Day 3 Assignment: Cat vs Dog Classifier

Transfer Learning with PyTorch
Computer Vision Bootcamp

made with ♡, by Aryan

Due: Next Session

1 Objective

Build a binary image classifier using transfer learning that distinguishes between cats and dogs, achieving greater than 90 percent accuracy on the test set.

2 Background

Transfer learning lets us leverage models pretrained on ImageNet (1.4M images, 1000 classes) to:

- Train with much less data (100s instead of 1000s of images)
- Achieve better accuracy faster
- Reduce training time from days to minutes

3 Requirements

3.1 Dataset

- Use the Dogs vs Cats dataset from Kaggle: `https://www.kaggle.com/c/dogs-vs-cats`
- Or any cat/dog image collection (minimum 500 images per class)
- Split: 80% train, 10% validation, 10% test
- Organize as:

```
data/
  train/
    cats/
      cat.1.jpg
      cat.2.jpg
      ...
    dogs/
      dog.1.jpg
      dog.2.jpg
      ...
  val/
    cats/
    dogs/
```

```
test/
  cats/
  dogs/
```

3.2 Model Requirements

1. Use ResNet18 pretrained on ImageNet
2. Freeze all layers except the final classification layer
3. Replace final layer with binary classifier (2 outputs)
4. Use CrossEntropyLoss as loss function
5. Use Adam optimizer

3.3 Training Requirements

- Train for minimum 5 epochs
- Implement data augmentation (5+ techniques)
- Implement learning rate scheduling (ReduceLROnPlateau or StepLR)
- Track and plot training & validation metrics (loss and accuracy)
- Save the best model based on validation accuracy
- Achieve $> 90\%$ accuracy on test set

3.4 Data Augmentation

Implement at least 5 of these for training:

- Random horizontal flip
- Random rotation (± 15 degrees)
- Random resized crop
- Color jitter (brightness, contrast, saturation)
- Random affine transformations

Important: Only apply augmentation to training data, NOT validation/test data!

3.5 Evaluation

Your submission must include:

- Test accuracy ($> 90\%$ required)
- Confusion matrix visualization
- Training/validation loss curves
- Training/validation accuracy curves
- Example predictions showing 5 correct and 5 incorrect classifications

4 Starter Code

4.1 Data Preparation

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# Data augmentation for training
train_transforms = transforms.Compose([
    transforms.Resize(256),
    transforms.RandomCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2,
                           saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# No augmentation for validation/test
val_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])
])

# Load datasets
train_dataset = datasets.ImageFolder('data/train',
                                     transform=train_transforms)
val_dataset = datasets.ImageFolder('data/val',
                                   transform=val_transforms)
test_dataset = datasets.ImageFolder('data/test',
                                    transform=val_transforms)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

print(f'Training samples: {len(train_dataset)}')
print(f'Validation samples: {len(val_dataset)}')
print(f'Test samples: {len(test_dataset)}')
```

4.2 Model Setup

```python
# Load pretrained ResNet18
model = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1
    )

# Freeze all layers
```

```python
for param in model.parameters():
    param.requires_grad = False

# Replace final layer for binary classification
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 2)

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

print(f'Using device: {device}')
print(f'Training only final layer with {model.fc.in_features} output
    to 2')
```

4.3 Training Setup

```python
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

# Learning rate scheduler
scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='min', patience=3, factor=0.5
)

# Training tracking
train_losses = []
val_losses = []
train_accs = []
val_accs = []
best_val_acc = 0.0
```

4.4 Training Loop

```python
num_epochs = 10

for epoch in range(num_epochs):
    # Training phase
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

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

        # Forward
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```python

        # Statistics
        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = 100 * correct / total
    train_losses.append(train_loss)
    train_accs.append(train_acc)

    # Validation phase
    model.eval()
    val_running_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)

            val_running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            val_total += labels.size(0)
            val_correct += (predicted == labels).sum().item()

    val_loss = val_running_loss / len(val_loader)
    val_acc = 100 * val_correct / val_total
    val_losses.append(val_loss)
    val_accs.append(val_acc)

    # Update learning rate
    scheduler.step(val_loss)

    # Save best model
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save(model.state_dict(), 'best_model.pth')
        print(f'Saved best model with val_acc: {val_acc:.2f}%')

    print(f'Epoch [{epoch+1}/{num_epochs}]')
    print(f'  Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%')
    print(f'  Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%')
    print()
```

4.5 Evaluation on Test Set

```python
# Load best model
model.load_state_dict(torch.load('best_model.pth'))
model.eval()

correct = 0
total = 0
```

```python
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)
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

test_accuracy = 100 * correct / total
print(f'Test Accuracy: {test_accuracy:.2f}%')
```

4.6 Visualization

```python
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Plot training curves
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

ax1.plot(train_losses, label='Train Loss')
ax1.plot(val_losses, label='Val Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.set_title('Training and Validation Loss')
ax1.legend()

ax2.plot(train_accs, label='Train Acc')
ax2.plot(val_accs, label='Val Acc')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy (%)')
ax2.set_title('Training and Validation Accuracy')
ax2.legend()

plt.tight_layout()
plt.savefig('training_curves.png')
plt.show()

# Confusion matrix
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.savefig('confusion_matrix.png')
plt.show()
```

5 Grading Rubric

Criteria	Points
Transfer Learning Implementation	20
Data Augmentation (5+ techniques)	15
Learning Rate Scheduling	15
Training Loop & Validation	20
Model Saving & Loading	10
Evaluation & Visualization	15
Achieves ¿90% Test Accuracy	5
Total	**100**

6 Bonus Challenges (Optional)

6.1 Bonus 1: Fine-tuning (+10 points)

After initial training, unfreeze the last few ResNet layers and fine-tune with a lower learning rate (0.0001).

```
# Unfreeze last residual block
for param in model.layer4.parameters():
    param.requires_grad = True

# Fine-tune with lower LR
optimizer = optim.Adam([
    {'params': model.layer4.parameters(), 'lr': 0.0001},
    {'params': model.fc.parameters(), 'lr': 0.001}
])
```

6.2 Bonus 2: Try Different Architectures (+10 points)

Compare ResNet18 with MobileNetV2 and report which performs better.

6.3 Bonus 3: Visualize Predictions (+10 points)

Display a grid showing:

- 10 correctly classified images
- 10 incorrectly classified images
- Include predicted and actual labels

7 Submission

Submit a ZIP file: `Day3_Assignment_YourName.zip`

```
Day3_Assignment_YourName/
|-- train.py                # Complete training script
|-- evaluate.py             # Evaluation script
|-- best_model.pth          # Saved model weights
|-- training_curves.png     # Loss and accuracy plots
```

```
|-- confusion_matrix.png      # Confusion matrix
|-- README.txt                # Your name, test accuracy, notes
```

README.txt should include:

- Your name
- Final test accuracy
- Data augmentation techniques used
- Learning rate schedule used
- Any challenges faced
- Bonus challenges attempted (if any)

8 Tips for Success

1. **Start early!** Download dataset first (it's large)
2. **Use validation set** to tune hyperparameters, not test set
3. **Monitor training curves** - if overfitting, add more augmentation
4. **Try different learning rates** if not converging
5. **Save checkpoints** regularly to avoid losing progress
6. **Use GPU** if available (Colab provides free GPUs)

9 Common Issues & Solutions

- **Out of memory:** Reduce batch size to 16 or 8
- **Low accuracy ($<$ 80%):** Train longer, try different LR
- **Overfitting:** Add more augmentation, increase dropout
- **Loss not decreasing:** Check if data is normalized correctly

Good luck! Build something amazing!