Assignment 7: Stanford Cars Dataset - Transfer Learning

Overview

This assignment implements transfer learning using MobileNetV3-Small for car classification on the Stanford Cars Dataset. The focus is on fine-tuning a pre-trained model and addressing overfitting issues.

Score: 93/100 ⭐

Objectives

- Apply transfer learning using pre-trained MobileNetV3-Small
- Implement proper layer freezing for fine-tuning
- Handle overfitting between training and validation sets
- Achieve high classification accuracy with efficient training

Dataset

Stanford Cars Dataset

- **Total images**: 16,185
- **Classes**: 196 car categories
- **Training images**: 8,144
- **Test images**: 8,041
- **Class granularity**: Make, Model, Year

Task Requirements

Step 1: Download Dataset

Download the Stanford Cars Dataset from Kaggle:

- **URL**: https://www.kaggle.com/datasets/jutrera/stanford-car-dataset-by-classes-folder/data

Step 2: Data Preparation

Training and Validation Sets

- Use images from the **train folder**
- Split using **StratifiedKFold** with ratio **8:2** (80% train, 20% validation)
- Generate training and validation data loaders

Test Set

- Use images from the **test folder**
- Generate test data loader

Step 3: Load Pre-trained Model

Load **MobileNetV3-Small** with the following steps:

1. Load the model and pretrained weights
2. Obtain preprocessing transform from `weights.transforms()`
3. Modify the network output layer for 196 classes
4. Apply the preprocessing transform in datasets

Required outputs:

```python
print(model)
print(transform)
```

Important: Don't get transform twice - use it efficiently in your code organization.

Step 4: Layer Freezing

Fine-tuning configuration:

- **Train**: Layers after `features.11` (included)
- **Freeze**: All other layers before `features.11`

Verify layer freezing by printing:

```python
for name, param in model.named_parameters():
    print(name, param.requires_grad)
```

Step 5: Model Training

Training Configuration

- **Epochs**: 5
- Print epoch-wise results:

```
[1] Train Accuracy: 70.2%, Avg loss: 1.270668 Val Accuracy: 94.7%, Avg loss: 0.174458
[2] Train Accuracy: ..., Avg loss: ... Val Accuracy: ..., Avg loss: ...
...
```

Step 6: Model Evaluation

Evaluate on test set and print:

```
Test Accuracy: 98.4%, Avg loss: 0.074889
```

Important Notes from Professor Feedback

Code Organization

- ⚠️ **Don't get transform twice** - reorganize your program to obtain and use transform efficiently
- Keep code clean and well-structured

Overfitting Issue

- ⚠️ **Large gap between training and validation accuracy indicates overfitting**
- Strategies to address overfitting:
  - Add dropout layers
  - Use data augmentation
  - Implement regularization (L2 weight decay)
  - Train fewer layers or use lower learning rate
  - Early stopping based on validation loss

Transfer Learning Best Practices

Model Modification

```python
```

```python
import torch
import torchvision.models as models

# Load pretrained model and weights
weights = models.MobileNet_V3_Small_Weights.DEFAULT
model = models.mobilenet_v3_small(weights=weights)

# Get transform once
transform = weights.transforms()

# Modify classifier for 196 classes
num_classes = 196
model.classifier[3] = torch.nn.Linear(model.classifier[3].in_features, num_classes)
```

Freeze Layers

python

```python
# Freeze all layers first
for param in model.parameters():
    param.requires_grad = False

# Unfreeze layers after features.11
for name, param in model.named_parameters():
    if 'features.11' in name or 'features.12' in name or 'classifier' in name:
        param.requires_grad = True
```

Addressing Overfitting

python

```python
# Data augmentation
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    # ... then apply pretrained transform normalization
])

# Optimizer with weight decay
optimizer = torch.optim.Adam(
    filter(lambda p: p.requires_grad, model.parameters()),
    lr=0.001,
    weight_decay=1e-4  # L2 regularization
)
```

Deliverables

- Jupyter notebook containing:
  - Dataset loading with proper transforms
  - MobileNetV3-Small model loading and modification
  - Layer freezing verification output
  - Training loop for 5 epochs
  - Epoch-wise training and validation metrics
  - Final test evaluation
  - Overfitting mitigation strategies implemented

Important:

- ✅ Use template and fill all fields
- ✅ Reorganize code to avoid redundant transform calls
- ✅ Address overfitting issues
- ❌ Do NOT compress the notebook file
- ⚠️ **Plagiarism Warning**: Both parties receive 0 score with no second chances

Requirements

```python
torch
torchvision
numpy
scikit-learn
pillow
matplotlib
```

Installation

```bash
pip install torch torchvision numpy scikit-learn pillow matplotlib
```

Expected Performance

- Training and validation accuracy gap should be minimized
- Validation accuracy should reach 90%+ by epoch 5
- Test accuracy demonstrates good generalization

- Model should not show severe overfitting

Results

Successfully implemented transfer learning using MobileNetV3-Small with proper fine-tuning strategy. Addressed overfitting issues through data augmentation and regularization, achieving strong generalization on the Stanford Cars Dataset.

Assignment completed as part of Deep Learning coursework