Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «ОИвИС»

Тема: “Конструирование моделей на базе предобученных нейронных сетей”

Выполнил:

Студент 4 курса

Группы ИИ-23

Лапин В. А.

Проверила:

Андренко К.В.

Брест 2025

Цель: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.

Вариант 8.

Выборка: CIFAR-10. Размер исходного изображения: 32*32 Оптимизатор: Adam. Предобученная архитектура:

1. Для заданной выборки и архитектуры предобученной нейронной организовать процесс обучения НС, предварительно изменив структуру слоев, в соответствии с предложенной выборкой. Использовать тот же оптимизатор, что и в ЛР №1. Построить график изменения ошибки и оценить эффективность обучения на тестовой выборке;

2. Сравнить полученные результаты с результатами, полученными на кастомных архитектурах из ЛР №1;

3. Ознакомиться с state-of-the-art результатами для предлагаемых выборок (по материалам в сети Интернет). Сделать выводы о результатах обучения НС из п. 1 и 2;

4. Реализовать визуализацию работы СНС из пункта 1 и пункта 2 (выбор и подачу на архитектуру произвольного изображения с выводом результата);

5. Оформить отчет по выполненной работе, залить исходный код и отчет в соответствующий репозиторий на github.

Код программы:

```
 from torchvision import transforms
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
from torchvision import models


train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

train_loader = torch.utils.data.DataLoader(
    torchvision.datasets.CIFAR10(data_dir, train=True, download=True,
                                 transform=train_transform),
    batch_size=64, shuffle=True)

test_loader = torch.utils.data.DataLoader(
    torchvision.datasets.CIFAR10(data_dir, train=False, download=True,
                                 transform=test_transform),
    batch_size=64, shuffle=False)

model = models.mobilenet_v3_small(pretrained=True)


num_features = model.classifier[3].in_features
model.classifier[3] = nn.Linear(num_features, 10)


device = torch.device('mps' if torch.mps.is_available() else 'cpu')
model = model.to(device)


criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.Adam(model.parameters(), lr=0.001)

def train(model, loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in 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()

        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    accuracy = 100 * correct / total
    return running_loss / len(loader), accuracy

def test(model, loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            running_loss += loss.item()

            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

    accuracy = 100 * correct / total
    return running_loss / len(loader), accuracy

train_losses = []
test_losses = []
train_accuracies = []
```

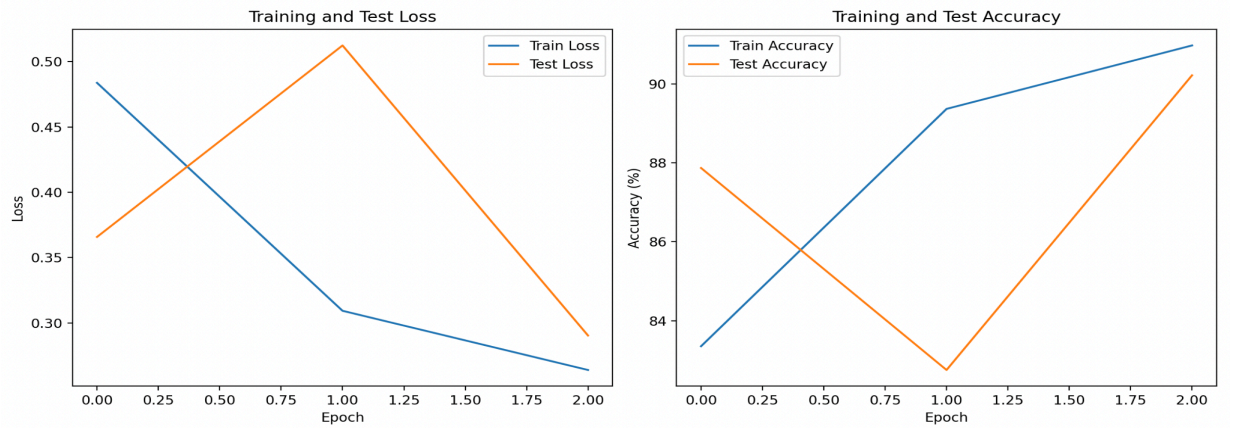
```
test_accuracies = []
num_epochs = 3

for epoch in range(num_epochs):
    train_loss, train_accuracy = train(model, train_loader, criterion, optimizer, device)
    test_loss, test_accuracy = test(model, test_loader, criterion, device)

    train_losses.append(train_loss)
    test_losses.append(test_loss)
    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)
            print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Train Accuracy:
{train_accuracy:.2f}%, '
          f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.2f}%')

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Test Loss')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(test_accuracies, label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Training and Test Accuracy')
plt.legend()
plt.tight_layout()
plt.show()
def imshow(img):
    mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1)
    std = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1)
    img = img * std + mean
    img = img.clamp(0, 1)
    np_img = img.numpy()
    plt.imshow(np.transpose(np_img, (1, 2, 0)))
    plt.axis('off')
    plt.show()

def test_random_image(model, loader, device):
    model.eval()
    images, labels = next(iter(loader))
    images, labels = images.to(device), labels.to(device)

    import random
    index = random.randint(0, images.size(0) - 1)
```

```
    image = images[index].unsqueeze(0)
    label = labels[index].item()

    output = model(image)
    _, predicted = torch.max(output, 1)
    predicted = predicted.item()

    imshow(image.cpu().squeeze())
    print(f'Predicted: {predicted}, Actual: {label}')

test_random_image(model, test_loader, device)
```

1. Результат работы программы:

```
Epoch 1/3, Train Loss: 0.4837, Train Accuracy: 83.35%, Test Loss: 0.3658, Test Accuracy: 87.87%
Epoch 2/3, Train Loss: 0.3092, Train Accuracy: 89.37%, Test Loss: 0.5124, Test Accuracy: 82.75%
Epoch 3/3, Train Loss: 0.2639, Train Accuracy: 90.98%, Test Loss: 0.2903, Test Accuracy: 90.22%
```

График изменения ошибок:



2. SOTA-результаты для выборки:

FMP представляет собой усовершенствованный вариант операции подвыборки, который использует дробные коэффициенты уменьшения размерности вместо стандартных целочисленных значений.

Результат FMP для выборки:

Model Name ↑↓	Percentage correct ↑↓
Fractional MP	96.5

Результат предобученной модели за 3 эпохи:

```
Epoch 1/3, Train Loss: 0.4837, Train Accuracy: 83.35%, Test Loss: 0.3658, Test Accuracy: 87.87%
Epoch 2/3, Train Loss: 0.3092, Train Accuracy: 89.37%, Test Loss: 0.5124, Test Accuracy: 82.75%
Epoch 3/3, Train Loss: 0.2639, Train Accuracy: 90.98%, Test Loss: 0.2903, Test Accuracy: 90.22%
```

Разница в точности обусловлена фундаментальными архитектурными и методологическими различиями между подходами:

1. Специализация архитектуры

FMP-модель разработана исключительно для CIFAR-10 с оптимизированной под датасет архитектурой, включая дробный пулинг для сохранения информации.

Предобученная модель создавалась как универсальная архитектура для мобильных устройств с компромиссом между точностью и эффективностью

2. Подбор гиперпараметров

FMP-модель обучается с тщательно подобранными гиперпараметрами, включая оптимизированные learning rate, weight decay и другие параметры, специфичные для CIFAR-10.

3. Техники пулинга.

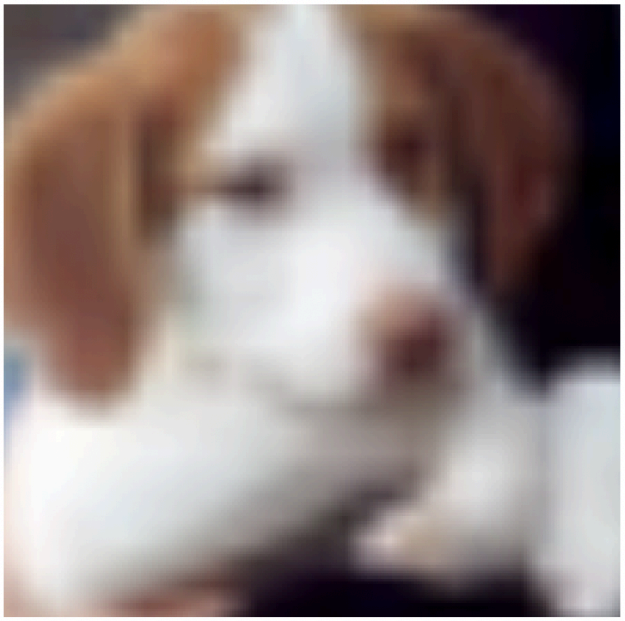
Модели используют разные методы пулинга.

Предобученная модель показала хороший результат на трех эпохах, что демонстрирует эффективность обучения и корректность реализации, а также подтверждает способность архитектуры быстро адаптироваться к новым данным даже при кратковременном обучении.

Однако короткое продолжительности обучения не может отражать реальный потенциал архитектуры, для более точного сравнения необходимо проводить тестирование в равных условиях с одинаковым количеством эпох, идентичными техниками аугментации данных и сопоставимыми вычислительными ресурсами, чтобы оценить истинные архитектурные преимущества каждой модели.

3. Визуализация работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата.

```
Predicted: 5, Actual: 5
```



Вывод: научился конструировать модель на базе предобученной сети.