Lego defect detection

This notebook trains and evaluates a ResNet18 pretrained classifier on Lego brick images organized into:

```
- 'defect' and 'no_defect' folders for both train and
valid splits.
```

For computational constrains, images will be resized to (224, 224) and no augmentation techniques will be applied.

In [6]:
```
# import libraries
import numpy as np
import os
import shutil
import random
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torchvision.models import resnet18, ResNet18_Weights
from torch.utils.data import DataLoader
from sklearn.metrics import precision_score, confusion_matrix, classifica
```

In [7]:
```
# 1. Define parameters
data_dir = './'
train_dir = os.path.join(data_dir, 'train')
valid_dir = os.path.join(data_dir, 'valid')
num_classes = 2  # 'defect' and 'no_defect'
batch_size = 16
num_epochs = 15
learning_rate = 0.001

# Use GPU if available, otherwise use CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Set a fixed random seed for reproducibility
seed = 42
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

In [8]:
```
# 2. Data transforms
# Images are resized to 224x224 and converted to tensors
transform = transforms.Compose([
    transforms.transforms.Resize((224, 224)),
    transforms.ToTensor(),
```

```
])

# 3. Define datasets and loaders
# ImageFolder expects data in class subfolders (defect/no_defect)
train_dataset = datasets.ImageFolder(train_dir, transform=transform)
valid_dataset = datasets.ImageFolder(valid_dir, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=T
valid_loader = DataLoader(valid_dataset, batch_size=batch_size, shuffle=F

print(f"Classes: {train_dataset.classes}")
print(train_dataset.class_to_idx)
```

```
Classes: ['defect', 'no_defect']
{'defect': 0, 'no_defect': 1}
```

In [9]:
```
# 4. Load pretrained ResNet and freeze backbone
# We use a ResNet18 pretrained on ImageNet

model = models.resnet18(weights=ResNet18_Weights.DEFAULT)

# Freeze all layers so only the final layer is trained
for param in model.parameters():
    param.requires_grad = False

# Replace the final fully connected layer to match our number of classes
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)
```

In [10]:
```
# 5. Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=learning_rate)
```

In [11]:
```
# 6. Training loop
print("Starting training...")
train_losses = []
valid_losses = []

num_epochs = 20

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)
    epoch_loss = running_loss / len(train_loader.dataset)
    train_losses.append(epoch_loss)

    # --- Validation loss ---
    model.eval()
    val_running_loss = 0.0
    with torch.no_grad():
```

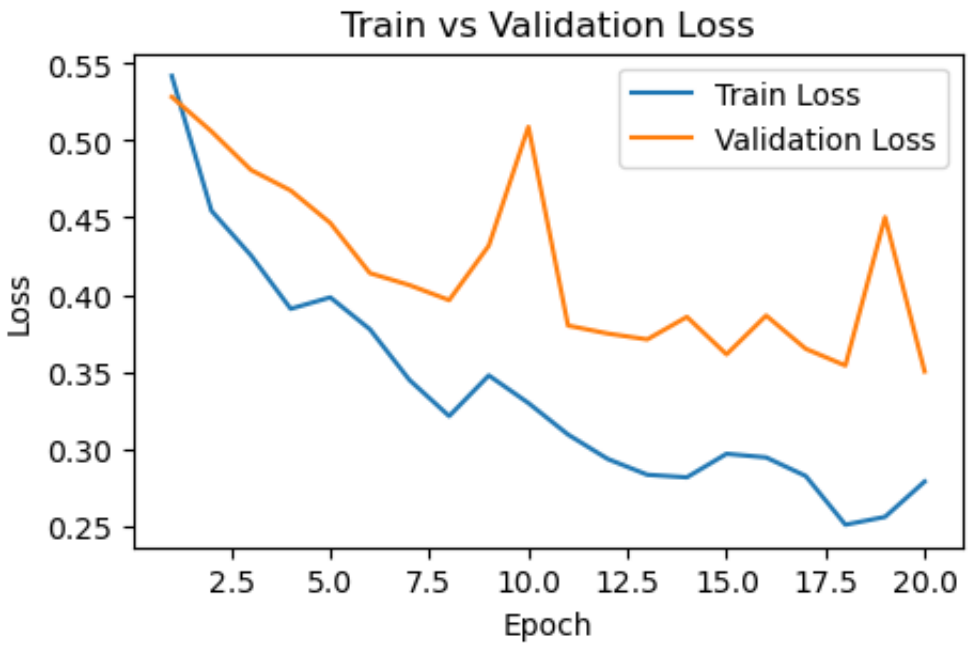
```
        for images, labels in valid_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            val_loss = criterion(outputs, labels)
            val_running_loss += val_loss.item() * images.size(0)
    val_epoch_loss = val_running_loss / len(valid_loader.dataset)
    valid_losses.append(val_epoch_loss)

    print(f"Epoch {epoch+1}/{num_epochs}, loss: {epoch_loss:.4f}, Val_los
```

```
Starting training...
Epoch 1/20, loss: 0.5420, Val_loss: 0.5284
Epoch 2/20, loss: 0.4544, Val_loss: 0.5060
Epoch 3/20, loss: 0.4254, Val_loss: 0.4809
Epoch 4/20, loss: 0.3907, Val_loss: 0.4676
Epoch 5/20, loss: 0.3982, Val_loss: 0.4465
Epoch 6/20, loss: 0.3774, Val_loss: 0.4139
Epoch 7/20, loss: 0.3444, Val_loss: 0.4061
Epoch 8/20, loss: 0.3211, Val_loss: 0.3964
Epoch 9/20, loss: 0.3475, Val_loss: 0.4317
Epoch 10/20, loss: 0.3295, Val_loss: 0.5089
Epoch 11/20, loss: 0.3091, Val_loss: 0.3798
Epoch 12/20, loss: 0.2932, Val_loss: 0.3747
Epoch 13/20, loss: 0.2829, Val_loss: 0.3709
Epoch 14/20, loss: 0.2813, Val_loss: 0.3855
Epoch 15/20, loss: 0.2966, Val_loss: 0.3611
Epoch 16/20, loss: 0.2942, Val_loss: 0.3863
Epoch 17/20, loss: 0.2821, Val_loss: 0.3647
Epoch 18/20, loss: 0.2505, Val_loss: 0.3539
Epoch 19/20, loss: 0.2556, Val_loss: 0.4503
Epoch 20/20, loss: 0.2785, Val_loss: 0.3500
```

In [12]:

```
# Plot the loss curves
plt.figure(figsize=(5,3))
plt.plot(range(1, num_epochs+1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs+1), valid_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Train vs Validation Loss')
plt.legend()
plt.show()
```



In [13]:
```python
# 7. Evaluation
# We evaluate the model using precision (for the 'defect' class)
model.eval()
all_labels = []
all_preds = []
with torch.no_grad():
    for images, labels in valid_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        all_labels.extend(labels.cpu().numpy())
        all_preds.extend(predicted.cpu().numpy())

# Compute precision for the 'defect' class (in our case is class 0)
precision = precision_score(all_labels, all_preds, pos_label=0, zero_divi
print(f"Validation Precision (defect class): {precision:.4f}")

# Compute and display the confusion matrix
cm = confusion_matrix(all_labels, all_preds)
print("Confusion Matrix:")
print(cm)
```

```
Validation Precision (defect class): 0.8739
Confusion Matrix:
[[208   8]
 [ 30  54]]
```

In [14]:
```python
class_report = classification_report(all_labels, all_preds)
print(class_report)
```

	precision	recall	f1-score	support
0	0.87	0.96	0.92	216
1	0.87	0.64	0.74	84
accuracy			0.87	300
macro avg	0.87	0.80	0.83	300
weighted avg	0.87	0.87	0.87	300

The more data, the better performance

In [16]:
```python
# Create new full_train folders
os.makedirs('full_train/defect', exist_ok=True)
os.makedirs('full_train/no_defect', exist_ok=True)

# Copy train images
for cls in ['defect', 'no_defect']:
    for fname in os.listdir(f'train/{cls}'):
        shutil.copy2(f'train/{cls}/{fname}', f'full_train/{cls}/{fname}')
# Copy valid images
for cls in ['defect', 'no_defect']:
    for fname in os.listdir(f'valid/{cls}'):
        shutil.copy2(f'valid/{cls}/{fname}', f'full_train/{cls}/{fname}')

# Create dataset and loader for full_train
full_train_dataset = datasets.ImageFolder('full_train', transform=transfo
full_train_loader = DataLoader(full_train_dataset, batch_size=batch_size,

# Re-initialize the model (for a fresh start)
model = models.resnet18(weights=ResNet18_Weights.DEFAULT)
for param in model.parameters():
    param.requires_grad = False
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=learning_rate)

print("Starting final training on all data...")
final_train_losses = []
num_epochs_full = 10

for epoch in range(num_epochs_full):
    model.train()
    running_loss = 0.0
    for images, labels in full_train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)
    epoch_loss = running_loss / len(full_train_loader.dataset)
```

```
    final_train_losses.append(epoch_loss)
    print(f"Epoch {epoch+1}/{num_epochs_full}, loss: {epoch_loss:.4f}")

# Save the final model
torch.save(model.state_dict(), 'lego_resnet18_full.pth')
print("Full-data-trained model weights saved as lego_resnet18_full.pth")
```

```
Starting final training on all data...
Epoch 1/10, loss: 0.5632
Epoch 2/10, loss: 0.4569
Epoch 3/10, loss: 0.4231
Epoch 4/10, loss: 0.4082
Epoch 5/10, loss: 0.3648
Epoch 6/10, loss: 0.3713
Epoch 7/10, loss: 0.3443
Epoch 8/10, loss: 0.3417
Epoch 9/10, loss: 0.3371
Epoch 10/10, loss: 0.3105
Full-data-trained model weights saved as lego_resnet18_full.pth
```

In [17]:
```
# save the models weights
torch.save(model.state_dict(), 'lego_resnet18_full.pth')
```

In [18]:
```
# Recreate the model architecture
model = models.resnet18(weights=ResNet18_Weights.DEFAULT)
for param in model.parameters():
    param.requires_grad = False
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)
```

Test set evaluation

In [20]:
```
test_dir = os.path.join(data_dir, 'test')
test_dataset = datasets.ImageFolder(test_dir, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=Fal

# Load the saved weights
model.load_state_dict(torch.load('lego_resnet18_full.pth', map_location=d

# Now we can run predictions on your test set
model.eval()
test_labels = []
test_preds = []
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        test_labels.extend(labels.cpu().numpy())
        test_preds.extend(predicted.cpu().numpy())

test_precision = precision_score(test_labels, test_preds, pos_label=0, ze
print(f"Test Precision (defect class): {test_precision:.4f}")
test_cm = confusion_matrix(test_labels, test_preds)
print('Test Confusion Matrix:')
```

```
print(test_cm)
```

```
Test Precision (defect class): 0.8926
Test Confusion Matrix:
[[108   5]
 [ 13  29]]
```

In [21]:
```
class_report = classification_report(test_labels, test_preds, target_name
print(class_report)
```

```
              precision    recall  f1-score   support

      defect       0.89      0.96      0.92       113
   no_defect       0.85      0.69      0.76        42

    accuracy                           0.88       155
   macro avg       0.87      0.82      0.84       155
weighted avg       0.88      0.88      0.88       155
```

In [ ]:
```

```