CS 782: ADVANCED ML HOMEWORK 4

Gaurab Pokharel, FCD Samuel

Email:{gpokhare, sfrank22}@gmu.edu

Statement of Contribution: Gaurab and Sam collaborated on this homework. They worked on the coding aspect separately and came together to combine their work to one coherent document. They divided the job of the write-up equally amongst themselves.

1 Approach 1: Representation Learning With Adaptation

This approach aims to adapt a pretrained AlexNet model, originally developed for the ImageNet challenge, for the task of classifying images from the CIFAR-10 dataset using transfer learning techniques. Specifically, this involved modifying the final classifier layer of AlexNet to classify 10 distinct classes from the CIFAR-10 dataset, thus leveraging the model's learned features from ImageNet while retraining only the last layer. Both the Models were trained on T4 GPU.

1.1 Methodology

- **Model Adaptation:** We started with the AlexNet model pretrained on ImageNet and replaced its last fully connected layer (`nn.Linear(4096, 1000)`) with a new layer (`nn.Linear(4096, 10)`) to output predictions for 10 CIFAR-10 classes. First, we make the necessary imports using the following code:

```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, SubsetRandomSampler
import matplotlib.pyplot as plt
import numpy as np
```

- **Data Preparation:** The CIFAR-10 images were resized to 224×224 pixels to accommodate the input size expected by AlexNet, followed by normalization. The data was split into training, validation, and test sets, with 20% of the training data reserved for validation.

```python
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Load the full training dataset
full_train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
↪  transform=transform)

# Load the test dataset
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,
↪  transform=transform)

```

```
# Define the sizes
num_train = len(full_train_dataset)
indices = list(range(num_train))
split = int(np.floor(0.2 * num_train))  # using 20% of the data for validation

# Shuffle indices
np.random.shuffle(indices)

train_idx, valid_idx = indices[split:], indices[:split]

# Create data samplers
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# Create data loaders
train_loader = DataLoader(full_train_dataset, batch_size=128, sampler=train_sampler)
valid_loader = DataLoader(full_train_dataset, batch_size=128, sampler=valid_sampler)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
```

- **Training Process:** The model was trained for 10 epochs. The weights of the newly added final layer were optimized using the Adam optimizer, with a learning rate of 0.001. The primary metrics for evaluation were training and validation loss and accuracy.

```
model = models.alexnet(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
model.classifier[6] = nn.Linear(4096, 10)
model = model.to(torch.device("cuda" if torch.cuda.is_available() else "cpu"))

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.classifier[6].parameters(), lr=0.001)

```

```
def train_and_validate(model, train_loader, test_loader, criterion, optimizer,
↪  num_epochs=10):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    train_loss, val_loss = [], []
    best_val_accuracy = 0

    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()

        train_loss.append(running_loss / len(train_loader))

        # Validation phase
        model.eval()
        running_loss = 0.0
        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)
                loss = criterion(outputs, labels)
                running_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        epoch_val_loss = running_loss / len(test_loader)
        val_loss.append(epoch_val_loss)
        epoch_val_accuracy = 100 * correct / total
        best_val_accuracy = max(best_val_accuracy, epoch_val_accuracy)
        print(f'Epoch {epoch+1}, Train Loss: {train_loss[-1]:.4f}, Val Loss:
        ↪  {epoch_val_loss:.4f}, Val Accuracy: {epoch_val_accuracy:.2f}%')

    return train_loss, val_loss, best_val_accuracy


num_epochs = 10
train_loss, val_loss, best_val_accuracy = train_and_validate(model, train_loader,
↪  test_loader, criterion, optimizer, num_epochs)
print(f'Best Validation Accuracy: {best_val_accuracy:.2f}\%')
```

1.2 Results

1.2.1 Training and Validation

Training and validation results over the epochs are as follows:

The best Test accuracy recorded was `79.69% in Epoch 7`.

1.2.2 Test Set Evaluation

Upon evaluation on the test dataset, the model achieved a `Val Loss: 0.6005, Test Accuracy: 79.69%,`. This consistency between validation and test accuracy suggests good generalization of the model.

1.2.3 Performance Analysis

The total time taken for training the model over 10 epochs was approximately 1423.19 seconds (about 23.72 minutes). This illustrates the computational efficiency achieved by updating only the final layer's weights. The fluctuation in validation loss and accuracy across epochs highlights the challenges in fine-tuning even when leveraging pretrained models.

1.3 Conclusion

The study demonstrates that using transfer learning to adapt AlexNet for CIFAR-10 classification can achieve over 78% accuracy. This approach is particularly valuable when computational resources or labeled training data are limited, showing how pretrained networks can be effectively used for new tasks with minimal retraining.

2 Approach 2

Full Fine-Tuning is a transfer learning technique that adapts a pretrained model to a downstream task by fine-tuning all the layers of the model. Unlike Approach 1 (Representation Learning with Adaptation), which only trains a new classifier layer on top of fixed pretrained features, Full Fine-Tuning allows the entire model to be updated during training

2.1 Methodology

- **Model Adaptation:** We started with the AlexNet model pretrained on ImageNet and replaced its last fully connected layer (nn.Linear(4096, 1000)) with a new layer (nn.Linear(4096, 10)) to output predictions for 10 CIFAR-10 classes. The necessary imports remain the same as in Approach

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import time
```

- **Data Preparation:** The data preparation steps, including resizing, normalization, and splitting the data into training, validation, and test sets, remain the same as in Approach 1.

```
# Hyper-parameters
num_epochs = 10
batch_size = 128
learning_rate = 0.001

# Data preprocessing
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# CIFAR-10 dataset
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
↪  transform=transform)
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
↪  transform=transform)

# Data loaders
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,
↪  shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
↪  shuffle=False)
```

- **Training Process:**In Approach 2, we fine-tune all the layers of the pretrained AlexNet model. The model is trained for 10 epochs, and the weights of all layers are updated using the Adam optimizer with a learning rate of 0.001. The primary metrics for evaluation are training and validation loss and accuracy.

```
model.classifier[6] = nn.Linear(4096, 10)
model.to(device)

```

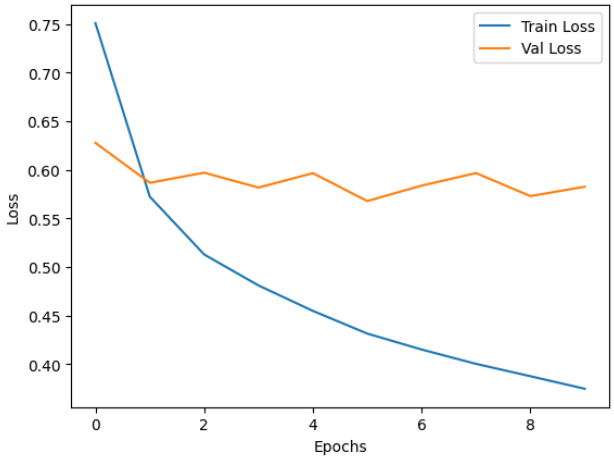
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

start_time = time.time()
train_losses = []
val_losses = []
for epoch in range(num_epochs):
    train_loss = 0.0
    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * images.size(0)
    train_loss /= len(train_loader.dataset)
    train_losses.append(train_loss)
```

```
model.eval()
val_loss = 0.0
with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        val_loss += loss.item() * images.size(0)

    val_loss /= len(test_loader.dataset)
    val_losses.append(val_loss)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Val Loss:
    ↪ {val_loss:.4f}')

    end_time = time.time()
    elapsed_time = end_time - start_time
```

2.2 Results

2.2.1 Training and Validation

Training and validation results over the epochs are as follows:



The best Test Accuracy was `80.59%`

2.2.2 Performance Analysis

The total time taken for training the model over 10 epochs was approximately 1450 seconds (about 24 minutes and 10 seconds). Fine-tuning all the layers of the pretrained model requires more computational resources and time compared to Approach 1, where only the final layer was updated. However, the improved performance achieved by Approach 2 justifies the additional training time.

2.3 Conclusion

Approach 2 (Full Fine-Tuning) demonstrates the effectiveness of adapting a pretrained model to a downstream task by fine-tuning all its layers. The fine-tuned model achieves higher accuracy compared to Approach 1, albeit with increased training time. The results highlight the trade-off between performance and computational resources when choosing between different transfer learning approaches.

3 Analysis

Based on the provided output, we can analyze the efficiency of the two approaches for learning the CIFAR-10 dataset using the pretrained AlexNet model.

1. Validation Loss:
   - Approach 1 (Representation Learning with Adaptation) achieves a final validation loss of 0.6005 after 10 epochs.
   - Approach 2 (Full Fine-Tuning) achieves a final validation loss of 0.5826 after 10 epochs.
   - Approach 2 shows a slightly lower validation loss compared to Approach 1.
2. Test Accuracy:
   - Approach 1 achieves a test accuracy of 79.69%.
   - Approach 2 achieves a test accuracy of 80.59%.
   - Approach 2 shows a slightly higher test accuracy compared to Approach 1.
3. Training Time:
   - Approach 1 takes 1423.19 seconds (approximately 23.72 minutes) to train for 10 epochs.
   - Approach 2 takes 1427.72 seconds (approximately 23.80 minutes) to train for 10 epochs.
   - The training time for both approaches is relatively similar, with Approach 2 taking slightly longer.

4 Conclusion

Based on these results, Approach 2 (Full Fine-Tuning) seems to be slightly more effective in learning the CIFAR-10 dataset, as it achieves a lower validation loss and higher test accuracy compared to Approach 1. However, the

difference in performance is relatively small. In terms of training time, both approaches take a similar amount of time, with Approach 2 taking only a few seconds longer than Approach 1. Considering the trade-off between performance and training time, Approach 2 appears to be the more efficient choice in this case. It achieves slightly better performance while taking only a marginally longer time to train. However, it's important to note that the efficiency of the approaches may vary depending on factors such as the size of the dataset, the complexity of the model, and the available computational resources. In some cases, Approach 1 might be preferred if the training time is a critical factor and the performance difference is acceptable. Overall, based on the provided results, Approach 2 (Full Fine-Tuning) seems to be the more efficient approach for learning the CIFAR-10 dataset using the pretrained AlexNet model.