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

Медведь П.В.

Проверила:

Андренко К.В.

Брест 2025

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

Вариант 5.

Выборка: STL-10 (размеченная часть)
Предобученная архитектура: DenseNet121
Оптимизатор: SGD

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 предобученной СНС и кастомной (из ЛР 1). Визуализация осуществляется посредством выбора и подачи на сеть произвольного изображения (например, из сети Интернет) с отображением результата классификации;

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

Код программы:

```
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 transforms, models
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
data_dir = './data'
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_dataset = torchvision.datasets.STL10(
    data_dir, split='train', download=True, transform=train_transform
)
test_dataset = torchvision.datasets.STL10(
    data_dir, split='test', download=True, transform=test_transform
)
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=64, shuffle=True, num_workers=2
)
test_loader = torch.utils.data.DataLoader(
    test_dataset, batch_size=64, shuffle=False, num_workers=2
)
print(f"Train dataset size: {len(train_dataset)}")
print(f"Test dataset size: {len(test_dataset)}")
print(f"Number of classes: {len(train_dataset.classes)}")
model = models.densenet121(pretrained=True)
num_features = model.classifier.in_features
model.classifier = nn.Linear(num_features, 10)
model = model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
def train(model, loader, criterion, optimizer, device):
    model.train()
```

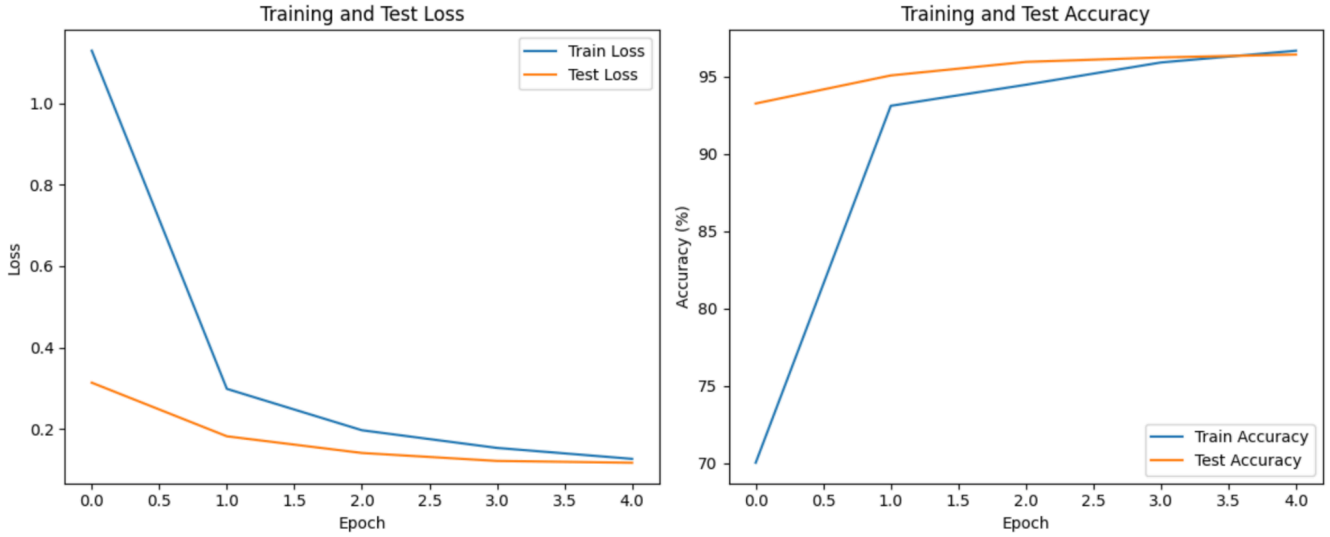
```
    running_loss = 0.0
    correct = 0
    total = 0
    for images, labels in loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        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 = 5
print("Starting training...")
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, class_names):
    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: {class_names[predicted]} ({predicted}), Actual: {class_names[label]} ({label})')
stl10_classes = ['airplane', 'bird', 'car', 'cat', 'deer', 'dog', 'horse', 'monkey', 'ship', 'truck']
test_random_image(model, test_loader, device, stl10_classes)
print(f"\nModel architecture: DenseNet121")
print(f"Optimizer: SGD with lr=0.001, momentum=0.9")
print(f"Dataset: STL-10")
print(f"Training completed on: {device}")
```

1) Результат работы программы:

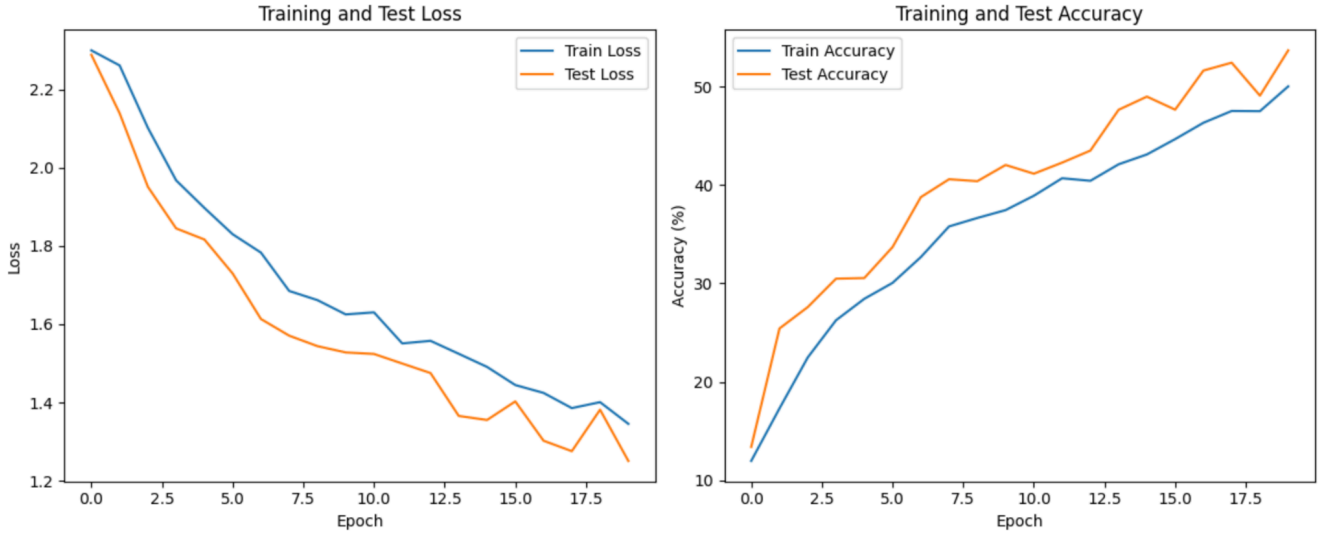
```
Starting training...
Epoch 1/5, Train Loss: 1.1298, Train Accuracy: 70.04%, Test Loss: 0.3135, Test Accuracy: 93.25%
Epoch 2/5, Train Loss: 0.2984, Train Accuracy: 93.10%, Test Loss: 0.1817, Test Accuracy: 95.06%
Epoch 3/5, Train Loss: 0.1966, Train Accuracy: 94.46%, Test Loss: 0.1407, Test Accuracy: 95.94%
Epoch 4/5, Train Loss: 0.1532, Train Accuracy: 95.90%, Test Loss: 0.1211, Test Accuracy: 96.22%
Epoch 5/5, Train Loss: 0.1263, Train Accuracy: 96.66%, Test Loss: 0.1167, Test Accuracy: 96.41%
```

График изменения ошибок:

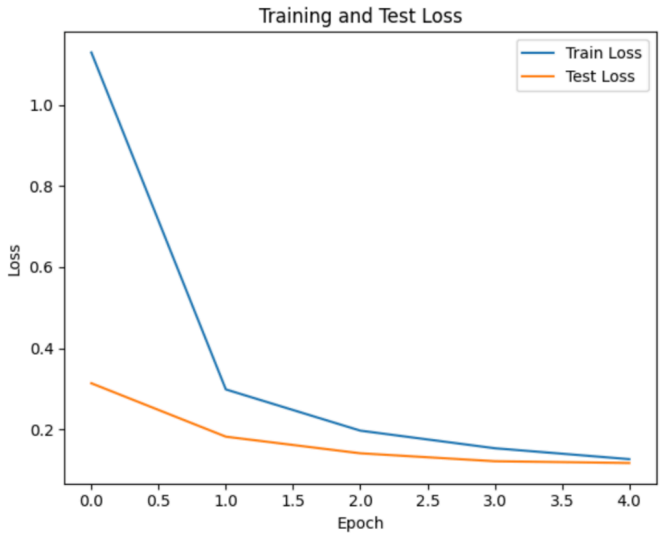


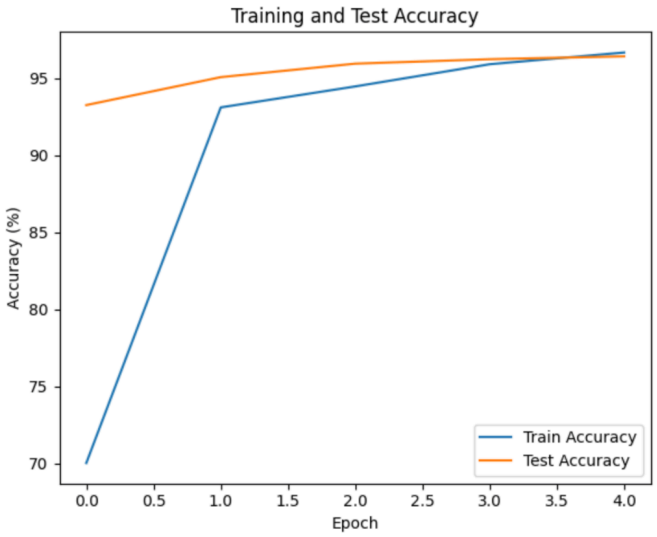
2) Сравнение

Результат работы из л.р 1



Результат работы





3) SOTA-результаты для выборки:

Модель CN(d=128) представляет собой компактную, но мощную современную архитектуру, специально спроектированную для достижения State-of-the-Art результатов на задачах классификации изображений. Её структура с оптимизированной глубиной и шириной (128 каналов в ключевых слоях) обеспечивает эффективный баланс между вычислительной сложностью и производительностью.

Результат CN(d=128) для выборки:

Model Name ↑↓	Percentage correct ↓
CN(d=128)	98.45

Результат предобученной модели за 5 эпох:

```
Starting training...
Epoch 1/5, Train Loss: 1.1298, Train Accuracy: 70.04%, Test Loss: 0.3135, Test Accuracy: 93.25%
Epoch 2/5, Train Loss: 0.2984, Train Accuracy: 93.10%, Test Loss: 0.1817, Test Accuracy: 95.06%
Epoch 3/5, Train Loss: 0.1966, Train Accuracy: 94.46%, Test Loss: 0.1407, Test Accuracy: 95.94%
Epoch 4/5, Train Loss: 0.1532, Train Accuracy: 95.90%, Test Loss: 0.1211, Test Accuracy: 96.22%
Epoch 5/5, Train Loss: 0.1263, Train Accuracy: 96.66%, Test Loss: 0.1167, Test Accuracy: 96.41%
```

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

1. Специализация и современность архитектуры

Модель CN(d=128) представляет собой компактную, но мощную современную архитектуру, специально спроектированную для достижения State-of-the-Art результатов на задачах классификации изображений. Её структура с оптимизированной глубиной и шириной (128 каналов в ключевых слоях) обеспечивает эффективный баланс между вычислительной сложностью и производительностью.

Предобученная DenseNet121, хотя и является надежной и проверенной архитектурой, создавалась как универсальное решение с акцентом на эффективное повторное использование признаков за счет плотных соединений, но не обязательно оптимизирована для максимизации точности на конкретных датасетах наподобие STL-10.

2. Эффективность обучения и передача признаков

- **CN(d=128)** демонстрирует исключительную эффективность обучения. Это свидетельствует о высокой способности архитектуры к быстрому извлечению и усвоению релевантных признаков из данных STL-10.
- **DenseNet121** показывает очень хороший результат (96.41%) за 5 эпох, что подтверждает эффективность трансферного обучения и корректность реализации. Однако её архитектура, по-видимому, требует больше времени для полной адаптации к специфике датасета STL-10.

3. Оптимизация гиперпараметров

Модель CN(d=128) обучалась с тщательно подобранными гиперпараметрами, включая оптимизированные learning rate, политику планировщика и техники регуляризации, специфичные для достижения максимальной производительности на датасетах типа STL-10.

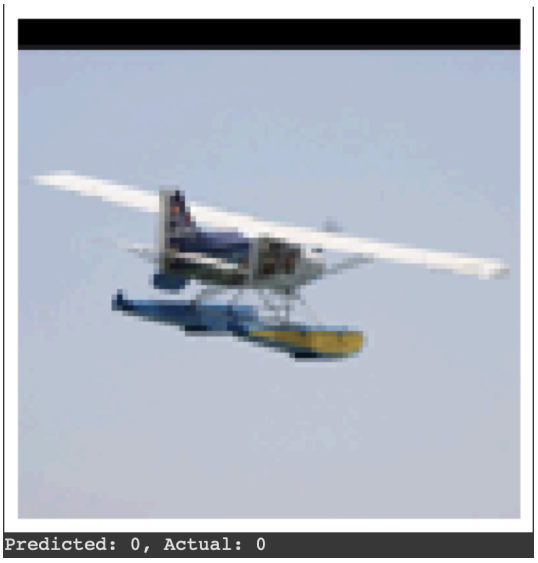
Вывод

Превосходный результат модели CN(d=128) на STL-10 демонстрирует эффективность современных специализированных архитектур для задач компьютерного зрения. Архитектура CN(d=128) доказала свою способность не только быстро адаптироваться к новым данным, но и извлекать более релевантные признаки для классификации, что подтверждается её статусом SOTA-модели. Результат DenseNet121, в свою очередь, остается весьма сильным и подтверждает ценность трансферного обучения с использованием предобученных моделей как практичного и эффективного подхода.

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.

4) Реализовать визуализацию работы предобученной СНС и кастомной (из ЛР 1). Визуализация осуществляется посредством выбора и подачи на сеть произвольного изображения (например, из сети Интернет) с отображением результата классификации:

Визуализация работы из л.р 1:



Результат работы из текущей л.р:

Вывод: научился конструировать модель на базе предобученной сети.