CO543 Image Processing
Lab 07
Convolutional Neural Networks

Goal

The goal of this lab is to understand and apply the principles of transfer learning and to compare different convolutional neural network architectures and hyperparameters.

Important Concepts of Transfer Learning

Transfer learning involves leveraging pre-trained models, and deep learning models previously trained on large datasets like ImageNet for vision tasks. Developers can use these models to expedite and enhance the training of new models. Two primary techniques in transfer learning are fine-tuning and feature extraction.

- **Fine-tuning** uses the pre-trained model as a base and retrains it on a new dataset with a very small learning rate to adapt it to a new task. This process involves adjusting the weights of the pre-trained model slightly to suit the new data.

- **Feature extraction**, on the other hand, treats the pre-trained model as a fixed feature extractor. In this approach, the model's learned features are used as they are, and only the final classification layer is replaced and trained on the new dataset.

Why Use Transfer Learning?

- Reduced Training Time: Transfer learning utilizes pre-trained models, significantly cutting down the time required to train a model for a new task, as opposed to training from scratch.

- Improved Performance with Limited Data: Deep learning models typically require large datasets for effective training. Transfer learning leverages knowledge from a pre-trained model on a large dataset, improving performance even with limited task-specific data.

- Reduced Computational Resources: Utilizing a pre-trained model reduces the computational resources needed for training, which is beneficial when hardware resources are limited.

- Faster Experimentation: Starting with a pre-trained model allows for rapid experimentation with different model architectures and fine-tuning for specific needs, accelerating the development process.

Let's Train a State-of-the-art model on the MNIST Dataset

Step 1: Install the necessary libraries

```
pip install torch torchvision
```

Step 2: Import required libraries and their functions

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchvision import models
from torch.utils.data import DataLoader
```

Step 3: Load and Preprocess the MNIST Dataset

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

train_dataset = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=transform)

test_dataset = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Normalization of input data, such as using Normalize(mean=[0.5], std=[0.5]), is crucial. This step involves subtracting the mean and dividing by the standard deviation of the dataset, which speeds up the training process by standardizing the input values.

Transforming input data to a suitable form and scale for efficient processing by the model is also essential for the effective training and testing of deep learning models on image datasets.

The reason why we resize is because the model we are using for the task which is VGG16 is trained on 224x 224 images. `ToTensor()` is used to transform 0-255 discrete values into 0-1 values which are easier to work with neural networks.

Step 4: Load the Pre-trained VGG Model and Modify It

```
class MNISTVGGNet(nn.Module):
    def __init__(self):
        super(MNISTVGGNet, self).__init__()
        self.vgg16 = models.vgg16(weights=models.VGG16_Weights.IMAGENET1K_V1)

        self.vgg16.features[0] = nn.Conv2d(1, 64, kernel_size=11, stride=4, padding=2)

        num_ftrs = self.vgg16.classifier[6].in_features
        self.alexnet.classifier[6] = nn.Linear(num_ftrs, 10)

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

We set the first layer to the convolutional layer and it takes one channel as input because we have one channel (Grayscale) image and VGG16 was trained on 3 channel (RGB) images.

Step 5: Define the Loss Function and optimizer and initialize the model

```
device = 'cuda' if torch.cuda.is_available() else
'cpu'

model = MNISTVGGNet().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(vgg.parameters(), lr=0.001)
```

If GPU resources are available, set the training device to `'cuda'`, otherwise the model will train on the CPU.

Step 6: Train the Model

```
num_epochs = 5
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()

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss:
{running_loss/len(train_loader):.4f}")
```

Step 7: Evaluate the Model

```
model.eval()
correct = 0
total = 0
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()

print(f'Accuracy of the model on the test images: {100 * correct / total} %')
```

Lab Task 1: Transfer Learning

Train two state-of-the-art models **resnet18** and **AlexNet** on MNIST digits dataset (10 epoch, Adam optimizer with learning rate 0.001) and compare the results (loss, accuracy).

Identify and explain the main architectural differences in these two models.

Lab Task 2: Impact of Hyperparameters

Model A

- Convolutional Layer with 8 features
- Maxpool Layer with a kernel size of 2
- Linear Layer with 64 nodes
- Linear Layer with 10 nodes
- Use the relu activation function for nonlinear activation

Model B

- Convolutional Layer with 8 features
- Convolutional Layer with 16 features
- Maxpool Layer with a kernel size of 2
- Linear Layer with 64 nodes
- Linear Layer with 10 nodes
- Use the relu activation function for nonlinear activation

1. Train Model A and Model B for the above dataset considering data points as images, use the Adam optimizer with a learning rate of 0.001
2. Observe and write down the difference in trained model performance
3. What could be the reason for these observed differences in performance? Explain
4. Explore the effect of different activation functions
    a. Train model A with all nonlinear activation functions set to ReLu
    b. Train the model A with all nonlinear activation functions set to Sigmoid
    c. Train the model A with all nonlinear activation functions set to tanh
    d. Observe and discuss the differences between changing activation functions and trained model performance

5. Effect of the optimizer learning rate
    a. Trained the Model B on Adam optimizer with a learning rate of 0.1
    b. Trained the Model B on Adam optimizer with a learning rate of 0.01
    c. Trained the Model B on Adam optimizer with a learning rate of 0.001
    d. Observe and discuss the effect of learning rate on model performance

Submission

You need to submit all python files containing the relevant functions named according to the relevant question names or as indicated in the lab sheet, along with the main function to run them and display your outputs. Make sure to include the images you used to run the codes as well.

You need to submit a report (eYYXXXresults.pdf) including

- Screenshots for the codes
- All results from your code (your output images, graphs, tables under each section after performing the required functions).
- Necessary explanations

under every part of the lab task.

You can submit a single ZIP file as eYYXXXlab07.zip including all:

- Python source codes
- All Input images
- Report

Note: YYXXX indicates your registration number in all cases

References

1. Parameters and Hyperparameters in Machine Learning and Deep Learning
2. A Practical Guide to Transfer Learning using PyTorch
3. Understand the Impact of Learning Rate on Neural Network Performance
4. Activation Functions in Neural Networks