02_textile_defect_training_baseline_raw_split

March 27, 2025

1 Textile Defect Detection CNN

1.1 1. Environment Setup

[1]:
```
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
import time
import os
from pathlib import Path
```

1.1.1 Check CUDA availability

[2]:
```
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

[3]:
```
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
```

[4]:
```
# Data transforms
train_transform = transforms.Compose([
    transforms.Resize(IMAGE_SIZE),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    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])
])
```

[5]:
```
# Create datasets
train_dataset = datasets.ImageFolder(TRAIN_DIR, transform=train_transform)
test_dataset = datasets.ImageFolder(TEST_DIR, transform=test_transform)
```

[6]:
```
# Class names
class_names = train_dataset.classes
print(f"Classes: {class_names}")
```

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

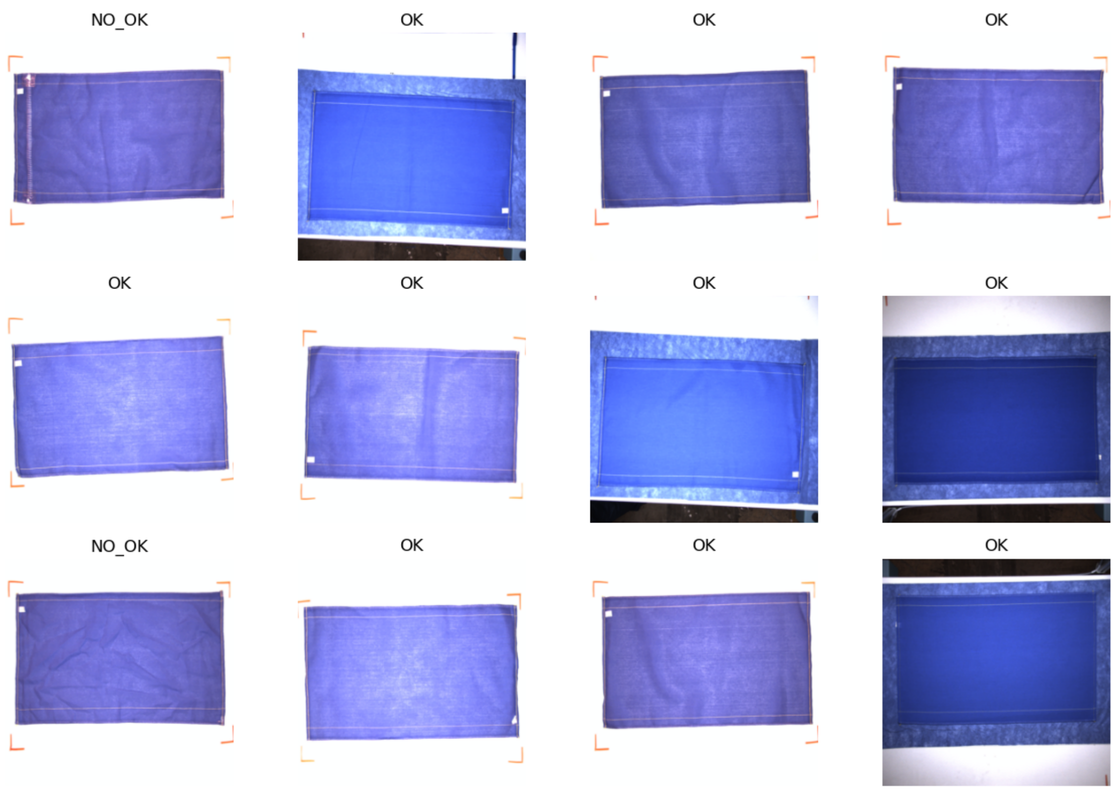
[7]:
```
# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True,␣
↪num_workers=4, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False,␣
↪num_workers=4, pin_memory=True)
```

[8]:
```
# Show batch
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))):
```

```python
        plt.subplot(3, 4, i+1)
        plt.imshow(images[i])
        plt.title(class_names[labels[i]])
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

[9]:
```python
# Display sample
images, labels = next(iter(train_loader))
show_batch(images, labels)
```



1.3 3. CNN Model Definition

[10]:
```python
class TextileDefectCNN(nn.Module):
    def __init__(self, num_classes=2):
        super(TextileDefectCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
```

```python
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

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

            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256 * 16 * 16, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

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

[11]:
```python
model = TextileDefectCNN(num_classes=2).to(device)
```

[12]:
```python
# Print summary
print(model)
```

```
TextileDefectCNN(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (9): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=65536, out_features=512, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=512, out_features=2, bias=True)
  )
)
```

1.4 4. Training Setup

[13]:
```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=3,␣
↪factor=0.1)
```

[14]:
```
# Training function
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()
    best_acc = 0.0

    train_loss_history = []
    train_acc_history = []
    val_loss_history = []
    val_acc_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()
            optimizer.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

        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)

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

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

        scheduler.step(epoch_loss)

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

        # Save best model
        if epoch_acc > best_acc:
            best_acc = epoch_acc
            torch.save(model.state_dict(), '../models/
↪best_model_baseline_raw_split.pt')
```

```
    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.
↪0f}s')
    print(f'Best val Acc: {best_acc:.4f}')

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

1.5 5. Model Training

[15]:
```
# Train the model
model, train_loss, train_acc, val_loss, val_acc = train_model(
    model, criterion, optimizer, scheduler, num_epochs=20
)
```

```
Epoch 1/20
----------
Train Loss: 1.1048 Acc: 0.6679
Val Loss: 0.6231 Acc: 0.7391

Epoch 2/20
----------
Train Loss: 0.6201 Acc: 0.7380
Val Loss: 0.6091 Acc: 0.7391

Epoch 3/20
----------
Train Loss: 0.5903 Acc: 0.7380
Val Loss: 0.5728 Acc: 0.7391

Epoch 4/20
----------
Train Loss: 0.5884 Acc: 0.7380
Val Loss: 0.5825 Acc: 0.7391

Epoch 5/20
----------
Train Loss: 0.5752 Acc: 0.7380
Val Loss: 0.5690 Acc: 0.7391

Epoch 6/20
----------
Train Loss: 0.5691 Acc: 0.7380
Val Loss: 0.5642 Acc: 0.7391

Epoch 7/20
```

```
----------
Train Loss: 0.5705 Acc: 0.7380
Val Loss: 0.5644 Acc: 0.7391

Epoch 8/20
----------
Train Loss: 0.5703 Acc: 0.7380
Val Loss: 0.5578 Acc: 0.7391

Epoch 9/20
----------
Train Loss: 0.5743 Acc: 0.7380
Val Loss: 0.5567 Acc: 0.7391

Epoch 10/20
----------
Train Loss: 0.5578 Acc: 0.7380
Val Loss: 0.5467 Acc: 0.7391

Epoch 11/20
----------
Train Loss: 0.5333 Acc: 0.7380
Val Loss: 0.5447 Acc: 0.7391

Epoch 12/20
----------
Train Loss: 0.5156 Acc: 0.7675
Val Loss: 0.5710 Acc: 0.7391

Epoch 13/20
----------
Train Loss: 0.4861 Acc: 0.7823
Val Loss: 0.5522 Acc: 0.7971

Epoch 14/20
----------
Train Loss: 0.4673 Acc: 0.8339
Val Loss: 0.4841 Acc: 0.7971

Epoch 15/20
----------
Train Loss: 0.4570 Acc: 0.8118
Val Loss: 0.5073 Acc: 0.7826

Epoch 16/20
----------
Train Loss: 0.4394 Acc: 0.8081
Val Loss: 0.4947 Acc: 0.7826
```

```
Epoch 17/20
----------
Train Loss: 0.4463 Acc: 0.7934
Val Loss: 0.4414 Acc: 0.7826

Epoch 18/20
----------
Train Loss: 0.4136 Acc: 0.8155
Val Loss: 0.4425 Acc: 0.8116

Epoch 19/20
----------
Train Loss: 0.4106 Acc: 0.8303
Val Loss: 0.4566 Acc: 0.7681

Epoch 20/20
----------
Train Loss: 0.3802 Acc: 0.8339
Val Loss: 0.4905 Acc: 0.8116

Training complete in 6m 52s
Best val Acc: 0.8116
```

1.6 6. Training Visualization

[16]:
```
# Plot training history
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 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, 2, 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.show()
```

1.7 7. Model Evaluation

[17]:
```
# Load best model
model.load_state_dict(torch.load('../models/best_model_baseline_raw_split.pt'))
model.eval()
```

[17]:
```
TextileDefectCNN(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (9): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=65536, out_features=512, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
```

```
    (4): Linear(in_features=512, out_features=2, bias=True)
  )
)
```

[18]:
```
# Test evaluation
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)

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

[19]:
```
# Classification report
print(classification_report(all_labels, all_preds, target_names=class_names))
```

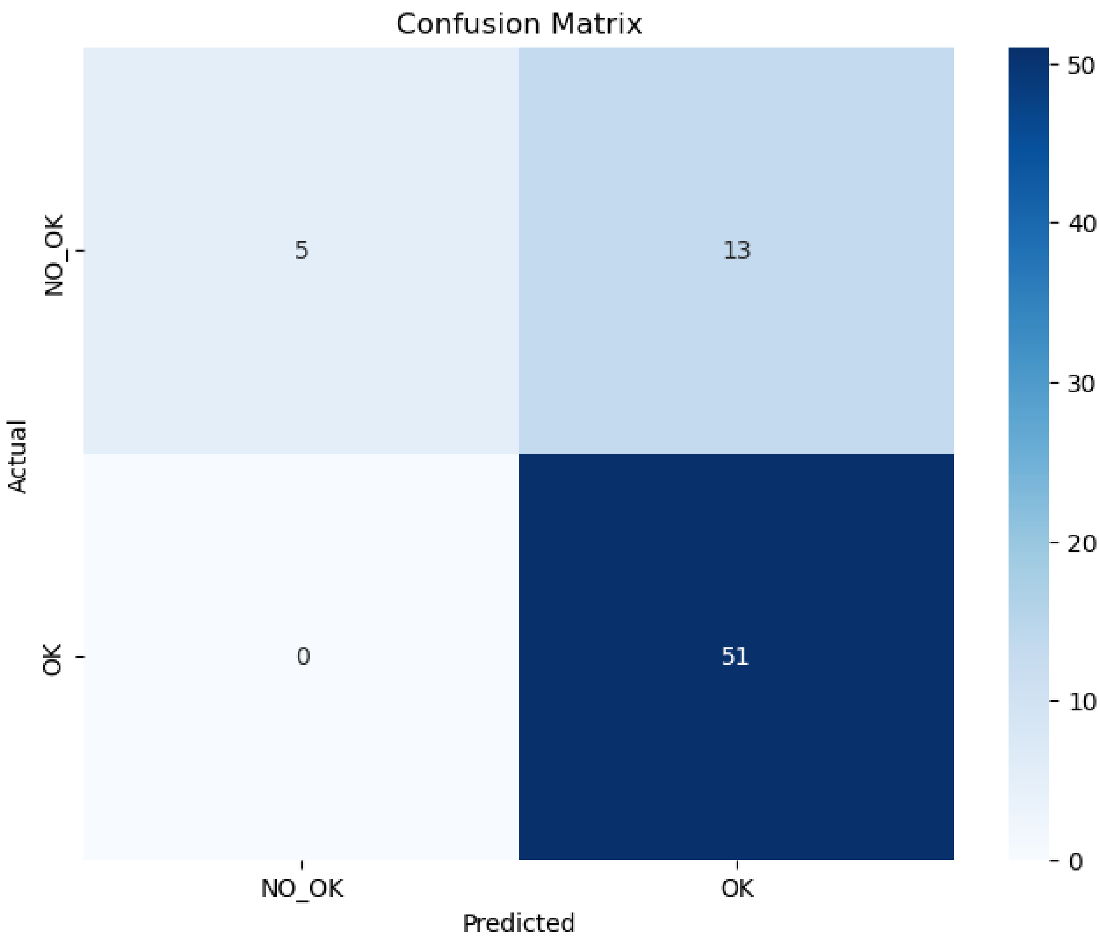
```
              precision    recall  f1-score   support

       NO_OK       1.00      0.28      0.43        18
          OK       0.80      1.00      0.89        51

    accuracy                           0.81        69
   macro avg       0.90      0.64      0.66        69
weighted avg       0.85      0.81      0.77        69
```

[20]:
```
# Confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

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()
```



1.8 8. Save Final Model

[21]:
```
# Save the entire model
torch.save({
    'model_state_dict': model.state_dict(),
    'class_names': class_names,
    'image_size': IMAGE_SIZE
}, '../models/textile_defect_model_baseline_raw_split.pt')

print("Model saved successfully!")
```

```
Model saved successfully!
```