03_textile_defect_training_balanced

March 27, 2025

1 Textile Defect Detection CNN

1.1 1. Environment Setup

[7]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, datasets, models
from torch.utils.data import DataLoader
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
↪auc, precision_recall_curve, average_precision_score
import seaborn as sns
import time
import os
from pathlib import Path
```

1.1.1 Check CUDA availability

[4]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
    print(f"CUDA version: {torch.version.cuda}")
```

```
Using device: cuda
GPU: NVIDIA GeForce RTX 3060 Laptop GPU
CUDA version: 11.8
```

1.2 2. Data Preparation

[8]:
```
PROJECT_ROOT = Path.cwd().parent  # Goes up from notebooks/ to project root
print(f"Project root: {PROJECT_ROOT}")

# Set up paths relative to project root
DATA_ROOT = PROJECT_ROOT / "data"
```

```
TRAIN_DIR = DATA_ROOT / "split" / "train"
TEST_DIR = DATA_ROOT / "split" / "test"

BATCH_SIZE = 32
IMAGE_SIZE = (256, 256)  # Adjusted for textile defect detection
```

```
Project root: c:\Users\Marcony1\OneDrive - Fundacion Universidad de las Americas
Puebla\Documents\Git\textile-image-defect-detector
```

[ ]:
```
# Data transforms with fabric-specific augmentations
train_transform = transforms.Compose([
    transforms.Resize(IMAGE_SIZE),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize(IMAGE_SIZE),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

[10]:
```
# Create datasets
train_dataset = datasets.ImageFolder(TRAIN_DIR, transform=train_transform)
test_dataset = datasets.ImageFolder(TEST_DIR, transform=test_transform)
```

[11]:
```
# Class names
class_names = train_dataset.classes
print(f"Classes: {class_names}")
```

```
Classes: ['NO_OK', 'OK']
```

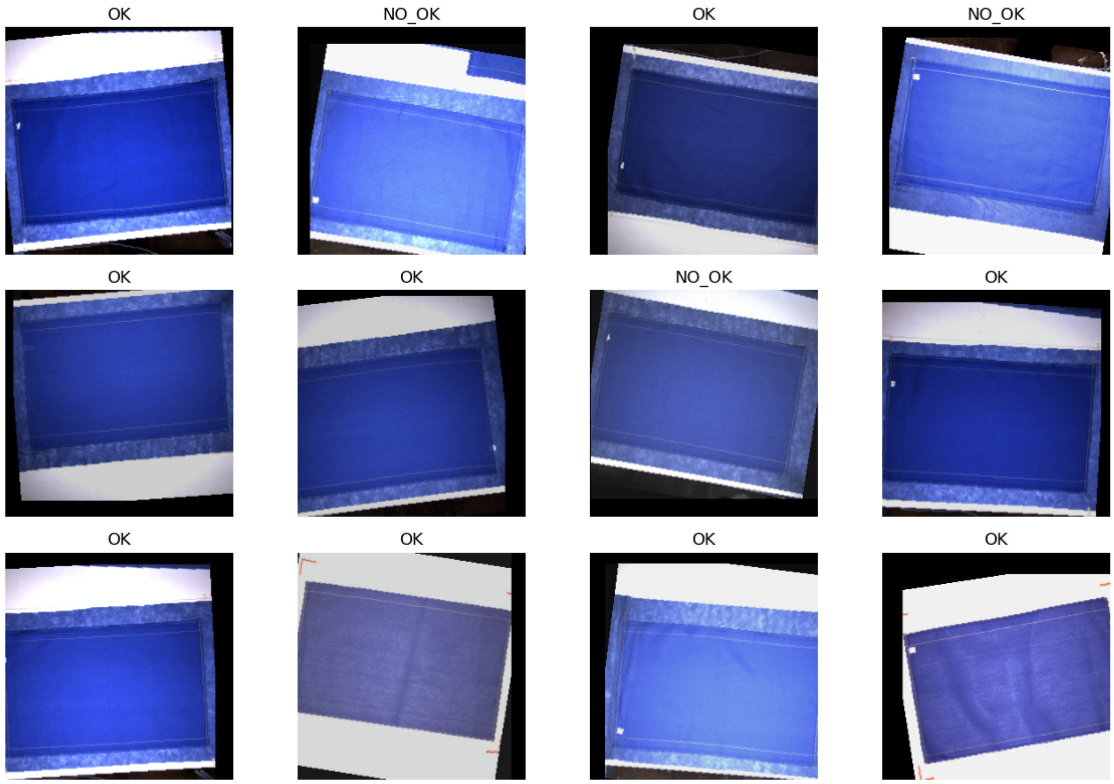
[12]:
```
# Calculate class weights for imbalance handling
from sklearn.utils.class_weight import compute_class_weight

train_targets = [label for _, label in train_dataset]
class_weights = compute_class_weight('balanced', classes=np.
 ↪unique(train_targets), y=train_targets)
class_weights = torch.tensor(class_weights, dtype=torch.float).to(device)
print(f"Class weights: {class_weights}")
```

```
Class weights: tensor([1.9085, 0.6775], device='cuda:0')
```

[13]:
```python
# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=4, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=4, pin_memory=True)
```

[14]:
```python
# Show batch function
def show_batch(images, labels):
    plt.figure(figsize=(12, 8))
    images = images.cpu().numpy().transpose((0, 2, 3, 1))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    images = std * images + mean
    images = np.clip(images, 0, 1)

    for i in range(min(12, len(images))):
        plt.subplot(3, 4, i+1)
        plt.imshow(images[i])
        plt.title(class_names[labels[i]])
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

[15]:
```python
# Display sample
images, labels = next(iter(train_loader))
show_batch(images, labels)
```



1.3 3. CNN Model Definition

[17]:
```
class ImprovedTextileDefectCNN(nn.Module):
    def __init__(self, num_classes=2):
        super(ImprovedTextileDefectCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(128, 256, kernel_size=3, padding=1),
```

```
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(256, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        self.attention = nn.Sequential(
            nn.Conv2d(512, 256, kernel_size=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 1, kernel_size=1),
            nn.Sigmoid()
        )

        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d((1, 1)),
            nn.Flatten(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        attention_mask = self.attention(x)
        x = x * attention_mask
        x = self.classifier(x)
        return x
```

[18]:
```
model = ImprovedTextileDefectCNN(num_classes=2).to(device)
```

[19]:
```
# Print summary
print(model)
```

```
ImprovedTextileDefectCNN(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (14): ReLU()
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (16): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (18): ReLU()
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (attention): Sequential(
    (0): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(256, 1, kernel_size=(1, 1), stride=(1, 1))
    (4): Sigmoid()
  )
  (classifier): Sequential(
    (0): AdaptiveAvgPool2d(output_size=(1, 1))
    (1): Flatten(start_dim=1, end_dim=-1)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): ReLU()
    (4): Dropout(p=0.5, inplace=False)
    (5): Linear(in_features=256, out_features=2, bias=True)
  )
)
```

1.4 4. Training Setup

[20]:
```
# Loss and optimizer
class FocalLoss(nn.Module):
    def __init__(self, alpha=None, gamma=2, reduction='mean'):
```

```
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.reduction = reduction

    def forward(self, inputs, targets):
        ce_loss = nn.functional.cross_entropy(inputs, targets, reduction='none')
        pt = torch.exp(-ce_loss)
        focal_loss = (1 - pt) ** self.gamma * ce_loss

        if self.alpha is not None:
            focal_loss = self.alpha[targets] * focal_loss

        if self.reduction == 'mean':
            return focal_loss.mean()
        elif self.reduction == 'sum':
            return focal_loss.sum()
        return focal_loss

criterion = FocalLoss(alpha=class_weights, gamma=2)
optimizer = optim.AdamW(model.parameters(), lr=0.001, weight_decay=1e-4)
scheduler = optim.lr_scheduler.OneCycleLR(optimizer, max_lr=0.01,
                                        steps_per_epoch=len(train_loader),
                                        epochs=20)
```

[ ]:
```
# Training function
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()
    best_f1 = 0.0

    train_loss_history = []
    train_acc_history = []
    val_loss_history = []
    val_acc_history = []
    val_f1_history = []

    for epoch in range(num_epochs):
        print(f'Epoch {epoch+1}/{num_epochs}')
        print('-' * 10)

        # Training phase
        model.train()
        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in train_loader:
            inputs = inputs.to(device)
```

```
            labels = labels.to(device)

            optimizer.zero_grad()

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            loss = criterion(outputs, labels)

            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
            optimizer.step()
            scheduler.step()

            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        epoch_loss = running_loss / len(train_dataset)
        epoch_acc = running_corrects.double() / len(train_dataset)

        train_loss_history.append(epoch_loss)
        train_acc_history.append(epoch_acc)

        print(f'Train Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

        # Validation phase
        model.eval()
        running_loss = 0.0
        running_corrects = 0
        all_preds = []
        all_labels = []

        with torch.no_grad():
            for inputs, labels in test_loader:
                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

        epoch_loss = running_loss / len(test_dataset)
        epoch_acc = running_corrects.double() / len(test_dataset)
```

```
        # Calculate F1 score for NOT_OK class
        from sklearn.metrics import f1_score
        not_ok_f1 = f1_score(all_labels, all_preds, pos_label=0)  # Assuming␣
↪NOT_OK is class 0

        val_loss_history.append(epoch_loss)
        val_acc_history.append(epoch_acc)
        val_f1_history.append(not_ok_f1)

        print(f'Val Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f} NOT_OK F1:␣
↪{not_ok_f1:.4f}')
        print()

        # Save best model based on NOT_OK F1 score
        if not_ok_f1 > best_f1:
            best_f1 = not_ok_f1
            torch.save(model.state_dict(), '../models/best_model_balanced.pt')
            print(f'New best model saved with NOT_OK F1: {best_f1:.4f}')

    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.
↪0f}s')
    print(f'Best NOT_OK F1: {best_f1:.4f}')

    return model, train_loss_history, train_acc_history, val_loss_history,␣
↪val_acc_history, val_f1_history
```

1.5 5. Model Training

[22]:
```
# Train the model
model, train_loss, train_acc, val_loss, val_acc, val_f1 = train_model(
    model, criterion, optimizer, scheduler, num_epochs=20
)
```

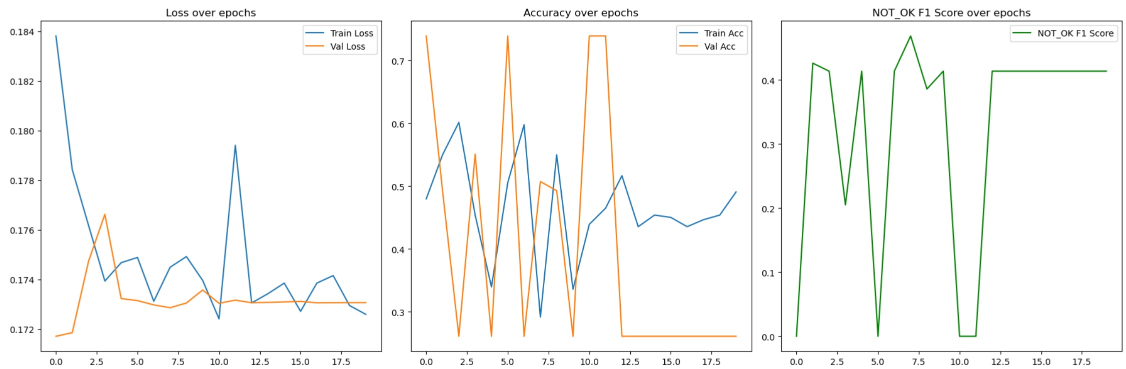
```
Epoch 1/20
----------
Train Loss: 0.1838 Acc: 0.4797
Val Loss: 0.1717 Acc: 0.7391 NOT_OK F1: 0.0000

Epoch 2/20
----------
Train Loss: 0.1784 Acc: 0.5498
Val Loss: 0.1718 Acc: 0.4928 NOT_OK F1: 0.4262

New best model saved with NOT_OK F1: 0.4262
Epoch 3/20
```

```
----------
Train Loss: 0.1762 Acc: 0.6015
Val Loss: 0.1747 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 4/20
----------
Train Loss: 0.1739 Acc: 0.4539
Val Loss: 0.1766 Acc: 0.5507 NOT_OK F1: 0.2051

Epoch 5/20
----------
Train Loss: 0.1747 Acc: 0.3395
Val Loss: 0.1732 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 6/20
----------
Train Loss: 0.1749 Acc: 0.5055
Val Loss: 0.1731 Acc: 0.7391 NOT_OK F1: 0.0000

Epoch 7/20
----------
Train Loss: 0.1731 Acc: 0.5978
Val Loss: 0.1730 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 8/20
----------
Train Loss: 0.1745 Acc: 0.2915
Val Loss: 0.1729 Acc: 0.5072 NOT_OK F1: 0.4688

New best model saved with NOT_OK F1: 0.4688
Epoch 9/20
----------
Train Loss: 0.1749 Acc: 0.5498
Val Loss: 0.1730 Acc: 0.4928 NOT_OK F1: 0.3860

Epoch 10/20
----------
Train Loss: 0.1740 Acc: 0.3358
Val Loss: 0.1736 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 11/20
----------
Train Loss: 0.1724 Acc: 0.4391
Val Loss: 0.1730 Acc: 0.7391 NOT_OK F1: 0.0000

Epoch 12/20
----------
Train Loss: 0.1794 Acc: 0.4649
```

```
Val Loss: 0.1732 Acc: 0.7391 NOT_OK F1: 0.0000

Epoch 13/20
----------
Train Loss: 0.1730 Acc: 0.5166
Val Loss: 0.1731 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 14/20
----------
Train Loss: 0.1734 Acc: 0.4354
Val Loss: 0.1731 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 15/20
----------
Train Loss: 0.1738 Acc: 0.4539
Val Loss: 0.1731 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 16/20
----------
Train Loss: 0.1727 Acc: 0.4502
Val Loss: 0.1731 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 17/20
----------
Train Loss: 0.1738 Acc: 0.4354
Val Loss: 0.1731 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 18/20
----------
Train Loss: 0.1741 Acc: 0.4465
Val Loss: 0.1731 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 19/20
----------
Train Loss: 0.1729 Acc: 0.4539
Val Loss: 0.1731 Acc: 0.2609 NOT_OK F1: 0.4138

Epoch 20/20
----------
Train Loss: 0.1726 Acc: 0.4908
Val Loss: 0.1731 Acc: 0.2609 NOT_OK F1: 0.4138

Training complete in 6m 50s
Best NOT_OK F1: 0.4688
```

1.6 6. Training Visualization

[23]:
```python
# Plot training history
plt.figure(figsize=(18, 6))

plt.subplot(1, 3, 1)
plt.plot(train_loss, label='Train Loss')
plt.plot(val_loss, label='Val Loss')
plt.title('Loss over epochs')
plt.legend()

plt.subplot(1, 3, 2)
plt.plot([x.cpu().numpy() for x in train_acc], label='Train Acc')
plt.plot([x.cpu().numpy() for x in val_acc], label='Val Acc')
plt.title('Accuracy over epochs')
plt.legend()

plt.subplot(1, 3, 3)
plt.plot(val_f1, label='NOT_OK F1 Score', color='green')
plt.title('NOT_OK F1 Score over epochs')
plt.legend()

plt.tight_layout()
plt.show()
```



1.7 7. Model Evaluation

[24]:
```python
# Load best model
model.load_state_dict(torch.load('../models/best_model_balanced.pt'))
model.eval()
```

[24]:
```
ImprovedTextileDefectCNN(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

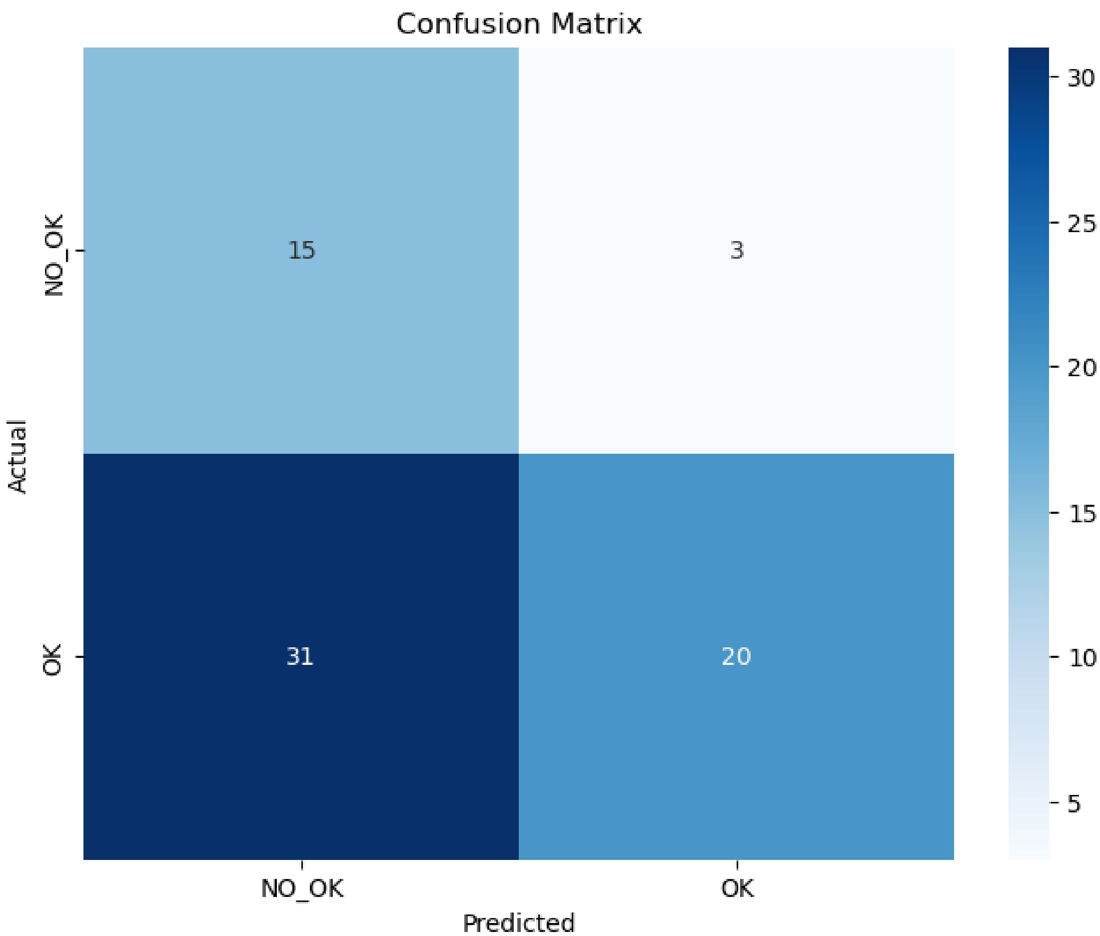
```
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (14): ReLU()
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (16): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (18): ReLU()
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (attention): Sequential(
    (0): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(256, 1, kernel_size=(1, 1), stride=(1, 1))
    (4): Sigmoid()
  )
  (classifier): Sequential(
    (0): AdaptiveAvgPool2d(output_size=(1, 1))
    (1): Flatten(start_dim=1, end_dim=-1)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): ReLU()
    (4): Dropout(p=0.5, inplace=False)
    (5): Linear(in_features=256, out_features=2, bias=True)
  )
)
```

[25]:
```python
# Test evaluation
all_preds = []
all_labels = []
all_probs = []

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs)
        probs = torch.softmax(outputs, dim=1)
        _, preds = torch.max(outputs, 1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())
        all_probs.extend(probs.cpu().numpy())
```

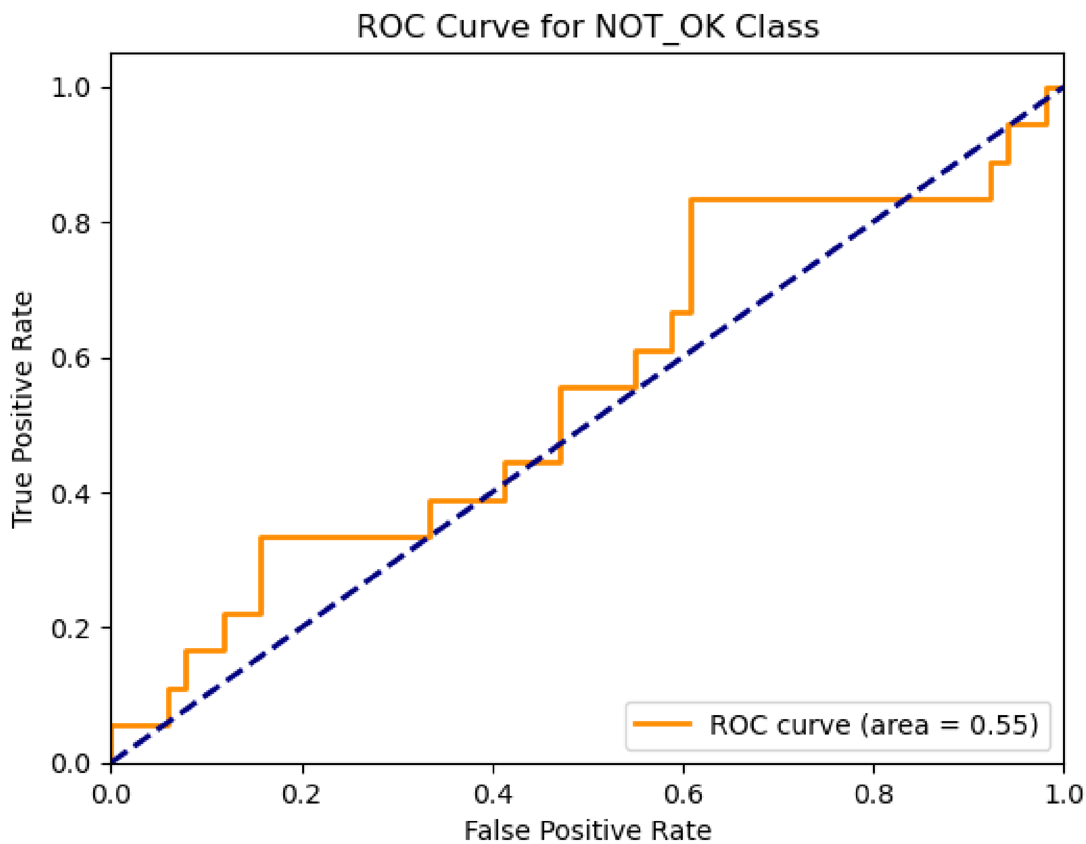
[26]:
```python
# Classification report
print(classification_report(all_labels, all_preds, target_names=class_names))
```

```
              precision    recall  f1-score   support

       NO_OK       0.33      0.83      0.47        18
          OK       0.87      0.39      0.54        51

    accuracy                           0.51        69
   macro avg       0.60      0.61      0.50        69
weighted avg       0.73      0.51      0.52        69
```

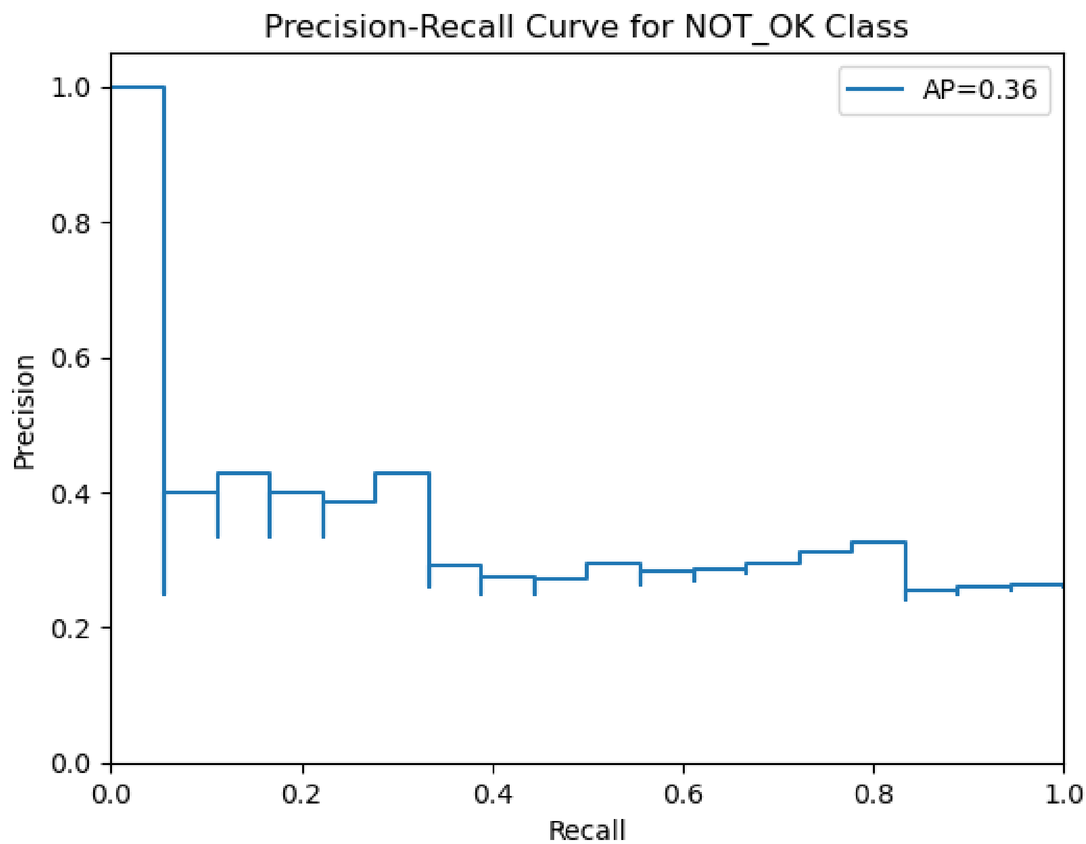
[27]:
```python
# Confusion matrix
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



[28]:
```python
# ROC Curve for NOT_OK class
not_ok_probs = [prob[0] for prob in all_probs]  # Probability of NOT_OK class
fpr, tpr, thresholds = roc_curve(all_labels, not_ok_probs, pos_label=0)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:
  ↪.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for NOT_OK Class')
plt.legend(loc="lower right")
plt.show()
```



[29]:
```
# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(all_labels, not_ok_probs,␣
    ↪pos_label=0)
average_precision = average_precision_score(all_labels, not_ok_probs,␣
    ↪pos_label=0)

plt.figure()
plt.step(recall, precision, where='post', label=f'AP={average_precision:.2f}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall Curve for NOT_OK Class')
plt.legend()
plt.show()
```



1.8 8. Save Final Model

[30]:
```
# Save the entire model
torch.save({
    'model_state_dict': model.state_dict(),
    'class_names': class_names,
    'image_size': IMAGE_SIZE,
    'class_weights': class_weights
}, '../models/textile_defect_model_balanced.pt')

print("Improved model saved successfully!")
```

```
Improved model saved successfully!
```