```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torchvision import models, datasets
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
```

```
## Step 1: Load and Preprocess Data
# Define transformations for images
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images for pre-trained model input
    transforms.ToTensor(),
    #transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  # Standard normalization for pre-trained models
])
```

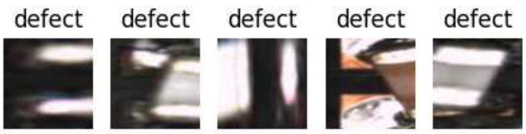
```
!unzip -qq ./chip_data.zip -d data
```

```
replace data/dataset/test/defect/D2_C97.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

```
dataset_path = "./data/dataset/"
train_dataset = datasets.ImageFolder(root=f"{dataset_path}/train", transform=transform)
test_dataset = datasets.ImageFolder(root=f"{dataset_path}/test", transform=transform)
```

```
def show_sample_images(dataset, num_images=5):
    fig, axes = plt.subplots(1, num_images, figsize=(5, 5))
    for i in range(num_images):
        image, label = dataset[i]
        image = image.permute(1, 2, 0)  # Convert tensor format (C, H, W) to (H, W, C)
        axes[i].imshow(image)
        axes[i].set_title(dataset.classes[label])
        axes[i].axis("off")
    plt.show()
```

```
show_sample_images(train_dataset)
```



```
print(f"Total number of training samples: {len(train_dataset)}")

# Get the shape of the first image in the dataset
first_image, label = train_dataset[0]
print(f"Shape of the first image: {first_image.shape}")
```

```
Total number of training samples: 172
Shape of the first image: torch.Size([3, 224, 224])
```

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
## Step 2: Load Pretrained Model and Modify for Transfer Learning
# Load a pre-trained VGG19 model
from torchvision.models import VGG19_Weights
model = models.vgg19(weights=models.VGG19_Weights.DEFAULT)
```

```
Downloading: "https://download.pytorch.org/models/vgg19-dcbb9e9d.pth" to /root/.cache/torch/hub/checkpoints/vgg19-dcbb9e9d.pth
100%|██████████| 548M/548M [00:06<00:00, 91.9MB/s]
```

```
from torchsummary import summary
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
# Print model summary
summary(model, input_size=(3, 224, 224), device=str(device))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
           Conv2d-17          [-1, 256, 56, 56]         590,080
             ReLU-18          [-1, 256, 56, 56]               0
        MaxPool2d-19          [-1, 256, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       1,180,160
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
           Conv2d-24          [-1, 512, 28, 28]       2,359,808
             ReLU-25          [-1, 512, 28, 28]               0
           Conv2d-26          [-1, 512, 28, 28]       2,359,808
             ReLU-27          [-1, 512, 28, 28]               0
        MaxPool2d-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
           Conv2d-31          [-1, 512, 14, 14]       2,359,808
             ReLU-32          [-1, 512, 14, 14]               0
           Conv2d-33          [-1, 512, 14, 14]       2,359,808
             ReLU-34          [-1, 512, 14, 14]               0
           Conv2d-35          [-1, 512, 14, 14]       2,359,808
             ReLU-36          [-1, 512, 14, 14]               0
        MaxPool2d-37            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-38            [-1, 512, 7, 7]               0
           Linear-39                 [-1, 4096]     102,764,544
             ReLU-40                 [-1, 4096]               0
          Dropout-41                 [-1, 4096]               0
           Linear-42                 [-1, 4096]      16,781,312
             ReLU-43                 [-1, 4096]               0
          Dropout-44                 [-1, 4096]               0
           Linear-45                 [-1, 1000]       4,097,000
================================================================
Total params: 143,667,240
Trainable params: 143,667,240
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 238.69
Params size (MB): 548.05
Estimated Total Size (MB): 787.31
----------------------------------------------------------------
```

```
# Modify the final fully connected layer to match the dataset classes
num_ftrs = model.classifier[-1].in_features
model.classifier[-1] = nn.Linear(num_ftrs, 1)
```

```
# Move model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
```

```
summary(model, input_size=(3, 224, 224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
           Conv2d-17          [-1, 256, 56, 56]         590,080
             ReLU-18          [-1, 256, 56, 56]               0
        MaxPool2d-19          [-1, 256, 28, 28]               0
```

```
           Conv2d-20          [-1, 512, 28, 28]       1,180,160
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
           Conv2d-24          [-1, 512, 28, 28]       2,359,808
             ReLU-25          [-1, 512, 28, 28]               0
           Conv2d-26          [-1, 512, 28, 28]       2,359,808
             ReLU-27          [-1, 512, 28, 28]               0
        MaxPool2d-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
           Conv2d-31          [-1, 512, 14, 14]       2,359,808
             ReLU-32          [-1, 512, 14, 14]               0
           Conv2d-33          [-1, 512, 14, 14]       2,359,808
             ReLU-34          [-1, 512, 14, 14]               0
           Conv2d-35          [-1, 512, 14, 14]       2,359,808
             ReLU-36          [-1, 512, 14, 14]               0
        MaxPool2d-37            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-38            [-1, 512, 7, 7]               0
           Linear-39                 [-1, 4096]     102,764,544
             ReLU-40                 [-1, 4096]               0
          Dropout-41                 [-1, 4096]               0
           Linear-42                 [-1, 4096]      16,781,312
             ReLU-43                 [-1, 4096]               0
          Dropout-44                 [-1, 4096]               0
           Linear-45                    [-1, 1]           4,097
================================================================
Total params: 139,574,337
Trainable params: 139,574,337
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 238.68
Params size (MB): 532.43
Estimated Total Size (MB): 771.69
----------------------------------------------------------------
```

```
# Freeze all layers except the final layer
for param in model.features.parameters():
    param.requires_grad = False  # Freeze feature extractor layers
```

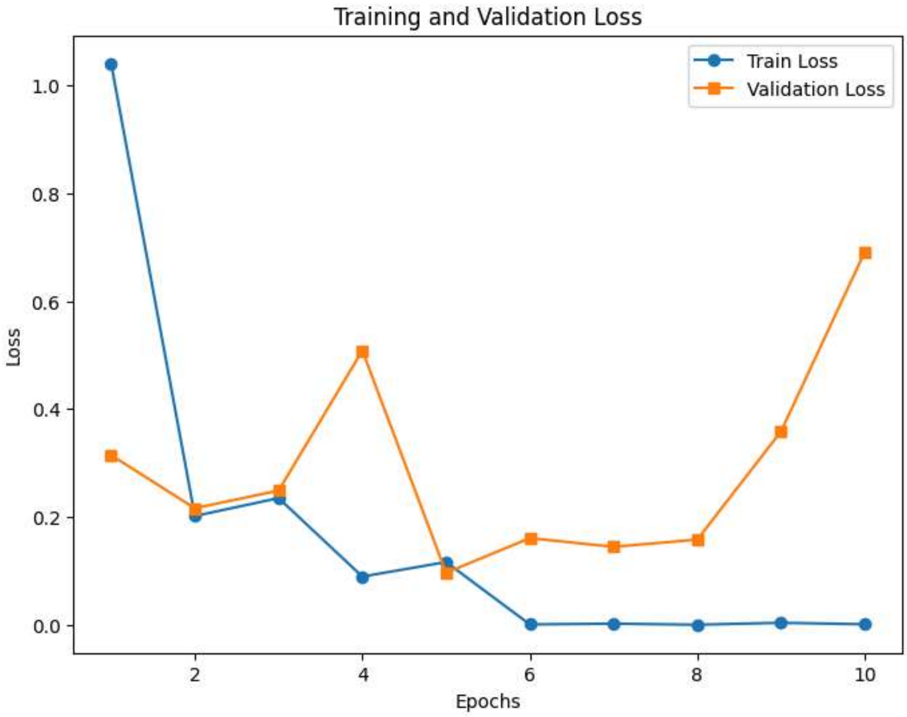
```
# Include the Loss function and optimizer
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
## Step 3: Train the Model
def train_model(model, train_loader,test_loader,num_epochs=10):
    train_losses = []
    val_losses = []
    model.train()
    for epoch in range(num_epochs):
        running_loss = 0.0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            labels = labels.unsqueeze(1).float() # Reshape labels to match output size
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        train_losses.append(running_loss / len(train_loader))

        # Compute validation loss
        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
                labels = labels.unsqueeze(1).float() # Reshape labels to match output size
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

        val_losses.append(val_loss / len(test_loader))
        model.train()

        print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_losses[-1]:.4f}, Validation Loss: {val_losses[-1]:.4f}')

    # Plot training and validation loss
    print("Name:A.LAHARI")
    print("Register Number:212223230111")
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss', marker='o')
    plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss', marker='s')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')
    plt.legend()
    plt.show()
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
```

```
train_model(model,train_loader,test_loader)
```

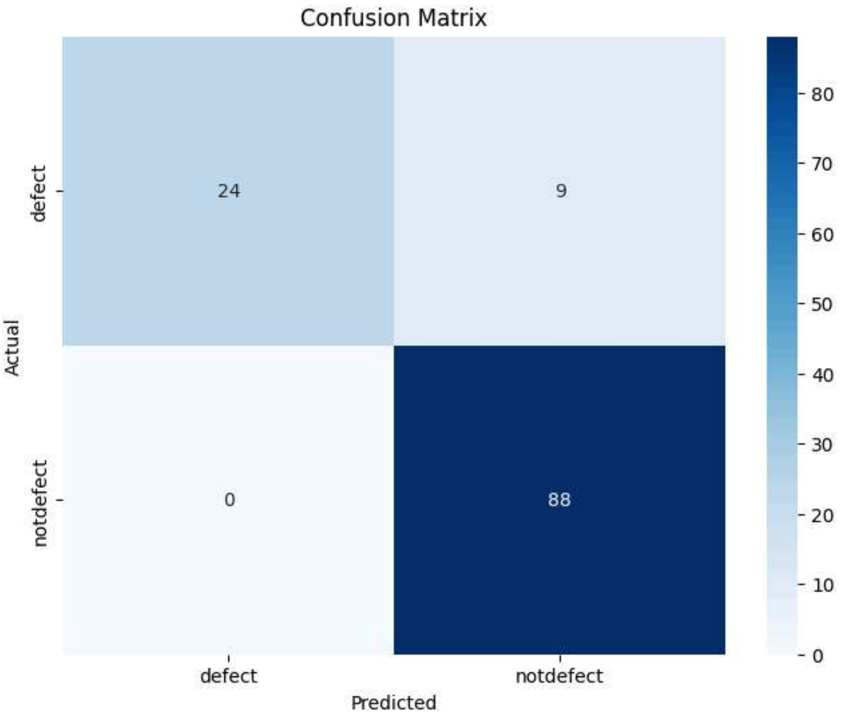
```
Epoch [1/10], Train Loss: 1.0404, Validation Loss: 0.3144
Epoch [2/10], Train Loss: 0.2016, Validation Loss: 0.2160
Epoch [3/10], Train Loss: 0.2348, Validation Loss: 0.2488
Epoch [4/10], Train Loss: 0.0893, Validation Loss: 0.5082
Epoch [5/10], Train Loss: 0.1163, Validation Loss: 0.0963
Epoch [6/10], Train Loss: 0.0006, Validation Loss: 0.1606
Epoch [7/10], Train Loss: 0.0020, Validation Loss: 0.1444
Epoch [8/10], Train Loss: 0.0001, Validation Loss: 0.1579
Epoch [9/10], Train Loss: 0.0037, Validation Loss: 0.3585
Epoch [10/10], Train Loss: 0.0009, Validation Loss: 0.6913
Name:A.LAHARI
Register Number:212223230111
```


Training and Validation Loss

```
def test_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            probs = torch.sigmoid(outputs)
            predicted = (probs > 0.5).int()
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    accuracy = correct / total
    print(f'Test Accuracy: {accuracy:.4f}')

    # Compute confusion matrix
    cm = confusion_matrix(all_labels, all_preds)
    print("Name : A.LAHARI")
    print("Register Number: 212223230111")
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=train_dataset.classes, yticklabels=train_dataset.classes)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

    # Print classification report
    print("Name : A.LAHARI")
    print("Register Number: 212223230111")
    print("Classification Report:")
    print(classification_report(all_labels, all_preds, target_names=train_dataset.classes))
```

```
test_model(model, test_loader)
```

```
Test Accuracy: 27.6612
Name : A.LAHARI
Register Number: 212223230111
```



```
Name : A.LAHARI
Register Number: 212223230111
Classification Report:
              precision    recall  f1-score   support

      defect       1.00      0.73      0.84        33
   notdefect       0.91      1.00      0.95        88

    accuracy                           0.93       121
   macro avg       0.95      0.86      0.90       121
weighted avg       0.93      0.93      0.92       121
```

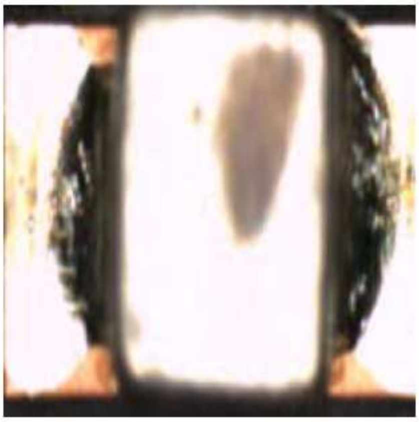
```
def predict_image(model, image_index, dataset):
    model.eval()
    image, label = dataset[image_index]
    with torch.no_grad():
        image_tensor = image.unsqueeze(0).to(device)
        output = model(image_tensor)

        # Apply sigmoid to get probability, threshold at 0.5
        prob = torch.sigmoid(output)
        predicted = (prob > 0.5).int().item()


    class_names = class_names = dataset.classes
    # Display the image
    image_to_display = transforms.ToPILImage()(image)
    plt.figure(figsize=(4, 4))
    plt.imshow(image_to_display)
    plt.title(f'Actual: {class_names[label]}\nPredicted: {class_names[predicted]}')
    plt.axis("off")
    plt.show()

    print(f'Actual: {class_names[label]}, Predicted: {class_names[predicted]}')
```

```
predict_image(model, image_index=46, dataset=test_dataset)
```

Actual: notdefect
Predicted: notdefect

```
Actual: notdefect, Predicted: notdefect
```

```
predict_image(model, image_index=9, dataset=test_dataset)
```



Actual: defect
Predicted: defect

```
Actual: defect, Predicted: defect
```