210236P_a03

November 11, 2024

EN3160 Assignment 3 on Neural Networks

Instructed by Dr. Ranga Rodrigo

Done by Jayakumar W.S. (210236P)

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

[73]:
```
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 = [ ]
```

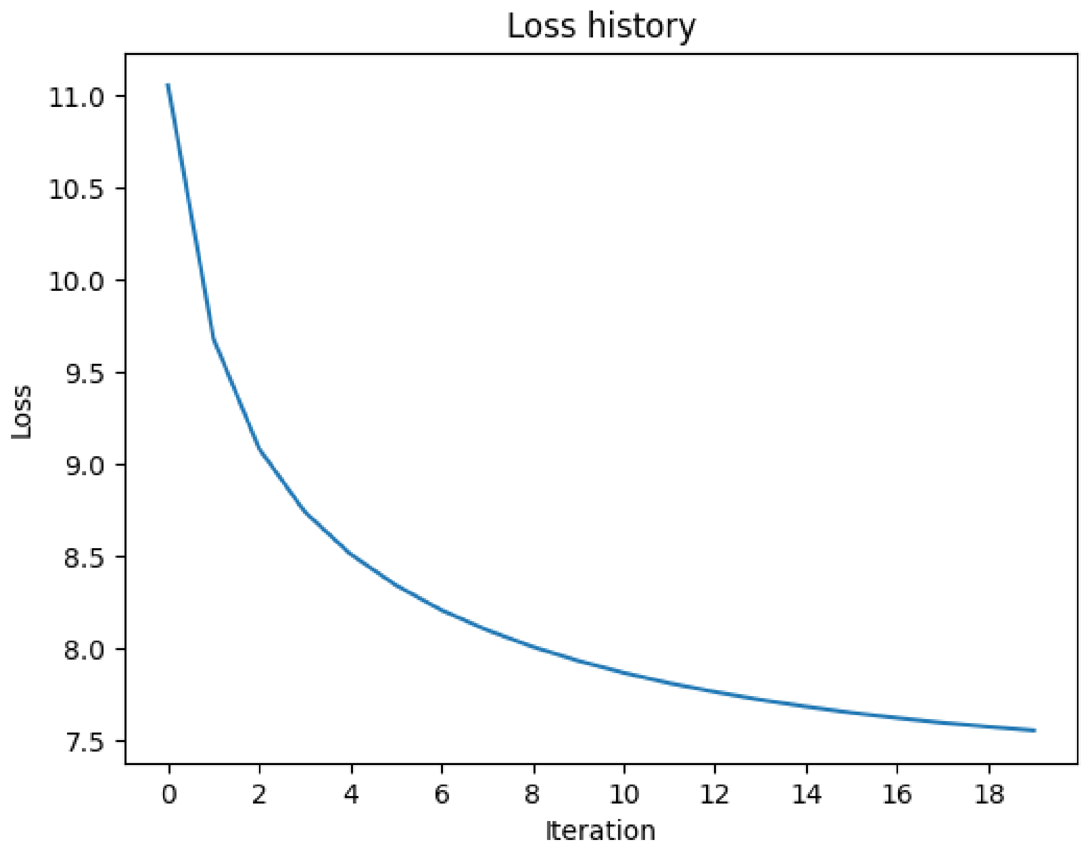
[5]:
```
for t in range(iterations):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # Get inputs and labels
        inputs, labels = data
        Ntr = inputs.shape[0]  # Batch size
        x_train = inputs.view(Ntr, -1).to(device)  # Flatten input to (Ntr, Din)
        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: 11.052271098565841
Epoch 2 / 20, Loss: 9.677958526294047
Epoch 3 / 20, Loss: 9.081760541033608
Epoch 4 / 20, Loss: 8.739522613627896
Epoch 5 / 20, Loss: 8.51057107678912
Epoch 6 / 20, Loss: 8.341256552602873
Epoch 7 / 20, Loss: 8.205948939479015
Epoch 8 / 20, Loss: 8.09767150421289
Epoch 9 / 20, Loss: 8.007261962060813
Epoch 10 / 20, Loss: 7.930841871506879
Epoch 11 / 20, Loss: 7.865830220553788
Epoch 12 / 20, Loss: 7.809729950746815
Epoch 13 / 20, Loss: 7.762098590914286
Epoch 14 / 20, Loss: 7.719234191372237
Epoch 15 / 20, Loss: 7.682436113851771
Epoch 16 / 20, Loss: 7.6493737638111075
Epoch 17 / 20, Loss: 7.620805551513066
Epoch 18 / 20, Loss: 7.594736617570952
Epoch 19 / 20, Loss: 7.573210502997927
Epoch 20 / 20, Loss: 7.552905830449197
```

[11]:
```
plt.plot(loss_history)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.xticks(range(0, iterations, 2))
```

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



[4]:
```
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
```

```
[13]: 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: 10.12%
Test accuracy: 10.17%
```

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

```
[ ]: 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

```
[7]: # 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 = [ ]
```

```
[6]: 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)
```

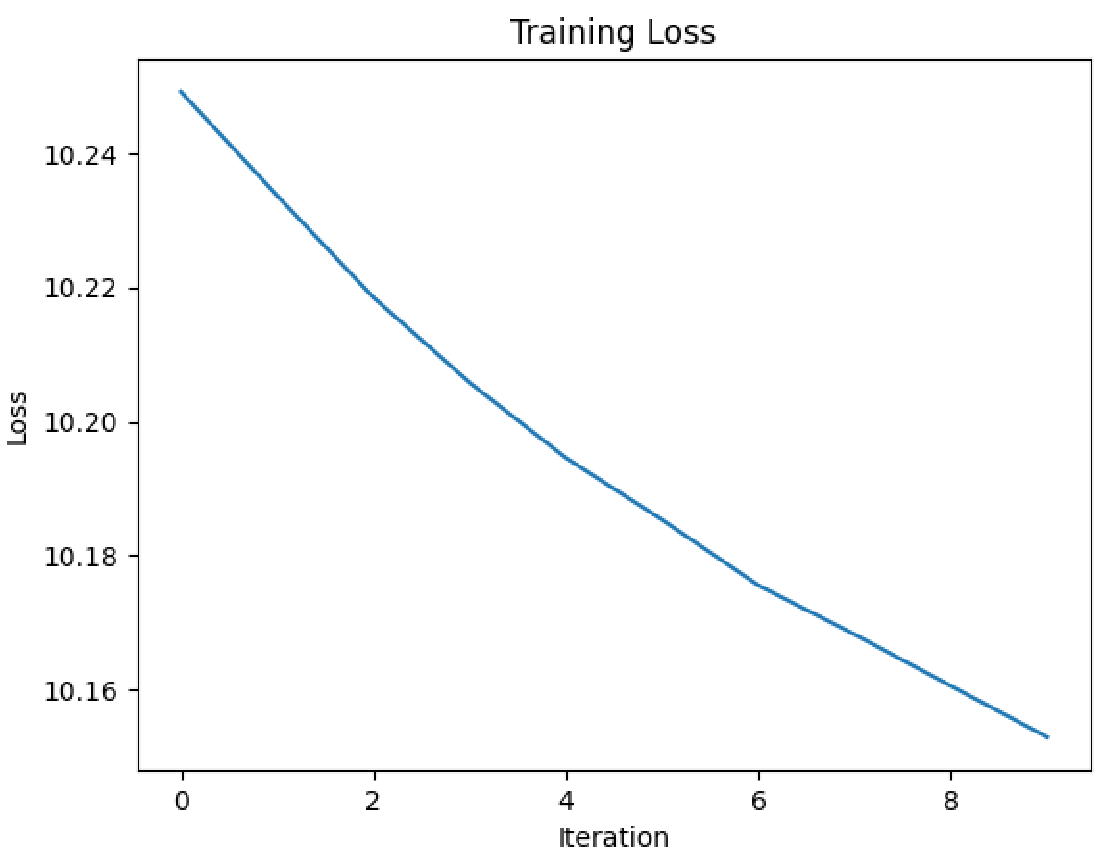
```
        y_train_onehot = nn.functional.one_hot(labels, K).float().to(device)  # One-hot labels

        # 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: 10.249201878323742
Epoch 2 / 10, Loss: 10.233684833890264
Epoch 3 / 10, Loss: 10.218501103511622
Epoch 4 / 10, Loss: 10.20577982977576
Epoch 5 / 10, Loss: 10.194548347861204
Epoch 6 / 10, Loss: 10.185316179169346
Epoch 7 / 10, Loss: 10.175520200418191
Epoch 8 / 10, Loss: 10.168194400143028
Epoch 9 / 10, Loss: 10.160515865147762
Epoch 10 / 10, Loss: 10.152873168598743
```

It is observed that the loss values decrease on each iteration

[7]:
```
plt.plot(loss_history)
plt.xlabel('Iteration')
plt.xticks(range(0, iterations, 2))
plt.ylabel('Loss')
plt.title('Training Loss')
plt.show()
```



[8]:
```
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
```

[9]:
```
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: 10.05%
Test accuracy: 9.77%
```

[10]:
```
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

[13]:
```
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
```

[16]:
```
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

[17]:
```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
```

[18]:
```
model = NeuralNetwork(Din, 100, K).to(device)
loss = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=reg)
iterations = 20
```

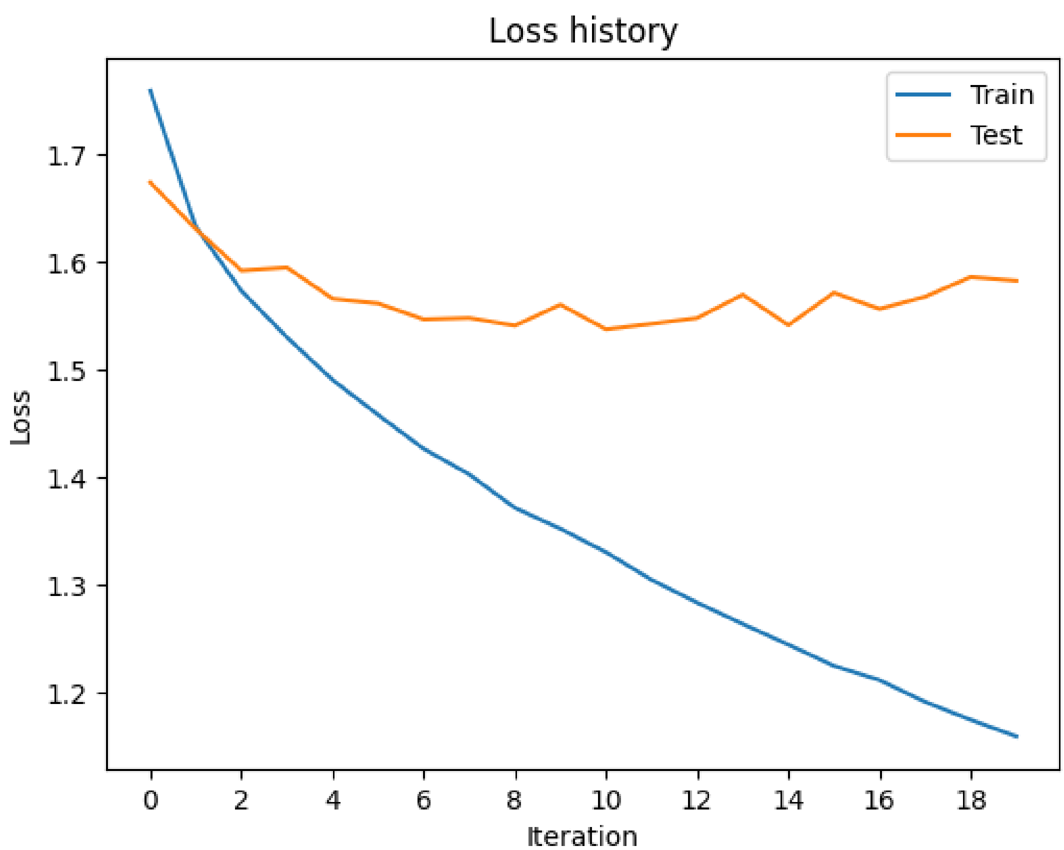
[19]:
```
train_accuracy_hist, test_accuracy_hist, train_loss_hist, test_loss_hist =␣
↪train(model, trainloader, testloader, iterations, optimizer, loss, device)
```

```
Epoch 1 / 20, Train Loss: 1.7594276244870684, Test Loss: 1.674093100590447,
Train Accuracy: 0.38396, Test Accuracy: 0.4194
Epoch 2 / 20, Train Loss: 1.6328852262088143, Test Loss: 1.630866586209867,
Train Accuracy: 0.43388, Test Accuracy: 0.4355
Epoch 3 / 20, Train Loss: 1.5735069845856113, Test Loss: 1.592388990968942,
Train Accuracy: 0.45558, Test Accuracy: 0.4492
Epoch 4 / 20, Train Loss: 1.5303320414121533, Test Loss: 1.5951702092021418,
Train Accuracy: 0.46966, Test Accuracy: 0.4508
Epoch 5 / 20, Train Loss: 1.490853445391127, Test Loss: 1.5662361551016664,
Train Accuracy: 0.48536, Test Accuracy: 0.4563
Epoch 6 / 20, Train Loss: 1.4583516220061046, Test Loss: 1.5618095154198595,
Train Accuracy: 0.4966, Test Accuracy: 0.4568
Epoch 7 / 20, Train Loss: 1.426805178209977, Test Loss: 1.547024865119983, Train
Accuracy: 0.50792, Test Accuracy: 0.469
Epoch 8 / 20, Train Loss: 1.403101083985217, Test Loss: 1.5482690997017077,
Train Accuracy: 0.51538, Test Accuracy: 0.4635
Epoch 9 / 20, Train Loss: 1.3720098289250564, Test Loss: 1.5411850149258257,
Train Accuracy: 0.5249, Test Accuracy: 0.4682
Epoch 10 / 20, Train Loss: 1.352620396412723, Test Loss: 1.5606188214244172,
Train Accuracy: 0.53088, Test Accuracy: 0.4726
Epoch 11 / 20, Train Loss: 1.3308184827205392, Test Loss: 1.5377594291592558,
Train Accuracy: 0.53986, Test Accuracy: 0.4712
Epoch 12 / 20, Train Loss: 1.3051582179768149, Test Loss: 1.5428463003505914,
Train Accuracy: 0.54882, Test Accuracy: 0.4692
Epoch 13 / 20, Train Loss: 1.2840144001591, Test Loss: 1.5482081445261313, Train
Accuracy: 0.55636, Test Accuracy: 0.4655
Epoch 14 / 20, Train Loss: 1.2642350733394587, Test Loss: 1.569972787992642,
Train Accuracy: 0.56242, Test Accuracy: 0.4648
Epoch 15 / 20, Train Loss: 1.2450444423000704, Test Loss: 1.541662802330602,
Train Accuracy: 0.57068, Test Accuracy: 0.4714
Epoch 16 / 20, Train Loss: 1.2253786445579236, Test Loss: 1.5717284226189026,
Train Accuracy: 0.57796, Test Accuracy: 0.4673
```

```
Epoch 17 / 20, Train Loss: 1.2121384872203445, Test Loss: 1.5566994610685891,
Train Accuracy: 0.58184, Test Accuracy: 0.4682
Epoch 18 / 20, Train Loss: 1.191947989752105, Test Loss: 1.5680276741996741,
Train Accuracy: 0.58844, Test Accuracy: 0.4679
Epoch 19 / 20, Train Loss: 1.175236813356994, Test Loss: 1.5862781468290872,
Train Accuracy: 0.59512, Test Accuracy: 0.4657
Epoch 20 / 20, Train Loss: 1.1599616979988279, Test Loss: 1.5828485254662485,
Train Accuracy: 0.5981, Test Accuracy: 0.469
```

[20]:

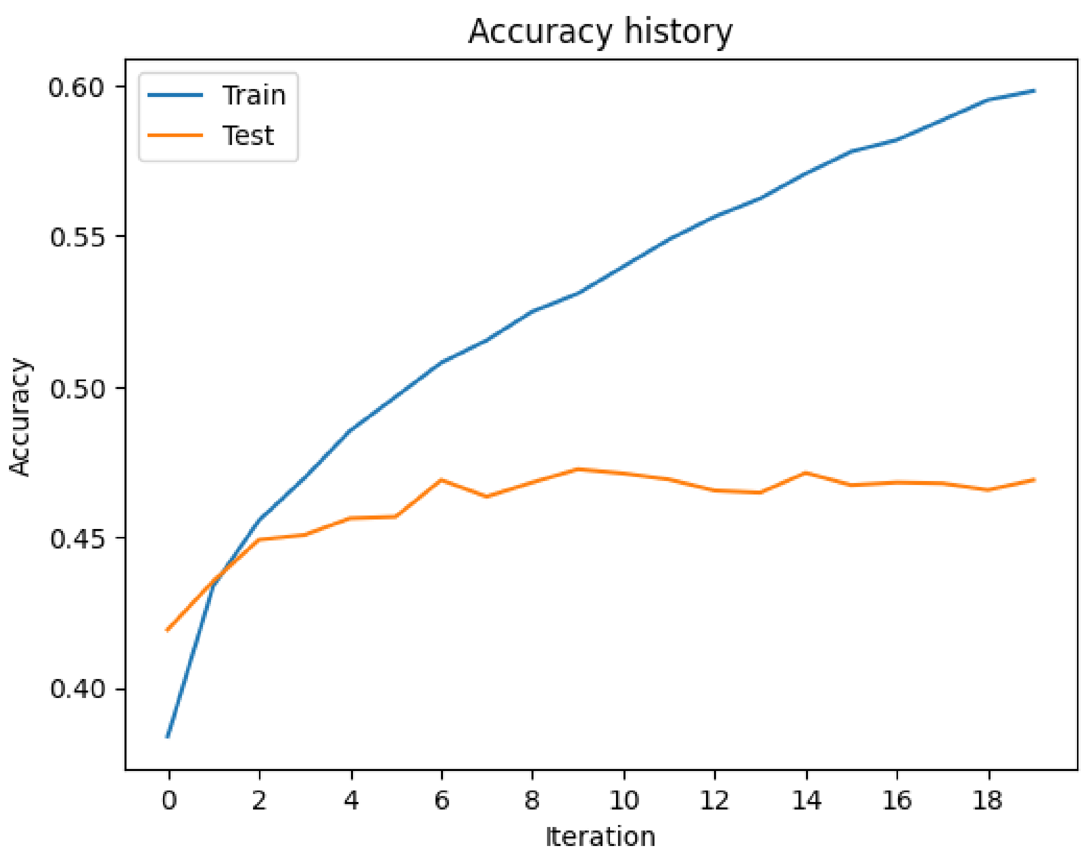
```
plt.plot(train_loss_hist, label='Train')
plt.plot(test_loss_hist, label='Test')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.xticks(range(0, iterations, 2))
plt.title('Loss history')
plt.legend()
plt.show()
```



[21]:

```
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()
```



[22]:
```
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
```

[23]:
```
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: 61.56%
Test accuracy: 46.90%
```

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.

[ ]:
```
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.

[24]:
```
batch_size = 32
```

[25]:
```
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

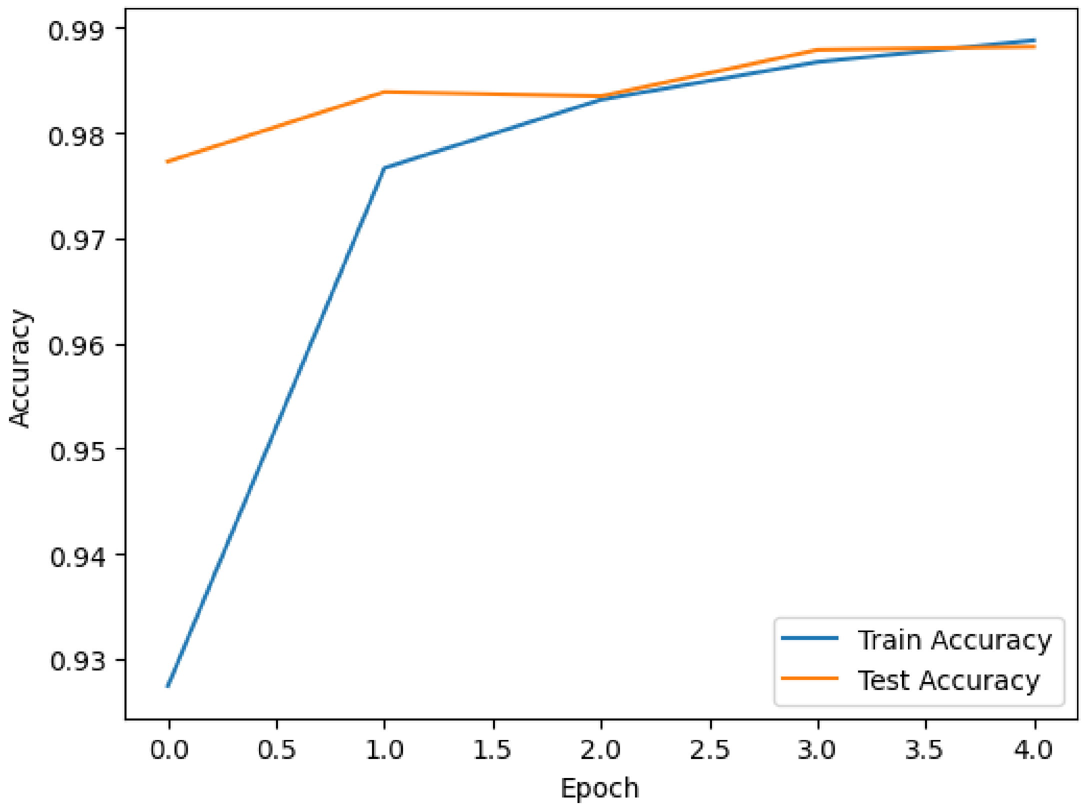
[26]:
```
class LeNet(nn.Module):
    def __init__(self, input_size, input_channels, output_size):
        super(LeNet, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(input_channels, 6, 5),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.conv2 = nn.Sequential(
            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
```

[34]:
```
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
```

[35]:
```
train_accuracy_hist, test_accuracy_hist, train_loss_hist, test_loss_hist =␣
    ↪train(lenet_model, trainloader_mnist, testloader_mnist, iterations,␣
    ↪optimizer, loss, device)
```

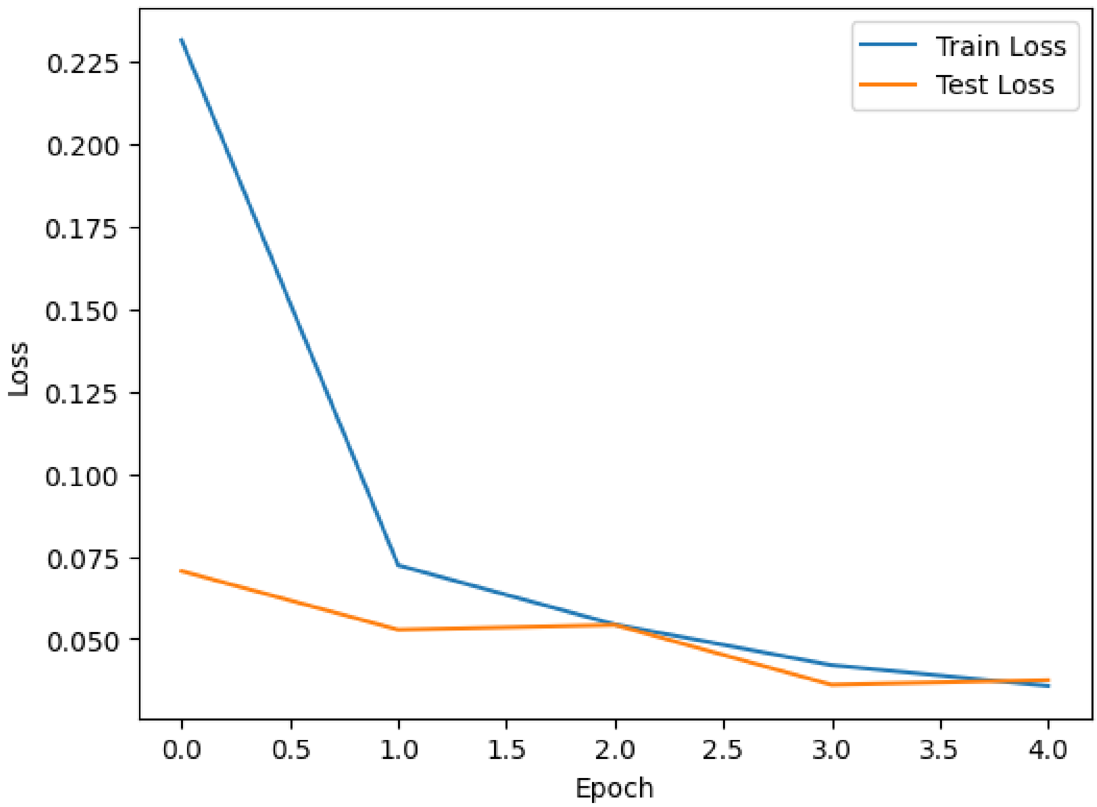
```
Epoch 1 / 5, Train Loss: 0.23151897346563638, Test Loss: 0.07052079150938584,
Train Accuracy: 0.92745, Test Accuracy: 0.9773
Epoch 2 / 5, Train Loss: 0.07228330143981923, Test Loss: 0.052754656721598105,
Train Accuracy: 0.9766833333333333, Test Accuracy: 0.9839
Epoch 3 / 5, Train Loss: 0.054350806503665326, Test Loss: 0.05420349845966769,
Train Accuracy: 0.9831666666666666, Test Accuracy: 0.9835
Epoch 4 / 5, Train Loss: 0.04200671799731596, Test Loss: 0.036045840951620965,
Train Accuracy: 0.98675, Test Accuracy: 0.9879
Epoch 5 / 5, Train Loss: 0.035717702910013034, Test Loss: 0.037477730094514015,
Train Accuracy: 0.9888166666666667, Test Accuracy: 0.9882
```

[36]:
```
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()
```



[37]:

```
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()
```



[38]:
```
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: 99.33%
Test accuracy: 98.82%
```

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 98.82%. 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 th 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

[59]:
```
resnet_model = torchvision.models.resnet18(weights = 'IMAGENET1K_V1').
↪to(device) # These are the default weights
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
```

[60]:
```
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
================================================================================
============================================================
```

[61]:
```
resnet_model.fc = nn.Linear(512, len(classes_hymenoptera)).to(device)
```

[62]:
```
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
================================================================================
============================================================
```

[63]:

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

[64]:

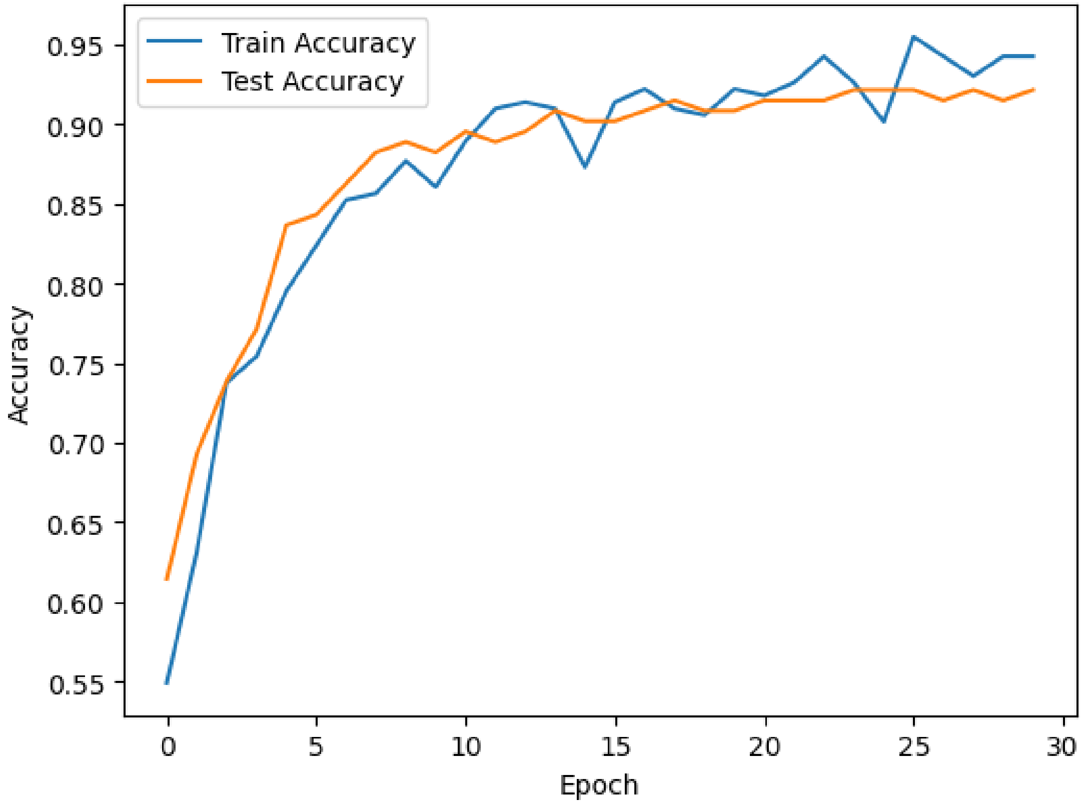
```
train_accuracy_hist, test_accuracy_hist, train_loss_hist, test_loss_hist =␣
↪train(resnet_model, trainloader_hymenoptera, testloader_hymenoptera,␣
↪iterations, optimizer, loss, device)
```

```
Epoch 1 / 30, Train Loss: 0.740321509540081, Test Loss: 0.6342014193534851,
Train Accuracy: 0.5491803278688525, Test Accuracy: 0.6143790849673203
Epoch 2 / 30, Train Loss: 0.6432950645685196, Test Loss: 0.5895971953868866,
Train Accuracy: 0.6311475409836066, Test Accuracy: 0.6928104575163399
Epoch 3 / 30, Train Loss: 0.5841600596904755, Test Loss: 0.5460123479366302,
Train Accuracy: 0.7377049180327869, Test Accuracy: 0.738562091503268
Epoch 4 / 30, Train Loss: 0.5604859106242657, Test Loss: 0.5282542407512665,
Train Accuracy: 0.7540983606557377, Test Accuracy: 0.7712418300653595
Epoch 5 / 30, Train Loss: 0.5212857164442539, Test Loss: 0.49475630521774294,
Train Accuracy: 0.7950819672131147, Test Accuracy: 0.8366013071895425
Epoch 6 / 30, Train Loss: 0.4841243512928486, Test Loss: 0.46025643348693845,
Train Accuracy: 0.8237704918032787, Test Accuracy: 0.8431372549019608
Epoch 7 / 30, Train Loss: 0.44586482644081116, Test Loss: 0.43508976697921753,
Train Accuracy: 0.8524590163934426, Test Accuracy: 0.8627450980392157
Epoch 8 / 30, Train Loss: 0.43780064582824707, Test Loss: 0.41472959518432617,
Train Accuracy: 0.8565573770491803, Test Accuracy: 0.8823529411764706
Epoch 9 / 30, Train Loss: 0.43369603529572487, Test Loss: 0.3919881582260132,
Train Accuracy: 0.8770491803278688, Test Accuracy: 0.8888888888888888
```

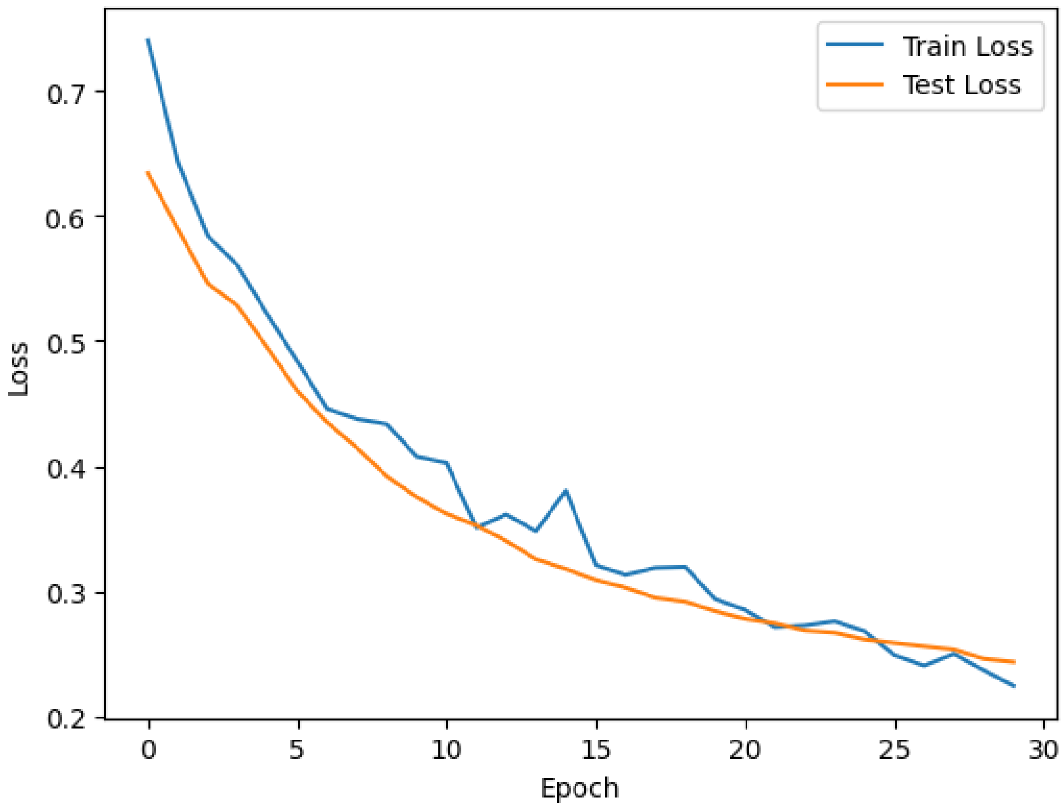
```
Epoch 10 / 30, Train Loss: 0.4077473431825638, Test Loss: 0.3754044145345688,
Train Accuracy: 0.860655737704918, Test Accuracy: 0.8823529411764706
Epoch 11 / 30, Train Loss: 0.40262167155742645, Test Loss: 0.36203463077545167,
Train Accuracy: 0.889344262295082, Test Accuracy: 0.8954248366013072
Epoch 12 / 30, Train Loss: 0.35084324330091476, Test Loss: 0.3527695506811142,
Train Accuracy: 0.9098360655737705, Test Accuracy: 0.8888888888888888
Epoch 13 / 30, Train Loss: 0.3614560030400753, Test Loss: 0.34041716158390045,
Train Accuracy: 0.9139344262295082, Test Accuracy: 0.8954248366013072
Epoch 14 / 30, Train Loss: 0.34818144142627716, Test Loss: 0.3257443279027939,
Train Accuracy: 0.9098360655737705, Test Accuracy: 0.9084967320261438
Epoch 15 / 30, Train Loss: 0.38046708330512047, Test Loss: 0.31786433458328245,
Train Accuracy: 0.8729508196721312, Test Accuracy: 0.9019607843137255
Epoch 16 / 30, Train Loss: 0.320930652320385, Test Loss: 0.3091241180896759,
Train Accuracy: 0.9139344262295082, Test Accuracy: 0.9019607843137255
Epoch 17 / 30, Train Loss: 0.3131910301744938, Test Loss: 0.3030261009931564,
Train Accuracy: 0.9221311475409836, Test Accuracy: 0.9084967320261438
Epoch 18 / 30, Train Loss: 0.31882288306951523, Test Loss: 0.2949344992637634,
Train Accuracy: 0.9098360655737705, Test Accuracy: 0.9150326797385621
Epoch 19 / 30, Train Loss: 0.3195708766579628, Test Loss: 0.29169304966926574,
Train Accuracy: 0.9057377049180327, Test Accuracy: 0.9084967320261438
Epoch 20 / 30, Train Loss: 0.2938222214579582, Test Loss: 0.28420203030109403,
Train Accuracy: 0.9221311475409836, Test Accuracy: 0.9084967320261438
Epoch 21 / 30, Train Loss: 0.2851744070649147, Test Loss: 0.27806246280670166,
Train Accuracy: 0.9180327868852459, Test Accuracy: 0.9150326797385621
Epoch 22 / 30, Train Loss: 0.27146914787590504, Test Loss: 0.2746284455060959,
Train Accuracy: 0.9262295081967213, Test Accuracy: 0.9150326797385621
Epoch 23 / 30, Train Loss: 0.27277664467692375, Test Loss: 0.26869991421699524,
Train Accuracy: 0.9426229508196722, Test Accuracy: 0.9150326797385621
Epoch 24 / 30, Train Loss: 0.2761826105415821, Test Loss: 0.26698081791400907,
Train Accuracy: 0.9262295081967213, Test Accuracy: 0.9215686274509803
Epoch 25 / 30, Train Loss: 0.2681906670331955, Test Loss: 0.2616105377674103,
Train Accuracy: 0.9016393442622951, Test Accuracy: 0.9215686274509803
Epoch 26 / 30, Train Loss: 0.24914774484932423, Test Loss: 0.2588020324707031,
Train Accuracy: 0.9549180327868853, Test Accuracy: 0.9215686274509803
Epoch 27 / 30, Train Loss: 0.24066543392837048, Test Loss: 0.25609440803527833,
Train Accuracy: 0.9426229508196722, Test Accuracy: 0.9150326797385621
Epoch 28 / 30, Train Loss: 0.250135937705636, Test Loss: 0.2535726338624954,
Train Accuracy: 0.930327868852459, Test Accuracy: 0.9215686274509803
Epoch 29 / 30, Train Loss: 0.23681160807609558, Test Loss: 0.24619961082935332,
Train Accuracy: 0.9426229508196722, Test Accuracy: 0.9150326797385621
Epoch 30 / 30, Train Loss: 0.2245215941220522, Test Loss: 0.24365059286355972,
Train Accuracy: 0.9426229508196722, Test Accuracy: 0.9215686274509803
```

[65]:
```
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()
```



[66]:
```
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 92.16%.

3.2 Using ResNet-18 as a feature extractor

[67]:
```
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)).to(device)
for param in resnet_model.parameters():
    param.requires_grad = False

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

[68]:
```
loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(resnet_model.parameters(), lr=0.001)
iterations = 30
```

[69]:
```
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, 10]
--                         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, 10]
5,130                    True
================================================================================
============================================================
Total params: 11,181,642
Trainable params: 5,130
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): 44.73
Estimated Total Size (MB): 1335.66
================================================================================
============================================================
```

[70]:
```
train_accuracy_hist, test_accuracy_hist, train_loss_hist, test_loss_hist =
    train(resnet_model, trainloader_hymenoptera, testloader_hymenoptera,
    iterations, optimizer, loss, device)
```

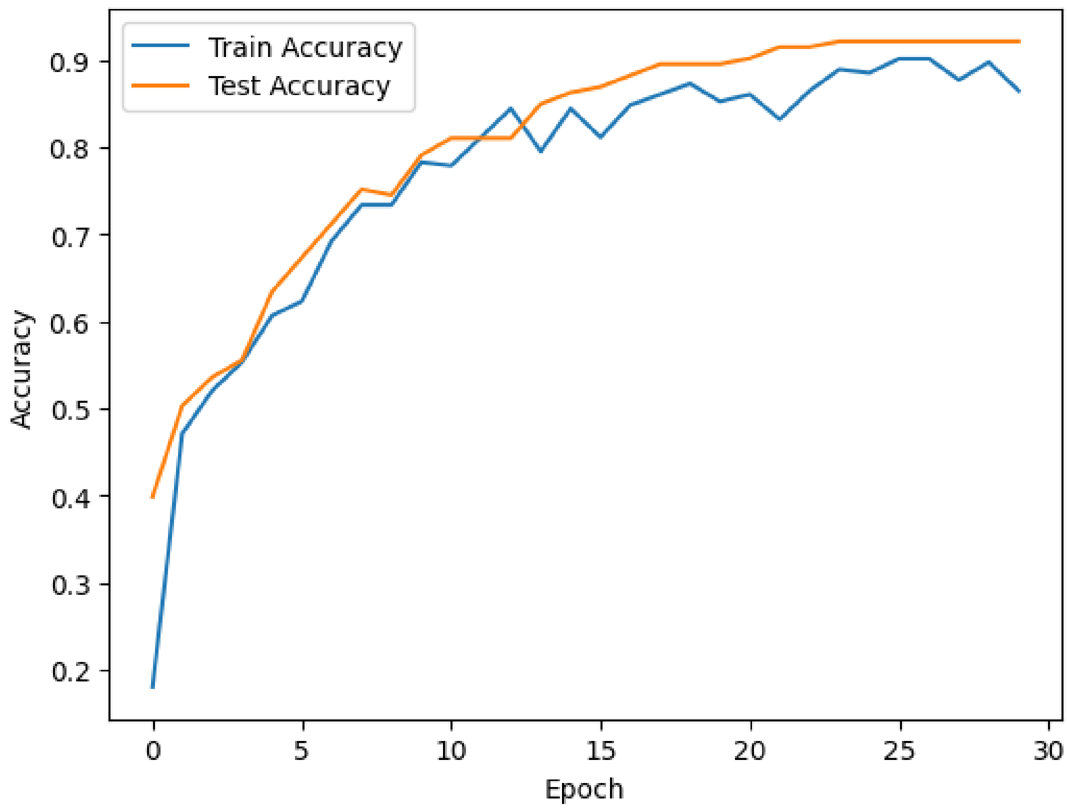
```
Epoch 1 / 30, Train Loss: 2.169361427426338, Test Loss: 1.623906660079956, Train
Accuracy: 0.18032786885245902, Test Accuracy: 0.39869281045751637
Epoch 2 / 30, Train Loss: 1.4038525372743607, Test Loss: 1.1809960842132567,
Train Accuracy: 0.4713114754098361, Test Accuracy: 0.5032679738562091
Epoch 3 / 30, Train Loss: 1.0933245494961739, Test Loss: 0.982301938533783,
Train Accuracy: 0.5204918032786885, Test Accuracy: 0.5359477124183006
Epoch 4 / 30, Train Loss: 0.9442420229315758, Test Loss: 0.8642782926559448,
Train Accuracy: 0.5532786885245902, Test Accuracy: 0.5555555555555556
Epoch 5 / 30, Train Loss: 0.8431264758110046, Test Loss: 0.785391116142273,
```

```
Train Accuracy: 0.6065573770491803, Test Accuracy: 0.6339869281045751
Epoch 6 / 30, Train Loss: 0.8003100976347923, Test Loss: 0.7292196869850158,
Train Accuracy: 0.6229508196721312, Test Accuracy: 0.673202614379085
Epoch 7 / 30, Train Loss: 0.7290625125169754, Test Loss: 0.683362352848053,
Train Accuracy: 0.6926229508196722, Test Accuracy: 0.7124183006535948
Epoch 8 / 30, Train Loss: 0.6734580993652344, Test Loss: 0.6364186406135559,
Train Accuracy: 0.7336065573770492, Test Accuracy: 0.7516339869281046
Epoch 9 / 30, Train Loss: 0.6541433706879616, Test Loss: 0.6063030123710632,
Train Accuracy: 0.7336065573770492, Test Accuracy: 0.7450980392156863
Epoch 10 / 30, Train Loss: 0.6121368557214737, Test Loss: 0.5750733911991119,
Train Accuracy: 0.7827868852459017, Test Accuracy: 0.7908496732026143
Epoch 11 / 30, Train Loss: 0.5844235718250275, Test Loss: 0.5531553149223327,
Train Accuracy: 0.7786885245901639, Test Accuracy: 0.8104575163398693
Epoch 12 / 30, Train Loss: 0.5474961921572685, Test Loss: 0.5237249076366425,
Train Accuracy: 0.8114754098360656, Test Accuracy: 0.8104575163398693
Epoch 13 / 30, Train Loss: 0.5326035730540752, Test Loss: 0.502429085969925,
Train Accuracy: 0.8442622950819673, Test Accuracy: 0.8104575163398693
Epoch 14 / 30, Train Loss: 0.53502157330513, Test Loss: 0.48578929901123047,
Train Accuracy: 0.7950819672131147, Test Accuracy: 0.8496732026143791
Epoch 15 / 30, Train Loss: 0.4917401447892189, Test Loss: 0.471936309337616,
Train Accuracy: 0.8442622950819673, Test Accuracy: 0.8627450980392157
Epoch 16 / 30, Train Loss: 0.4841150566935539, Test Loss: 0.4520291805267334,
Train Accuracy: 0.8114754098360656, Test Accuracy: 0.869281045751634
Epoch 17 / 30, Train Loss: 0.4584265500307083, Test Loss: 0.43817468285560607,
Train Accuracy: 0.8483606557377049, Test Accuracy: 0.8823529411764706
Epoch 18 / 30, Train Loss: 0.46567703410983086, Test Loss: 0.43019127249717715,
Train Accuracy: 0.860655737704918, Test Accuracy: 0.8954248366013072
Epoch 19 / 30, Train Loss: 0.44865116477012634, Test Loss: 0.4137004196643829,
Train Accuracy: 0.8729508196721312, Test Accuracy: 0.8954248366013072
Epoch 20 / 30, Train Loss: 0.4460241571068764, Test Loss: 0.4021461963653564,
Train Accuracy: 0.8524590163934426, Test Accuracy: 0.8954248366013072
Epoch 21 / 30, Train Loss: 0.44250325486063957, Test Loss: 0.3907356262207031,
Train Accuracy: 0.860655737704918, Test Accuracy: 0.9019607843137255
Epoch 22 / 30, Train Loss: 0.4419235624372959, Test Loss: 0.38154500126838686,
Train Accuracy: 0.8319672131147541, Test Accuracy: 0.9150326797385621
Epoch 23 / 30, Train Loss: 0.4109177030622959, Test Loss: 0.3718034625053406,
Train Accuracy: 0.8647540983606558, Test Accuracy: 0.9150326797385621
Epoch 24 / 30, Train Loss: 0.41230393573641777, Test Loss: 0.364398592710495,
Train Accuracy: 0.889344262295082, Test Accuracy: 0.9215686274509803
Epoch 25 / 30, Train Loss: 0.4102812893688679, Test Loss: 0.3571122646331787,
Train Accuracy: 0.8852459016393442, Test Accuracy: 0.9215686274509803
Epoch 26 / 30, Train Loss: 0.37579767778515816, Test Loss: 0.3505669295787811,
Train Accuracy: 0.9016393442622951, Test Accuracy: 0.9215686274509803
Epoch 27 / 30, Train Loss: 0.37975001707673073, Test Loss: 0.3444370269775391,
Train Accuracy: 0.9016393442622951, Test Accuracy: 0.9215686274509803
Epoch 28 / 30, Train Loss: 0.3856111168861389, Test Loss: 0.3392278730869293,
Train Accuracy: 0.8770491803278688, Test Accuracy: 0.9215686274509803
Epoch 29 / 30, Train Loss: 0.36211057752370834, Test Loss: 0.33200012147426605,
```

```
Train Accuracy: 0.8975409836065574, Test Accuracy: 0.9215686274509803
Epoch 30 / 30, Train Loss: 0.37362760677933693, Test Loss: 0.3237911552190781,
Train Accuracy: 0.8647540983606558, Test Accuracy: 0.9215686274509803
```

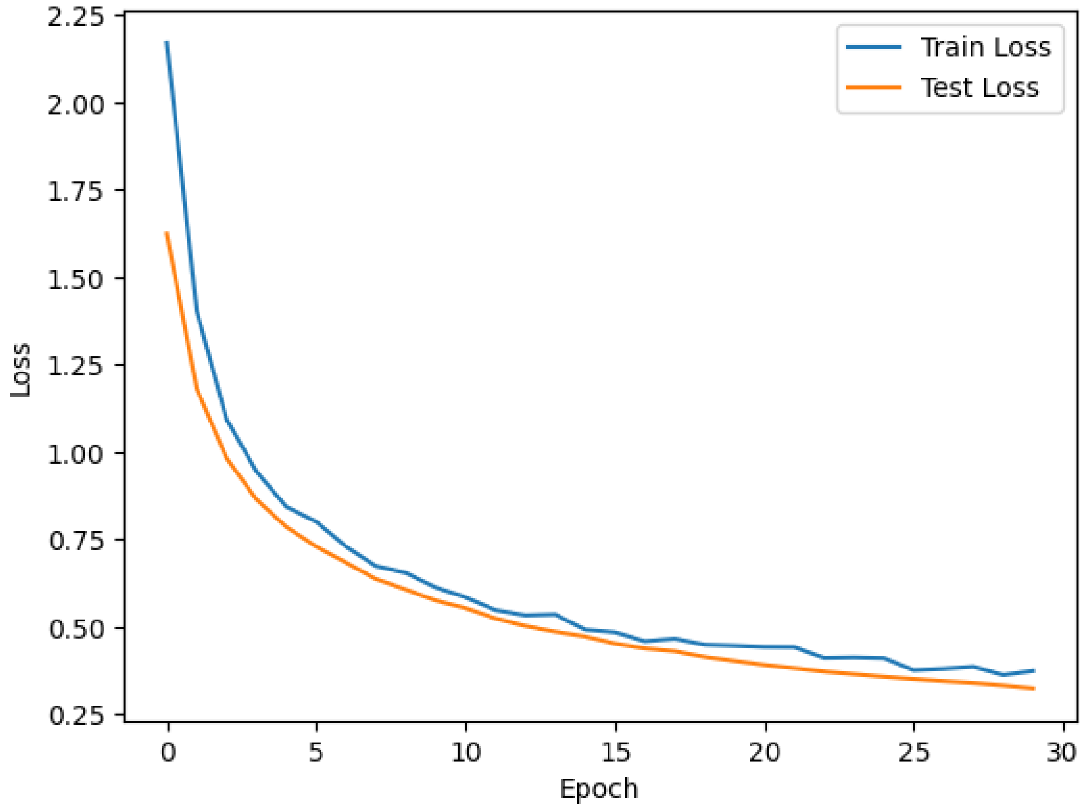
[71]:

```python
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()
```



[72]:

```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()
```



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 92.16% which matches the performance when it was finetuned.