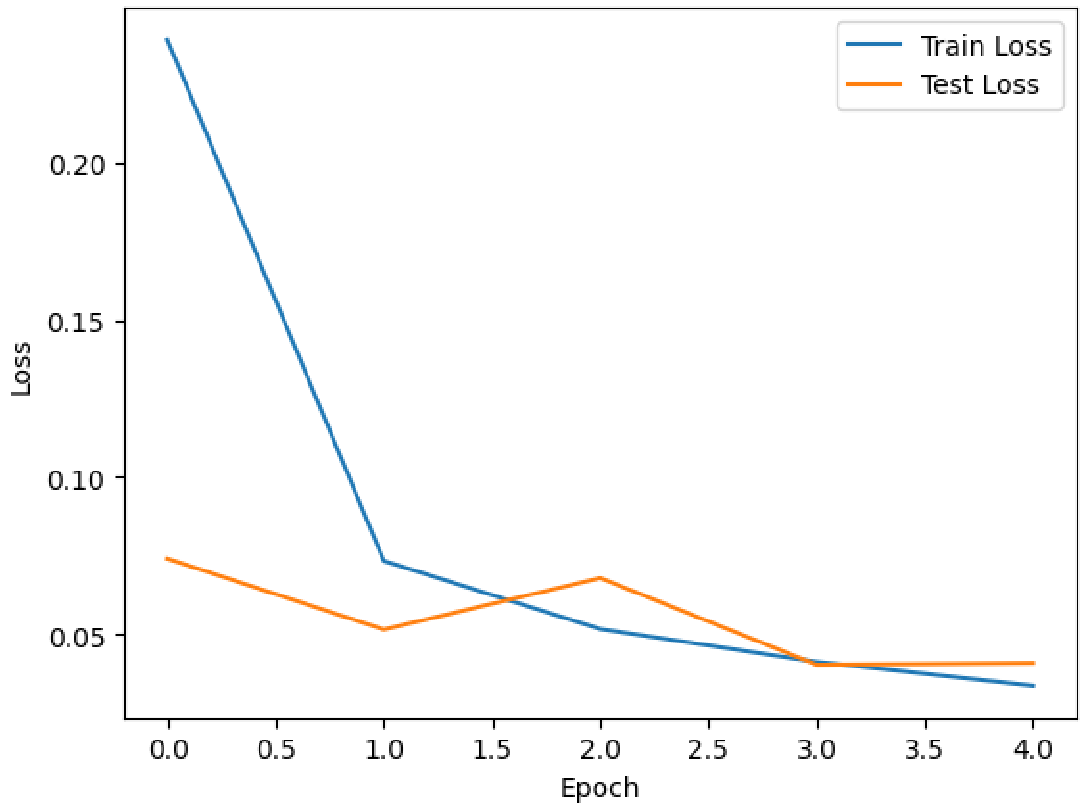
[30]:
```
train_accuracy_hist, test_accuracy_hist, train_loss_hist, test_loss_hist =␣
 ↪train(lenet_model, trainloader_mnist, testloader_mnist, iterations,␣
 ↪optimizer, loss, device)
```

```
Epoch 1 / 5, Train Loss: 0.23917823472023012, Test Loss: 0.07399401403372191,
Train Accuracy: 0.9249833333333334, Test Accuracy: 0.975
Epoch 2 / 5, Train Loss: 0.0732845237663947, Test Loss: 0.051476614563517605,
Train Accuracy: 0.9772, Test Accuracy: 0.9839
Epoch 3 / 5, Train Loss: 0.051611536767209566, Test Loss: 0.06785203708740607,
Train Accuracy: 0.9835833333333334, Test Accuracy: 0.9786
Epoch 4 / 5, Train Loss: 0.0411661645629288, Test Loss: 0.04024723509087889,
Train Accuracy: 0.9866833333333334, Test Accuracy: 0.9874
Epoch 5 / 5, Train Loss: 0.03360636993950078, Test Loss: 0.04080141513728717,
Train Accuracy: 0.9893833333333333, Test Accuracy: 0.9875
```

[31]:
```
plt.plot(train_accuracy_hist, label='Train Accuracy')
plt.plot(test_accuracy_hist, label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[32]:
```
plt.plot(train_loss_hist, label='Train Loss')
plt.plot(test_loss_hist, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[33]:
```
train_accuracy = calculate_accuracy(lenet_model, trainloader_mnist)
test_accuracy = calculate_accuracy(lenet_model, testloader_mnist)

print(f"Train accuracy: {train_accuracy:.2f}%")
print(f"Test accuracy: {test_accuracy:.2f}%")
```

```
Train accuracy: 99.12%
Test accuracy: 98.75%
```

Observing the plots of loss and accuracy, we can see that the model is performing well. As expected, the train and test losses are decreasing and the train and test accuracies are increasing with each epoch. After 5 epochs, the model was able to achieve a test accuracy of 98.75%. This is easy to achieve as the MNIST dataset is simple and LeNet-5 is a good architecture for this dataset. 5 epochs were used since the model began to overfit after this point.

3 Implementing ResNet-18

In this section, we will implement ResNet-18 architecture for classifying the hymenoptera dataset consiting of images of ants and bees. In this first section, we will be finetuning the netwrok where we will be using a pre-trained model and retraining it on the hymenoptera dataset. In the second section, we will be using the network as a feature extractor where we freeze the weights of the network and only train the final classification layer.

3.1 Finetuning the network

[34]:
```
resnet_model = torchvision.models.resnet18(weights = 'IMAGENET1K_V1').
↪to(device) # These are the default weights
batch_size = 32
data_folder = './data/hymenoptera_data'
train_transforms = transforms.Compose([transforms.RandomResizedCrop(224),␣
↪transforms.RandomHorizontalFlip(), transforms.ToTensor(), transforms.
↪Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])
trainset_hymenoptera = torchvision.datasets.ImageFolder(root=f'{data_folder}/
↪train', transform=train_transforms)
trainloader_hymenoptera = torch.utils.data.DataLoader(trainset_hymenoptera,␣
↪batch_size=batch_size, shuffle=True)
test_transforms = transforms.Compose([transforms.Resize(256), transforms.
↪CenterCrop(224), transforms.ToTensor(), transforms.Normalize([0.485, 0.456,␣
↪0.406], [0.229, 0.224, 0.225])])
testset_hymenoptera = torchvision.datasets.ImageFolder(root=f'{data_folder}/
↪val', transform=test_transforms)
testloader_hymenoptera = torch.utils.data.DataLoader(testset_hymenoptera,␣
↪batch_size=batch_size, shuffle=False)
classes_hymenoptera = trainset_hymenoptera.classes
```

[35]:
```
print(summary(resnet_model, input_size=(batch_size, 3, 224, 224),␣
↪col_names=["input_size", "output_size", "num_params", "trainable"]))
```

```
=====================================================================================
==============================================================
Layer (type:depth-idx)                   Input Shape               Output Shape
Param #                   Trainable
=====================================================================================
==============================================================
ResNet                                   [32, 3, 224, 224]         [32, 1000]
--                        True
 Conv2d: 1-1                            [32, 3, 224, 224]         [32, 64, 112,
112]        9,408                     True
 BatchNorm2d: 1-2                       [32, 64, 112, 112]        [32, 64, 112,
112]        128                       True
 ReLU: 1-3                              [32, 64, 112, 112]        [32, 64, 112,
112]        --                        --
 MaxPool2d: 1-4                         [32, 64, 112, 112]        [32, 64, 56,
```

```
56]           --                          --
 Sequential: 1-5                           [32, 64, 56, 56]          [32, 64, 56,
56]           --                          True
     BasicBlock: 2-1                       [32, 64, 56, 56]          [32, 64, 56,
56]           --                          True
         Conv2d: 3-1                      [32, 64, 56, 56]          [32, 64, 56,
56]            36,864                     True
         BatchNorm2d: 3-2                 [32, 64, 56, 56]          [32, 64, 56,
56]           128                         True
         ReLU: 3-3                        [32, 64, 56, 56]          [32, 64, 56,
56]           --                          --
         Conv2d: 3-4                      [32, 64, 56, 56]          [32, 64, 56,
56]            36,864                     True
         BatchNorm2d: 3-5                 [32, 64, 56, 56]          [32, 64, 56,
56]           128                         True
         ReLU: 3-6                        [32, 64, 56, 56]          [32, 64, 56,
56]           --                          --
     BasicBlock: 2-2                       [32, 64, 56, 56]          [32, 64, 56,
56]           --                          True
         Conv2d: 3-7                      [32, 64, 56, 56]          [32, 64, 56,
56]            36,864                     True
         BatchNorm2d: 3-8                 [32, 64, 56, 56]          [32, 64, 56,
56]           128                         True
         ReLU: 3-9                        [32, 64, 56, 56]          [32, 64, 56,
56]           --                          --
         Conv2d: 3-10                     [32, 64, 56, 56]          [32, 64, 56,
56]            36,864                     True
         BatchNorm2d: 3-11                [32, 64, 56, 56]          [32, 64, 56,
56]           128                         True
         ReLU: 3-12                       [32, 64, 56, 56]          [32, 64, 56,
56]           --                          --
 Sequential: 1-6                           [32, 64, 56, 56]          [32, 128, 28,
28]          --                          True
     BasicBlock: 2-3                       [32, 64, 56, 56]          [32, 128, 28,
28]          --                          True
         Conv2d: 3-13                      [32, 64, 56, 56]          [32, 128, 28,
28]           73,728                     True
         BatchNorm2d: 3-14                 [32, 128, 28, 28]         [32, 128, 28,
28]           256                        True
         ReLU: 3-15                        [32, 128, 28, 28]         [32, 128, 28,
28]          --                          --
         Conv2d: 3-16                      [32, 128, 28, 28]         [32, 128, 28,
28]           147,456                    True
         BatchNorm2d: 3-17                 [32, 128, 28, 28]         [32, 128, 28,
28]           256                        True
         Sequential: 3-18                  [32, 64, 56, 56]          [32, 128, 28,
28]           8,448                      True
         ReLU: 3-19                        [32, 128, 28, 28]         [32, 128, 28,
```

```
28]         --                          --
     BasicBlock: 2-4                     [32, 128, 28, 28]         [32, 128, 28,
28]         --                          True
         Conv2d: 3-20                    [32, 128, 28, 28]         [32, 128, 28,
28]          147,456                    True
         BatchNorm2d: 3-21               [32, 128, 28, 28]         [32, 128, 28,
28]          256                        True
         ReLU: 3-22                      [32, 128, 28, 28]         [32, 128, 28,
28]         --                          --
         Conv2d: 3-23                    [32, 128, 28, 28]         [32, 128, 28,
28]          147,456                    True
         BatchNorm2d: 3-24               [32, 128, 28, 28]         [32, 128, 28,
28]          256                        True
         ReLU: 3-25                      [32, 128, 28, 28]         [32, 128, 28,
28]         --                          --
 Sequential: 1-7                          [32, 128, 28, 28]         [32, 256, 14,
14]         --                          True
     BasicBlock: 2-5                     [32, 128, 28, 28]         [32, 256, 14,
14]         --                          True
         Conv2d: 3-26                    [32, 128, 28, 28]         [32, 256, 14,
14]          294,912                    True
         BatchNorm2d: 3-27               [32, 256, 14, 14]         [32, 256, 14,
14]          512                        True
         ReLU: 3-28                      [32, 256, 14, 14]         [32, 256, 14,
14]         --                          --
         Conv2d: 3-29                    [32, 256, 14, 14]         [32, 256, 14,
14]          589,824                    True
         BatchNorm2d: 3-30               [32, 256, 14, 14]         [32, 256, 14,
14]          512                        True
         Sequential: 3-31                [32, 128, 28, 28]         [32, 256, 14,
14]          33,280                     True
         ReLU: 3-32                      [32, 256, 14, 14]         [32, 256, 14,
14]         --                          --
     BasicBlock: 2-6                     [32, 256, 14, 14]         [32, 256, 14,
14]         --                          True
         Conv2d: 3-33                    [32, 256, 14, 14]         [32, 256, 14,
14]          589,824                    True
         BatchNorm2d: 3-34               [32, 256, 14, 14]         [32, 256, 14,
14]          512                        True
         ReLU: 3-35                      [32, 256, 14, 14]         [32, 256, 14,
14]         --                          --
         Conv2d: 3-36                    [32, 256, 14, 14]         [32, 256, 14,
14]          589,824                    True
         BatchNorm2d: 3-37               [32, 256, 14, 14]         [32, 256, 14,
14]          512                        True
         ReLU: 3-38                      [32, 256, 14, 14]         [32, 256, 14,
14]         --                          --
 Sequential: 1-8                          [32, 256, 14, 14]         [32, 512, 7,
```

```
7]           --                          True
     BasicBlock: 2-7                      [32, 256, 14, 14]         [32, 512, 7,
7]           --                          True
         Conv2d: 3-39                    [32, 256, 14, 14]         [32, 512, 7,
7]           1,179,648                   True
         BatchNorm2d: 3-40               [32, 512, 7, 7]           [32, 512, 7,
7]           1,024                       True
         ReLU: 3-41                      [32, 512, 7, 7]           [32, 512, 7,
7]           --                          --
         Conv2d: 3-42                    [32, 512, 7, 7]           [32, 512, 7,
7]           2,359,296                   True
         BatchNorm2d: 3-43               [32, 512, 7, 7]           [32, 512, 7,
7]           1,024                       True
         Sequential: 3-44                [32, 256, 14, 14]         [32, 512, 7,
7]           132,096                     True
         ReLU: 3-45                      [32, 512, 7, 7]           [32, 512, 7,
7]           --                          --
     BasicBlock: 2-8                      [32, 512, 7, 7]           [32, 512, 7,
7]           --                          True
         Conv2d: 3-46                    [32, 512, 7, 7]           [32, 512, 7,
7]           2,359,296                   True
         BatchNorm2d: 3-47               [32, 512, 7, 7]           [32, 512, 7,
7]           1,024                       True
         ReLU: 3-48                      [32, 512, 7, 7]           [32, 512, 7,
7]           --                          --
         Conv2d: 3-49                    [32, 512, 7, 7]           [32, 512, 7,
7]           2,359,296                   True
         BatchNorm2d: 3-50               [32, 512, 7, 7]           [32, 512, 7,
7]           1,024                       True
         ReLU: 3-51                      [32, 512, 7, 7]           [32, 512, 7,
7]           --                          --
 AdaptiveAvgPool2d: 1-9                   [32, 512, 7, 7]           [32, 512, 1,
1]           --                          --
 Linear: 1-10                             [32, 512]                 [32, 1000]
513,000                  True
================================================================================
============================================================
Total params: 11,689,512
Trainable params: 11,689,512
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 58.05
================================================================================
============================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 1271.92
Params size (MB): 46.76
Estimated Total Size (MB): 1337.94
================================================================================
```

```
================================================================
```

[36]:
```python
resnet_model.fc = nn.Linear(512, len(classes_hymenoptera)).to(device)
```

[37]:
```python
print(summary(resnet_model, input_size=(batch_size, 3, 224, 224),␣
↪col_names=["input_size", "output_size", "num_params", "trainable"]))
```

```
================================================================================
================================================================
Layer (type:depth-idx)                   Input Shape              Output Shape
Param #                    Trainable
================================================================================
================================================================
ResNet                                   [32, 3, 224, 224]        [32, 2]
--                         True
 Conv2d: 1-1                            [32, 3, 224, 224]        [32, 64, 112,
112]          9,408                     True
 BatchNorm2d: 1-2                       [32, 64, 112, 112]       [32, 64, 112,
112]          128                       True
 ReLU: 1-3                              [32, 64, 112, 112]       [32, 64, 112,
112]          --                        --
 MaxPool2d: 1-4                         [32, 64, 112, 112]       [32, 64, 56,
56]           --                        --
 Sequential: 1-5                        [32, 64, 56, 56]         [32, 64, 56,
56]           --                        True
      BasicBlock: 2-1                   [32, 64, 56, 56]         [32, 64, 56,
56]           --                        True
           Conv2d: 3-1                  [32, 64, 56, 56]         [32, 64, 56,
56]           36,864                    True
           BatchNorm2d: 3-2             [32, 64, 56, 56]         [32, 64, 56,
56]           128                       True
           ReLU: 3-3                    [32, 64, 56, 56]         [32, 64, 56,
56]           --                        --
           Conv2d: 3-4                  [32, 64, 56, 56]         [32, 64, 56,
56]           36,864                    True
           BatchNorm2d: 3-5             [32, 64, 56, 56]         [32, 64, 56,
56]           128                       True
           ReLU: 3-6                    [32, 64, 56, 56]         [32, 64, 56,
56]           --                        --
      BasicBlock: 2-2                   [32, 64, 56, 56]         [32, 64, 56,
56]           --                        True
           Conv2d: 3-7                  [32, 64, 56, 56]         [32, 64, 56,
56]           36,864                    True
           BatchNorm2d: 3-8             [32, 64, 56, 56]         [32, 64, 56,
56]           128                       True
           ReLU: 3-9                    [32, 64, 56, 56]         [32, 64, 56,
56]           --                        --
           Conv2d: 3-10                 [32, 64, 56, 56]         [32, 64, 56,
```

```
56]          36,864                    True
         BatchNorm2d: 3-11              [32, 64, 56, 56]          [32, 64, 56,
56]          128                       True
         ReLU: 3-12                     [32, 64, 56, 56]          [32, 64, 56,
56]          --                        --
 Sequential: 1-6                         [32, 64, 56, 56]          [32, 128, 28,
28]         --                        True
     BasicBlock: 2-3                    [32, 64, 56, 56]          [32, 128, 28,
28]         --                        True
         Conv2d: 3-13                   [32, 64, 56, 56]          [32, 128, 28,
28]         73,728                    True
         BatchNorm2d: 3-14              [32, 128, 28, 28]         [32, 128, 28,
28]         256                       True
         ReLU: 3-15                     [32, 128, 28, 28]         [32, 128, 28,
28]         --                        --
         Conv2d: 3-16                   [32, 128, 28, 28]         [32, 128, 28,
28]         147,456                   True
         BatchNorm2d: 3-17              [32, 128, 28, 28]         [32, 128, 28,
28]         256                       True
         Sequential: 3-18               [32, 64, 56, 56]          [32, 128, 28,
28]         8,448                     True
         ReLU: 3-19                     [32, 128, 28, 28]         [32, 128, 28,
28]         --                        --
     BasicBlock: 2-4                    [32, 128, 28, 28]         [32, 128, 28,
28]         --                        True
         Conv2d: 3-20                   [32, 128, 28, 28]         [32, 128, 28,
28]         147,456                   True
         BatchNorm2d: 3-21              [32, 128, 28, 28]         [32, 128, 28,
28]         256                       True
         ReLU: 3-22                     [32, 128, 28, 28]         [32, 128, 28,
28]         --                        --
         Conv2d: 3-23                   [32, 128, 28, 28]         [32, 128, 28,
28]         147,456                   True
         BatchNorm2d: 3-24              [32, 128, 28, 28]         [32, 128, 28,
28]         256                       True
         ReLU: 3-25                     [32, 128, 28, 28]         [32, 128, 28,
28]         --                        --
 Sequential: 1-7                         [32, 128, 28, 28]         [32, 256, 14,
14]         --                        True
     BasicBlock: 2-5                    [32, 128, 28, 28]         [32, 256, 14,
14]         --                        True
         Conv2d: 3-26                   [32, 128, 28, 28]         [32, 256, 14,
14]         294,912                   True
         BatchNorm2d: 3-27              [32, 256, 14, 14]         [32, 256, 14,
14]         512                       True
         ReLU: 3-28                     [32, 256, 14, 14]         [32, 256, 14,
14]         --                        --
         Conv2d: 3-29                   [32, 256, 14, 14]         [32, 256, 14,
```

```
14]          589,824                       True
         BatchNorm2d: 3-30              [32, 256, 14, 14]         [32, 256, 14,
14]          512                           True
         Sequential: 3-31               [32, 128, 28, 28]         [32, 256, 14,
14]          33,280                        True
         ReLU: 3-32                     [32, 256, 14, 14]         [32, 256, 14,
14]          --                            --
     BasicBlock: 2-6                    [32, 256, 14, 14]         [32, 256, 14,
14]          --                            True
         Conv2d: 3-33                   [32, 256, 14, 14]         [32, 256, 14,
14]          589,824                       True
         BatchNorm2d: 3-34              [32, 256, 14, 14]         [32, 256, 14,
14]          512                           True
         ReLU: 3-35                     [32, 256, 14, 14]         [32, 256, 14,
14]          --                            --
         Conv2d: 3-36                   [32, 256, 14, 14]         [32, 256, 14,
14]          589,824                       True
         BatchNorm2d: 3-37              [32, 256, 14, 14]         [32, 256, 14,
14]          512                           True
         ReLU: 3-38                     [32, 256, 14, 14]         [32, 256, 14,
14]          --                            --
 Sequential: 1-8                        [32, 256, 14, 14]         [32, 512, 7,
7]           --                            True
     BasicBlock: 2-7                    [32, 256, 14, 14]         [32, 512, 7,
7]           --                            True
         Conv2d: 3-39                   [32, 256, 14, 14]         [32, 512, 7,
7]           1,179,648                     True
         BatchNorm2d: 3-40              [32, 512, 7, 7]           [32, 512, 7,
7]           1,024                         True
         ReLU: 3-41                     [32, 512, 7, 7]           [32, 512, 7,
7]           --                            --
         Conv2d: 3-42                   [32, 512, 7, 7]           [32, 512, 7,
7]           2,359,296                     True
         BatchNorm2d: 3-43              [32, 512, 7, 7]           [32, 512, 7,
7]           1,024                         True
         Sequential: 3-44               [32, 256, 14, 14]         [32, 512, 7,
7]           132,096                       True
         ReLU: 3-45                     [32, 512, 7, 7]           [32, 512, 7,
7]           --                            --
     BasicBlock: 2-8                    [32, 512, 7, 7]           [32, 512, 7,
7]           --                            True
         Conv2d: 3-46                   [32, 512, 7, 7]           [32, 512, 7,
7]           2,359,296                     True
         BatchNorm2d: 3-47              [32, 512, 7, 7]           [32, 512, 7,
7]           1,024                         True
         ReLU: 3-48                     [32, 512, 7, 7]           [32, 512, 7,
7]           --                            --
         Conv2d: 3-49                   [32, 512, 7, 7]           [32, 512, 7,
```

```
7]           2,359,296                True
          BatchNorm2d: 3-50           [32, 512, 7, 7]          [32, 512, 7,
7]           1,024                    True
          ReLU: 3-51                  [32, 512, 7, 7]          [32, 512, 7,
7]           --                       --
 AdaptiveAvgPool2d: 1-9                [32, 512, 7, 7]          [32, 512, 1,
1]           --                       --
 Linear: 1-10                          [32, 512]                [32, 2]
1,026                    True
================================================================================
============================================================
Total params: 11,177,538
Trainable params: 11,177,538
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 58.03
================================================================================
============================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 1271.66
Params size (MB): 44.71
Estimated Total Size (MB): 1335.64
================================================================================
============================================================
```

[38]:
```
loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(resnet_model.parameters(), lr=0.001)
iterations = 30
```

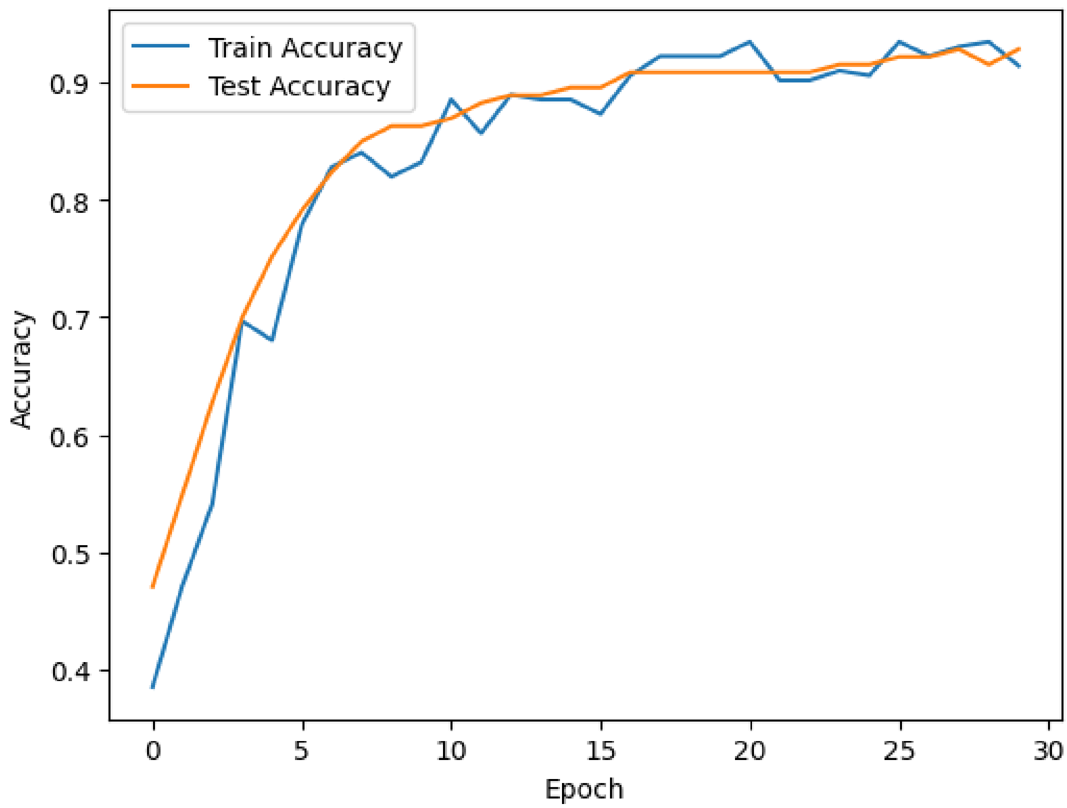
[39]:
```
train_accuracy_hist, test_accuracy_hist, train_loss_hist, test_loss_hist =␣
 ↪train(resnet_model, trainloader_hymenoptera, testloader_hymenoptera,␣
 ↪iterations, optimizer, loss, device)
```

```
Epoch 1 / 30, Train Loss: 0.8398880288004875, Test Loss: 0.7552562475204467,
Train Accuracy: 0.38524590163934425, Test Accuracy: 0.47058823529411764
Epoch 2 / 30, Train Loss: 0.7524572089314461, Test Loss: 0.6763516306877136,
Train Accuracy: 0.4713114754098361, Test Accuracy: 0.5490196078431373
Epoch 3 / 30, Train Loss: 0.669730469584465, Test Loss: 0.6167248964309693,
Train Accuracy: 0.5409836065573771, Test Accuracy: 0.6274509803921569
Epoch 4 / 30, Train Loss: 0.6051743626594543, Test Loss: 0.5640691518783569,
Train Accuracy: 0.6967213114754098, Test Accuracy: 0.6993464052287581
Epoch 5 / 30, Train Loss: 0.5841399431228638, Test Loss: 0.5305917978286743,
Train Accuracy: 0.680327868852459, Test Accuracy: 0.7516339869281046
Epoch 6 / 30, Train Loss: 0.516976211220026, Test Loss: 0.4924739897251129,
Train Accuracy: 0.7786885245901639, Test Accuracy: 0.7908496732026143
Epoch 7 / 30, Train Loss: 0.4892481788992882, Test Loss: 0.4643334150314331,
Train Accuracy: 0.8278688524590164, Test Accuracy: 0.8235294117647058
Epoch 8 / 30, Train Loss: 0.45660873502492905, Test Loss: 0.4346988558769226,
Train Accuracy: 0.8401639344262295, Test Accuracy: 0.8496732026143791
```

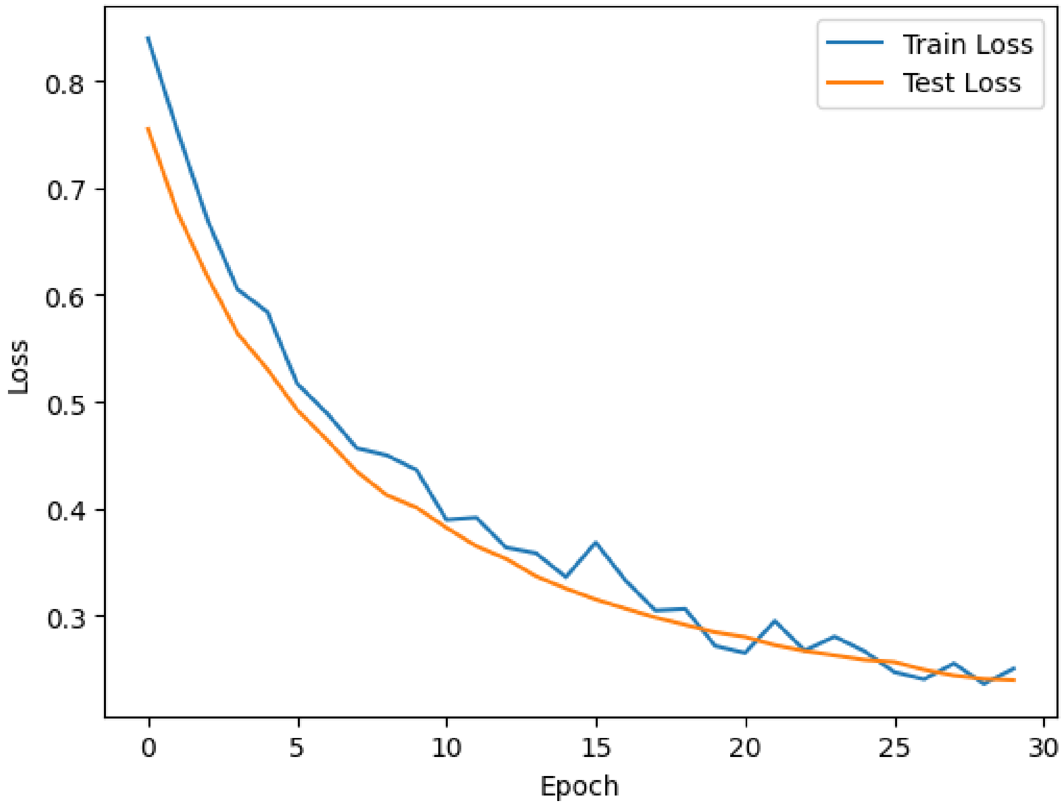
```
Epoch 9 / 30, Train Loss: 0.450012881308794, Test Loss: 0.41279898285865785,
Train Accuracy: 0.819672131147541, Test Accuracy: 0.8627450980392157
Epoch 10 / 30, Train Loss: 0.43615997582674026, Test Loss: 0.4011000096797943,
Train Accuracy: 0.8319672131147541, Test Accuracy: 0.8627450980392157
Epoch 11 / 30, Train Loss: 0.3896719589829445, Test Loss: 0.3822374701499939,
Train Accuracy: 0.8852459016393442, Test Accuracy: 0.869281045751634
Epoch 12 / 30, Train Loss: 0.39170483872294426, Test Loss: 0.36508873105049133,
Train Accuracy: 0.8565573770491803, Test Accuracy: 0.8823529411764706
Epoch 13 / 30, Train Loss: 0.3637463450431824, Test Loss: 0.35320048928260805,
Train Accuracy: 0.889344262295082, Test Accuracy: 0.8888888888888888
Epoch 14 / 30, Train Loss: 0.3584594503045082, Test Loss: 0.3369517534971237,
Train Accuracy: 0.8852459016393442, Test Accuracy: 0.8888888888888888
Epoch 15 / 30, Train Loss: 0.3361276090145111, Test Loss: 0.32540910243988036,
Train Accuracy: 0.8852459016393442, Test Accuracy: 0.8954248366013072
Epoch 16 / 30, Train Loss: 0.36849259585142136, Test Loss: 0.31521751880645754,
Train Accuracy: 0.8729508196721312, Test Accuracy: 0.8954248366013072
Epoch 17 / 30, Train Loss: 0.3329554833471775, Test Loss: 0.30659289956092833,
Train Accuracy: 0.9057377049180327, Test Accuracy: 0.9084967320261438
Epoch 18 / 30, Train Loss: 0.30484870448708534, Test Loss: 0.29837953150272367,
Train Accuracy: 0.9221311475409836, Test Accuracy: 0.9084967320261438
Epoch 19 / 30, Train Loss: 0.306391678750515, Test Loss: 0.29148478507995607,
Train Accuracy: 0.9221311475409836, Test Accuracy: 0.9084967320261438
Epoch 20 / 30, Train Loss: 0.27180954068899155, Test Loss: 0.2845373719930649,
Train Accuracy: 0.9221311475409836, Test Accuracy: 0.9084967320261438
Epoch 21 / 30, Train Loss: 0.26524688862264156, Test Loss: 0.2801322817802429,
Train Accuracy: 0.9344262295081968, Test Accuracy: 0.9084967320261438
Epoch 22 / 30, Train Loss: 0.29497666098177433, Test Loss: 0.2725300848484039,
Train Accuracy: 0.9016393442622951, Test Accuracy: 0.9084967320261438
Epoch 23 / 30, Train Loss: 0.2674539815634489, Test Loss: 0.2670877307653427,
Train Accuracy: 0.9016393442622951, Test Accuracy: 0.9084967320261438
Epoch 24 / 30, Train Loss: 0.28044826351106167, Test Loss: 0.26296400725841523,
Train Accuracy: 0.9098360655737705, Test Accuracy: 0.9150326797385621
Epoch 25 / 30, Train Loss: 0.266862602904439, Test Loss: 0.25873576700687406,
Train Accuracy: 0.9057377049180327, Test Accuracy: 0.9150326797385621
Epoch 26 / 30, Train Loss: 0.24728692881762981, Test Loss: 0.2566778600215912,
Train Accuracy: 0.9344262295081968, Test Accuracy: 0.9215686274509803
Epoch 27 / 30, Train Loss: 0.24057933874428272, Test Loss: 0.2494908958673477,
Train Accuracy: 0.9221311475409836, Test Accuracy: 0.9215686274509803
Epoch 28 / 30, Train Loss: 0.25542024709284306, Test Loss: 0.2440810650587082,
Train Accuracy: 0.930327868852459, Test Accuracy: 0.9281045751633987
Epoch 29 / 30, Train Loss: 0.23629208281636238, Test Loss: 0.24093491435050965,
Train Accuracy: 0.9344262295081968, Test Accuracy: 0.9150326797385621
Epoch 30 / 30, Train Loss: 0.2505239322781563, Test Loss: 0.23970454931259155,
Train Accuracy: 0.9139344262295082, Test Accuracy: 0.9281045751633987
```

[40]:
```
plt.plot(train_accuracy_hist, label='Train Accuracy')
plt.plot(test_accuracy_hist, label='Test Accuracy')
```

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



[41]:
```python
plt.plot(train_loss_hist, label='Train Loss')
plt.plot(test_loss_hist, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



We can observe the model to be performing well as the train and test losses are decreasing and the train and test accuracies are increasing with each epoch. After 30 epochs, the model was able to achieve a test accuracy of 92.81%.

3.2 Using ResNet-18 as a feature extractor

[42]:
```
resnet_model = torchvision.models.resnet18(weights = 'IMAGENET1K_V1').
↪to(device) # We need to reobtain the model as we have modified it previously
resnet_model.fc = nn.Linear(512, len(classes_hymenoptera)).to(device)
for param in resnet_model.parameters():
    param.requires_grad = False

for param in resnet_model.fc.parameters():
    param.requires_grad = True
```

[43]:
```
loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(resnet_model.parameters(), lr=0.001)
iterations = 30
```

[44]:
```
print(summary(resnet_model, input_size=(batch_size, 3, 224, 224),␣
↪col_names=["input_size", "output_size", "num_params", "trainable"]))
```

```
================================================================================
============================================================
Layer (type:depth-idx)                   Input Shape               Output Shape
Param #                  Trainable
================================================================================
============================================================
ResNet                                   [32, 3, 224, 224]         [32, 2]
--                       Partial
 Conv2d: 1-1                             [32, 3, 224, 224]         [32, 64, 112,
112]          (9,408)                    False
 BatchNorm2d: 1-2                        [32, 64, 112, 112]        [32, 64, 112,
112]          (128)                      False
 ReLU: 1-3                               [32, 64, 112, 112]        [32, 64, 112,
112]          --                         --
 MaxPool2d: 1-4                          [32, 64, 112, 112]        [32, 64, 56,
56]           --                         --
 Sequential: 1-5                         [32, 64, 56, 56]          [32, 64, 56,
56]           --                         False
      BasicBlock: 2-1                    [32, 64, 56, 56]          [32, 64, 56,
56]           --                         False
          Conv2d: 3-1                    [32, 64, 56, 56]          [32, 64, 56,
56]           (36,864)                   False
          BatchNorm2d: 3-2               [32, 64, 56, 56]          [32, 64, 56,
56]           (128)                      False
          ReLU: 3-3                      [32, 64, 56, 56]          [32, 64, 56,
56]           --                         --
          Conv2d: 3-4                    [32, 64, 56, 56]          [32, 64, 56,
56]           (36,864)                   False
          BatchNorm2d: 3-5               [32, 64, 56, 56]          [32, 64, 56,
56]           (128)                      False
```

```
        ReLU: 3-6                      [32, 64, 56, 56]          [32, 64, 56,
56]        --                          --
    BasicBlock: 2-2                    [32, 64, 56, 56]          [32, 64, 56,
56]        --                          False
        Conv2d: 3-7                    [32, 64, 56, 56]          [32, 64, 56,
56]        (36,864)                    False
        BatchNorm2d: 3-8               [32, 64, 56, 56]          [32, 64, 56,
56]        (128)                       False
        ReLU: 3-9                      [32, 64, 56, 56]          [32, 64, 56,
56]        --                          --
        Conv2d: 3-10                   [32, 64, 56, 56]          [32, 64, 56,
56]        (36,864)                    False
        BatchNorm2d: 3-11              [32, 64, 56, 56]          [32, 64, 56,
56]        (128)                       False
        ReLU: 3-12                     [32, 64, 56, 56]          [32, 64, 56,
56]        --                          --
 Sequential: 1-6                       [32, 64, 56, 56]          [32, 128, 28,
28]       --                          False
    BasicBlock: 2-3                    [32, 64, 56, 56]          [32, 128, 28,
28]       --                          False
        Conv2d: 3-13                   [32, 64, 56, 56]          [32, 128, 28,
28]       (73,728)                    False
        BatchNorm2d: 3-14              [32, 128, 28, 28]         [32, 128, 28,
28]       (256)                       False
        ReLU: 3-15                     [32, 128, 28, 28]         [32, 128, 28,
28]       --                          --
        Conv2d: 3-16                   [32, 128, 28, 28]         [32, 128, 28,
28]       (147,456)                   False
        BatchNorm2d: 3-17              [32, 128, 28, 28]         [32, 128, 28,
28]       (256)                       False
        Sequential: 3-18               [32, 64, 56, 56]          [32, 128, 28,
28]       (8,448)                     False
        ReLU: 3-19                     [32, 128, 28, 28]         [32, 128, 28,
28]       --                          --
    BasicBlock: 2-4                    [32, 128, 28, 28]         [32, 128, 28,
28]       --                          False
        Conv2d: 3-20                   [32, 128, 28, 28]         [32, 128, 28,
28]       (147,456)                   False
        BatchNorm2d: 3-21              [32, 128, 28, 28]         [32, 128, 28,
28]       (256)                       False
        ReLU: 3-22                     [32, 128, 28, 28]         [32, 128, 28,
28]       --                          --
        Conv2d: 3-23                   [32, 128, 28, 28]         [32, 128, 28,
28]       (147,456)                   False
        BatchNorm2d: 3-24              [32, 128, 28, 28]         [32, 128, 28,
28]       (256)                       False
        ReLU: 3-25                     [32, 128, 28, 28]         [32, 128, 28,
28]       --                          --
```

```
 Sequential: 1-7                          [32, 128, 28, 28]          [32, 256, 14,
14]         --                          False
     BasicBlock: 2-5                      [32, 128, 28, 28]          [32, 256, 14,
14]         --                          False
          Conv2d: 3-26                    [32, 128, 28, 28]          [32, 256, 14,
14]         (294,912)                   False
          BatchNorm2d: 3-27               [32, 256, 14, 14]          [32, 256, 14,
14]         (512)                       False
          ReLU: 3-28                      [32, 256, 14, 14]          [32, 256, 14,
14]         --                          --
          Conv2d: 3-29                    [32, 256, 14, 14]          [32, 256, 14,
14]         (589,824)                   False
          BatchNorm2d: 3-30               [32, 256, 14, 14]          [32, 256, 14,
14]         (512)                       False
          Sequential: 3-31                [32, 128, 28, 28]          [32, 256, 14,
14]         (33,280)                    False
          ReLU: 3-32                      [32, 256, 14, 14]          [32, 256, 14,
14]         --                          --
     BasicBlock: 2-6                      [32, 256, 14, 14]          [32, 256, 14,
14]         --                          False
          Conv2d: 3-33                    [32, 256, 14, 14]          [32, 256, 14,
14]         (589,824)                   False
          BatchNorm2d: 3-34               [32, 256, 14, 14]          [32, 256, 14,
14]         (512)                       False
          ReLU: 3-35                      [32, 256, 14, 14]          [32, 256, 14,
14]         --                          --
          Conv2d: 3-36                    [32, 256, 14, 14]          [32, 256, 14,
14]         (589,824)                   False
          BatchNorm2d: 3-37               [32, 256, 14, 14]          [32, 256, 14,
14]         (512)                       False
          ReLU: 3-38                      [32, 256, 14, 14]          [32, 256, 14,
14]         --                          --
 Sequential: 1-8                          [32, 256, 14, 14]          [32, 512, 7,
7]           --                          False
     BasicBlock: 2-7                      [32, 256, 14, 14]          [32, 512, 7,
7]           --                          False
          Conv2d: 3-39                    [32, 256, 14, 14]          [32, 512, 7,
7]           (1,179,648)                 False
          BatchNorm2d: 3-40               [32, 512, 7, 7]            [32, 512, 7,
7]           (1,024)                     False
          ReLU: 3-41                      [32, 512, 7, 7]            [32, 512, 7,
7]           --                          --
          Conv2d: 3-42                    [32, 512, 7, 7]            [32, 512, 7,
7]           (2,359,296)                 False
          BatchNorm2d: 3-43               [32, 512, 7, 7]            [32, 512, 7,
7]           (1,024)                     False
          Sequential: 3-44                [32, 256, 14, 14]          [32, 512, 7,
7]           (132,096)                   False
```

```
          ReLU: 3-45                    [32, 512, 7, 7]           [32, 512, 7,
7]          --                          --
      BasicBlock: 2-8                    [32, 512, 7, 7]           [32, 512, 7,
7]          --                          False
          Conv2d: 3-46                  [32, 512, 7, 7]           [32, 512, 7,
7]          (2,359,296)                 False
          BatchNorm2d: 3-47             [32, 512, 7, 7]           [32, 512, 7,
7]          (1,024)                     False
          ReLU: 3-48                    [32, 512, 7, 7]           [32, 512, 7,
7]          --                          --
          Conv2d: 3-49                  [32, 512, 7, 7]           [32, 512, 7,
7]          (2,359,296)                 False
          BatchNorm2d: 3-50             [32, 512, 7, 7]           [32, 512, 7,
7]          (1,024)                     False
          ReLU: 3-51                    [32, 512, 7, 7]           [32, 512, 7,
7]          --                          --
 AdaptiveAvgPool2d: 1-9                  [32, 512, 7, 7]           [32, 512, 1,
1]          --                          --
 Linear: 1-10                            [32, 512]                 [32, 2]
1,026                     True
================================================================================
============================================================
Total params: 11,177,538
Trainable params: 1,026
Non-trainable params: 11,176,512
Total mult-adds (Units.GIGABYTES): 58.03
================================================================================
============================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 1271.66
Params size (MB): 44.71
Estimated Total Size (MB): 1335.64
================================================================================
============================================================
```

[45]:

```python
train_accuracy_hist, test_accuracy_hist, train_loss_hist, test_loss_hist =
    train(resnet_model, trainloader_hymenoptera, testloader_hymenoptera,
    iterations, optimizer, loss, device)
```

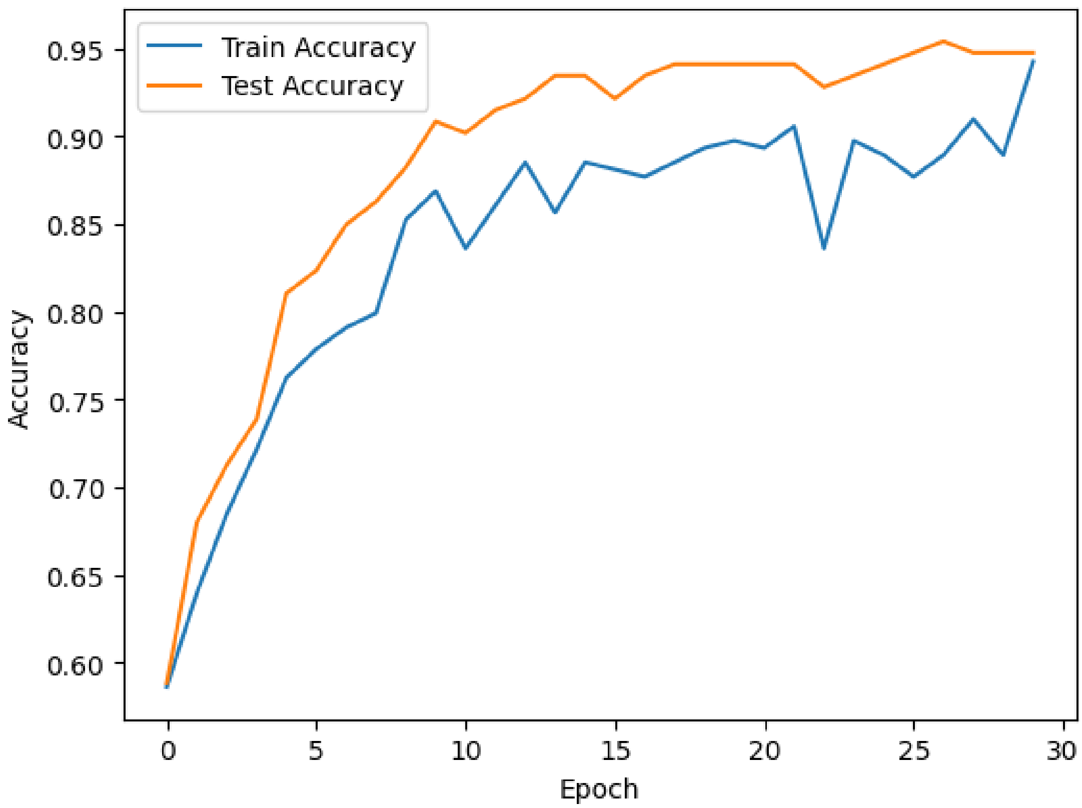
```
Epoch 1 / 30, Train Loss: 0.6621313095092773, Test Loss: 0.6414272069931031,
Train Accuracy: 0.5860655737704918, Test Accuracy: 0.5882352941176471
Epoch 2 / 30, Train Loss: 0.6426210552453995, Test Loss: 0.6073173403739929,
Train Accuracy: 0.639344262295082, Test Accuracy: 0.6797385620915033
Epoch 3 / 30, Train Loss: 0.6067900285124779, Test Loss: 0.5845212697982788,
Train Accuracy: 0.6844262295081968, Test Accuracy: 0.7124183006535948
Epoch 4 / 30, Train Loss: 0.5837375596165657, Test Loss: 0.5580122470855713,
Train Accuracy: 0.7213114754098361, Test Accuracy: 0.738562091503268
Epoch 5 / 30, Train Loss: 0.5456314645707607, Test Loss: 0.5373104989528656,
```

```
Train Accuracy: 0.7622950819672131, Test Accuracy: 0.8104575163398693
Epoch 6 / 30, Train Loss: 0.5299510098993778, Test Loss: 0.510362184047699,
Train Accuracy: 0.7786885245901639, Test Accuracy: 0.8235294117647058
Epoch 7 / 30, Train Loss: 0.5166481956839561, Test Loss: 0.49135775566101075,
Train Accuracy: 0.7909836065573771, Test Accuracy: 0.8496732026143791
Epoch 8 / 30, Train Loss: 0.49842673540115356, Test Loss: 0.4719221830368042,
Train Accuracy: 0.7991803278688525, Test Accuracy: 0.8627450980392157
Epoch 9 / 30, Train Loss: 0.4801369719207287, Test Loss: 0.457780134677887,
Train Accuracy: 0.8524590163934426, Test Accuracy: 0.8823529411764706
Epoch 10 / 30, Train Loss: 0.4504745453596115, Test Loss: 0.43878700733184817,
Train Accuracy: 0.8688524590163934, Test Accuracy: 0.9084967320261438
Epoch 11 / 30, Train Loss: 0.4289609156548977, Test Loss: 0.4310948669910431,
Train Accuracy: 0.8360655737704918, Test Accuracy: 0.9019607843137255
Epoch 12 / 30, Train Loss: 0.4363722987473011, Test Loss: 0.4210783183574677,
Train Accuracy: 0.860655737704918, Test Accuracy: 0.9150326797385621
Epoch 13 / 30, Train Loss: 0.4192662909626961, Test Loss: 0.4022731363773346,
Train Accuracy: 0.8852459016393442, Test Accuracy: 0.9215686274509803
Epoch 14 / 30, Train Loss: 0.4181019887328148, Test Loss: 0.3875797212123871,
Train Accuracy: 0.8565573770491803, Test Accuracy: 0.934640522875817
Epoch 15 / 30, Train Loss: 0.383378766477108, Test Loss: 0.38148173689842224,
Train Accuracy: 0.8852459016393442, Test Accuracy: 0.934640522875817
Epoch 16 / 30, Train Loss: 0.39727528393268585, Test Loss: 0.3802091419696808,
Train Accuracy: 0.8811475409836066, Test Accuracy: 0.9215686274509803
Epoch 17 / 30, Train Loss: 0.39227911457419395, Test Loss: 0.3675357699394226,
Train Accuracy: 0.8770491803278688, Test Accuracy: 0.934640522875817
Epoch 18 / 30, Train Loss: 0.37590841576457024, Test Loss: 0.35887229442596436,
Train Accuracy: 0.8852459016393442, Test Accuracy: 0.9411764705882353
Epoch 19 / 30, Train Loss: 0.36022039875388145, Test Loss: 0.3502146929502487,
Train Accuracy: 0.8934426229508197, Test Accuracy: 0.9411764705882353
Epoch 20 / 30, Train Loss: 0.3561762683093548, Test Loss: 0.3439750701189041,
Train Accuracy: 0.8975409836065574, Test Accuracy: 0.9411764705882353
Epoch 21 / 30, Train Loss: 0.3480180464684963, Test Loss: 0.33838188350200654,
Train Accuracy: 0.8934426229508197, Test Accuracy: 0.9411764705882353
Epoch 22 / 30, Train Loss: 0.3487807735800743, Test Loss: 0.3327960163354874,
Train Accuracy: 0.9057377049180327, Test Accuracy: 0.9411764705882353
Epoch 23 / 30, Train Loss: 0.40001729503273964, Test Loss: 0.33436612486839296,
Train Accuracy: 0.8360655737704918, Test Accuracy: 0.9281045751633987
Epoch 24 / 30, Train Loss: 0.3523205555975437, Test Loss: 0.32409325838088987,
Train Accuracy: 0.8975409836065574, Test Accuracy: 0.934640522875817
Epoch 25 / 30, Train Loss: 0.34047113358974457, Test Loss: 0.3185803860425949,
Train Accuracy: 0.889344262295082, Test Accuracy: 0.9411764705882353
Epoch 26 / 30, Train Loss: 0.3624703921377659, Test Loss: 0.3155204474925995,
Train Accuracy: 0.8770491803278688, Test Accuracy: 0.9477124183006536
Epoch 27 / 30, Train Loss: 0.35706062987446785, Test Loss: 0.3037082076072693,
Train Accuracy: 0.889344262295082, Test Accuracy: 0.954248366013072
Epoch 28 / 30, Train Loss: 0.3190745823085308, Test Loss: 0.3006920456886292,
Train Accuracy: 0.9098360655737705, Test Accuracy: 0.9477124183006536
Epoch 29 / 30, Train Loss: 0.32203957065939903, Test Loss: 0.29636829197406767,
```

```
Train Accuracy: 0.889344262295082, Test Accuracy: 0.9477124183006536
Epoch 30 / 30, Train Loss: 0.2873081173747778, Test Loss: 0.2917454868555069,
Train Accuracy: 0.9426229508196722, Test Accuracy: 0.9477124183006536
```

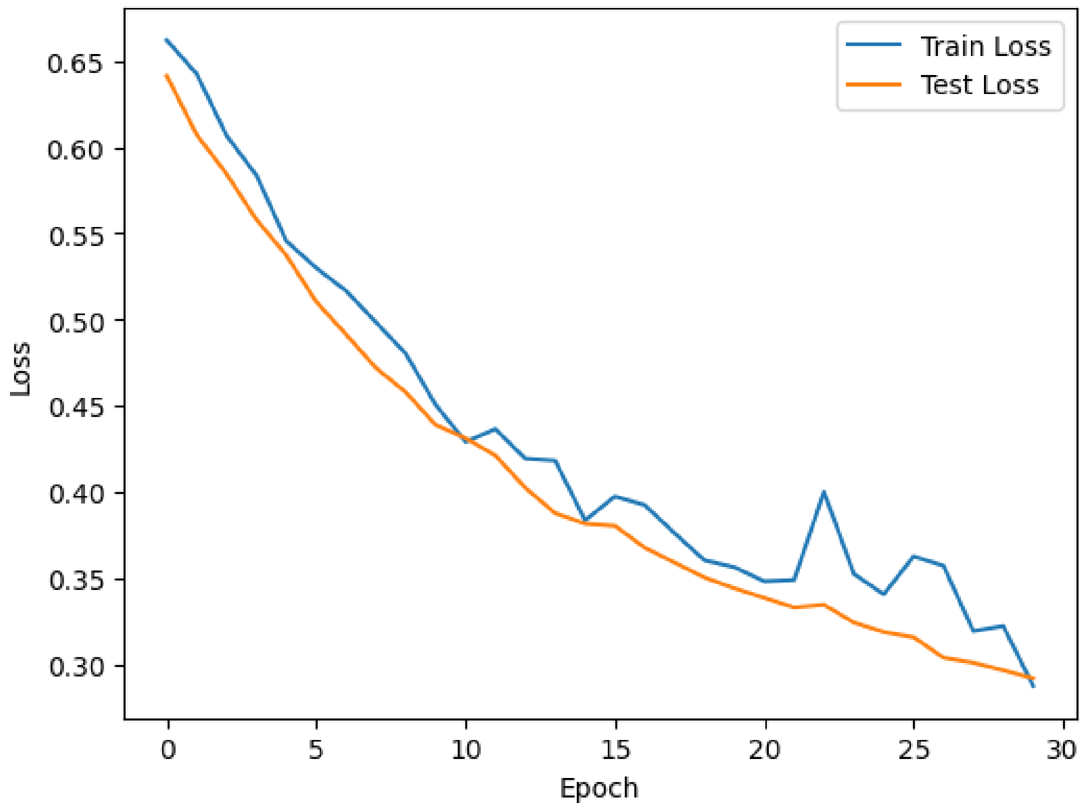
[46]:

```
plt.plot(train_accuracy_hist, label='Train Accuracy')
plt.plot(test_accuracy_hist, label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[47]:

```
plt.plot(train_loss_hist, label='Train Loss')
plt.plot(test_loss_hist, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



When using the model as a feature extractor, the performance is still very good. After 30 epochs, the model was able to achieve a test accuracy of 94.77% which exceeds the performance when it was finetuned. However, the accuracy on the training set is lower than when the model was finetuned. This is likely due to the model overfitting when it was finetuned. This behaviour was repeatably observed.