Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «Обработка изображений в ИС»

Тема: «Конструирование моделей на базе предобученных нейронных сетей»

Выполнил:

Студент 4 курса

Группы ИИ-21

Богуш А. Д.

Проверил:

Крощенко А.А.

Брест 2024

Цель: осуществить обучение НС, сконструированных на базе предообученных архитектур НС.

Вариант 8.

В-т	Выборка	Оптимизатор	Предобученная архитектура
8	Fashion-MNIST	Adam	ResNet34

Код программы:

```
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
import matplotlib.pyplot as plt
batch_size = 32
num_epochs = 10
learning_rate = 0.001
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
train_dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True,
transform=transform)
test_dataset = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True,
transform=transform)

train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
pretrained_model = models.resnet34(pretrained=True)
pretrained_model.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=(7, 7),
stride=(2, 2), padding=(3, 3), bias=False)
num_features = pretrained_model.fc.in_features
pretrained_model.fc = nn.Linear(num_features, 10)
untrained_model = models.resnet34(pretrained=False)
untrained_model.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=(7, 7),
stride=(2, 2), padding=(3, 3), bias=False)
untrained_model.fc = nn.Linear(num_features, 10)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
pretrained_model = pretrained_model.to(device)
untrained_model = untrained_model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer_pretrained = optim.Adam(pretrained_model.parameters(), lr=learning_rate)
optimizer_untrained = optim.Adam(untrained_model.parameters(), lr=learning_rate)
def train(model, optimizer, train_loader):
    model.train()
    train_losses = []
    for epoch in range(num_epochs):
        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()

        avg_loss = running_loss / len(train_loader)
        train_losses.append(avg_loss)
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {avg_loss:.4f}')
    return train_losses
def test(model, test_loader):
    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()

    accuracy = 100 * correct / total
    return accuracy
print("Training pretrained model...")
pretrained_losses = train(pretrained_model, optimizer_pretrained, train_loader)
pretrained_accuracy = test(pretrained_model, test_loader)
print(f'Accuracy of pretrained model: {pretrained_accuracy:.2f}%')

print("Training untrained model...")
untrained_losses = train(untrained_model, optimizer_untrained, train_loader)
untrained_accuracy = test(untrained_model, test_loader)
print(f'Accuracy of untrained model: {untrained_accuracy:.2f}%')
epochs = range(1, num_epochs + 1)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs, pretrained_losses, label='Pretrained Model Loss')
plt.plot(epochs, untrained_losses, label='Untrained Model Loss', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss Comparison')
plt.legend()

plt.subplot(1, 2, 2)
bars = plt.bar(['Pretrained', 'Untrained'], [pretrained_accuracy, untrained_accuracy],
color=['blue', 'orange'])
plt.ylabel('Accuracy (%)')
plt.title('Test Accuracy Comparison')

for bar in bars:
    yval = bar.get_height()
```

```
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{yval:.2f}%', ha='center',
va='bottom', fontsize=12)

plt.tight_layout()
plt.show()
```

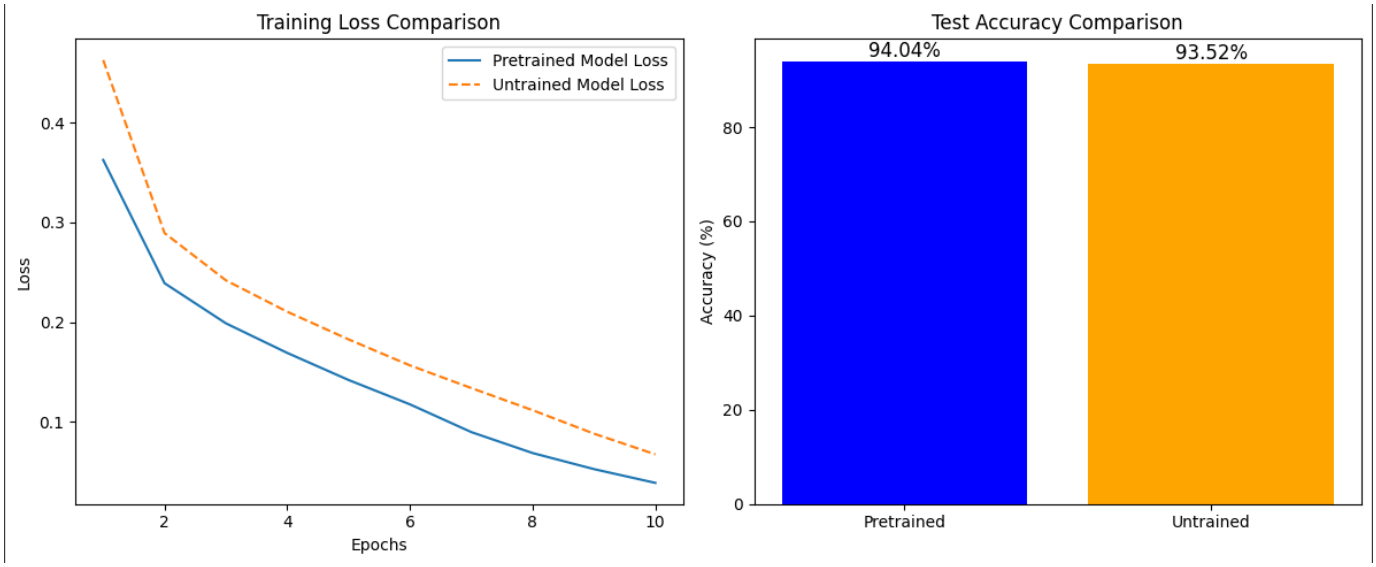

График ошибок и точности.

Вывод: осуществил обучение НС, сконструированных на базе предообученных архитектур НС.