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

Ежевский Е. Р.

Проверила:

Андренко К.В.

Брест 2025

Цель: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС

Общее задание

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.

Вариант:

В-т	Выборка	Оптимизатор	Предобученная архитектура
1	MNIST	SGD	AlexNet

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm

train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.Grayscale(3),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.Grayscale(3),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
```

```
])

train_loader = DataLoader(
   torchvision.datasets.MNIST('/files/', train=True, download=True,
                              transform=train_transform),
   batch_size=128, shuffle=True, num_workers=2, pin_memory=True)

test_loader = DataLoader(
   torchvision.datasets.MNIST('/files/', train=False, download=True,
                              transform=test_transform),
   batch_size=256, shuffle=False, num_workers=2, pin_memory=True)

model = torchvision.models.alexnet(weights='DEFAULT')

for param in model.features.parameters():
   param.requires_grad = False

num_classes = 10
model.classifier[6] = nn.Linear(4096, num_classes)

optimizer = optim.SGD(
   model.parameters(),
   lr=0.01,
   momentum=0.9,
   weight_decay=1e-4
)

scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

criterion = nn.CrossEntropyLoss()

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

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

   progress = tqdm(loader, desc="Training", leave=False)
   for images, labels in progress:
      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)
```

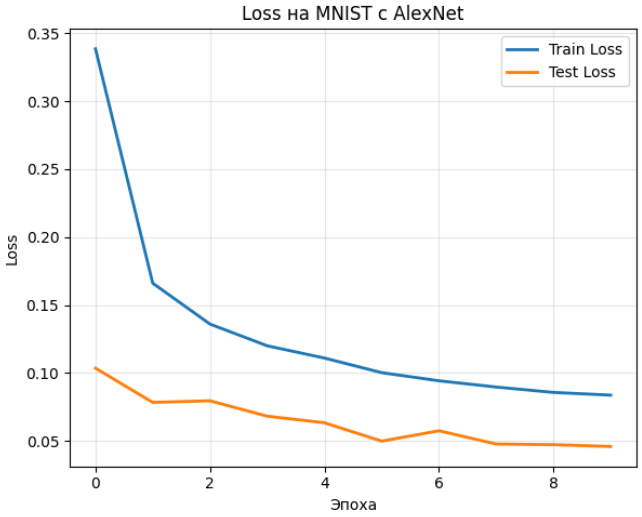
```
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

            progress.set_postfix(loss=loss.item(), acc=100*correct/total)

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

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

    progress = tqdm(loader, desc="Testing", leave=False)
    with torch.no_grad():
        for images, labels in progress:
            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)

            progress.set_postfix(loss=loss.item(), acc=100*correct/total)

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

train_losses, test_losses = [], []
train_accuracies, test_accuracies = [], []

num_epochs = 10

print("Начинаем обучение AlexNet на MNIST...")
for epoch in range(num_epochs):
    print(f"\nЭпоха {epoch+1}/{num_epochs}")

    train_loss, train_accuracy = train(model, train_loader, criterion, optimizer, device)
    test_loss, test_accuracy = test(model, test_loader, criterion, device)

    scheduler.step()

    train_losses.append(train_loss)
    test_losses.append(test_loss)
    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)

    print(f'Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.2f}%')
    print(f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.2f}%')
```

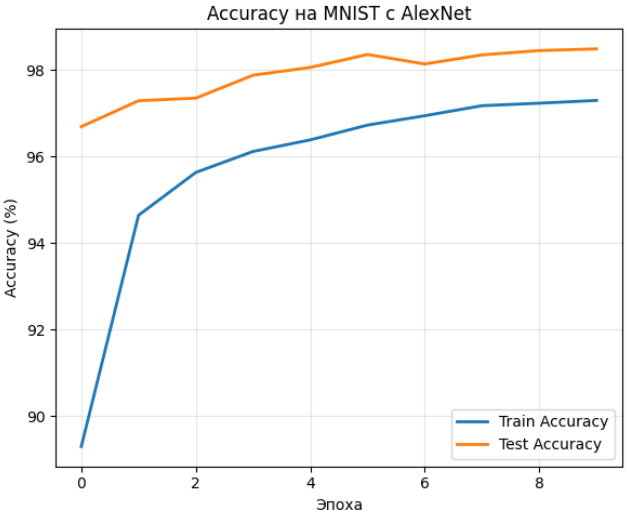
```
    print(f'Learning Rate: {scheduler.get_last_lr()[0]:.6f}')

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

plt.subplot(1,2,1)
plt.plot(train_losses, label='Train Loss', linewidth=2)
plt.plot(test_losses, label='Test Loss', linewidth=2)
plt.xlabel("Эпоха")
plt.ylabel("Loss")
plt.title("Loss на MNIST с AlexNet")
plt.grid(True, alpha=0.3)
plt.legend()

plt.subplot(1,2,2)
plt.plot(train_accuracies, label='Train Accuracy', linewidth=2)
plt.plot(test_accuracies, label='Test Accuracy', linewidth=2)
plt.xlabel("Эпоха")
plt.ylabel("Accuracy (%)")
plt.title("Accuracy на MNIST с AlexNet")
plt.grid(True, alpha=0.3)
plt.legend()

plt.tight_layout()
plt.show()

def imshow(img, ax):
    img = img.cpu().numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    img = std * img + mean
    img = np.clip(img, 0, 1)
    ax.imshow(img)
    ax.axis("off")

classes = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

dataiter = iter(test_loader)
images, labels = next(dataiter)
images, labels = images.to(device), labels.to(device)

model.eval()
with torch.no_grad():
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)

plt.figure(figsize=(16, 8))
for i in range(8):
    ax = plt.subplot(2, 4, i+1)
    imshow(images[i], ax)
    pred_label = classes[predicted[i]]
    true_label = classes[labels[i]]
```

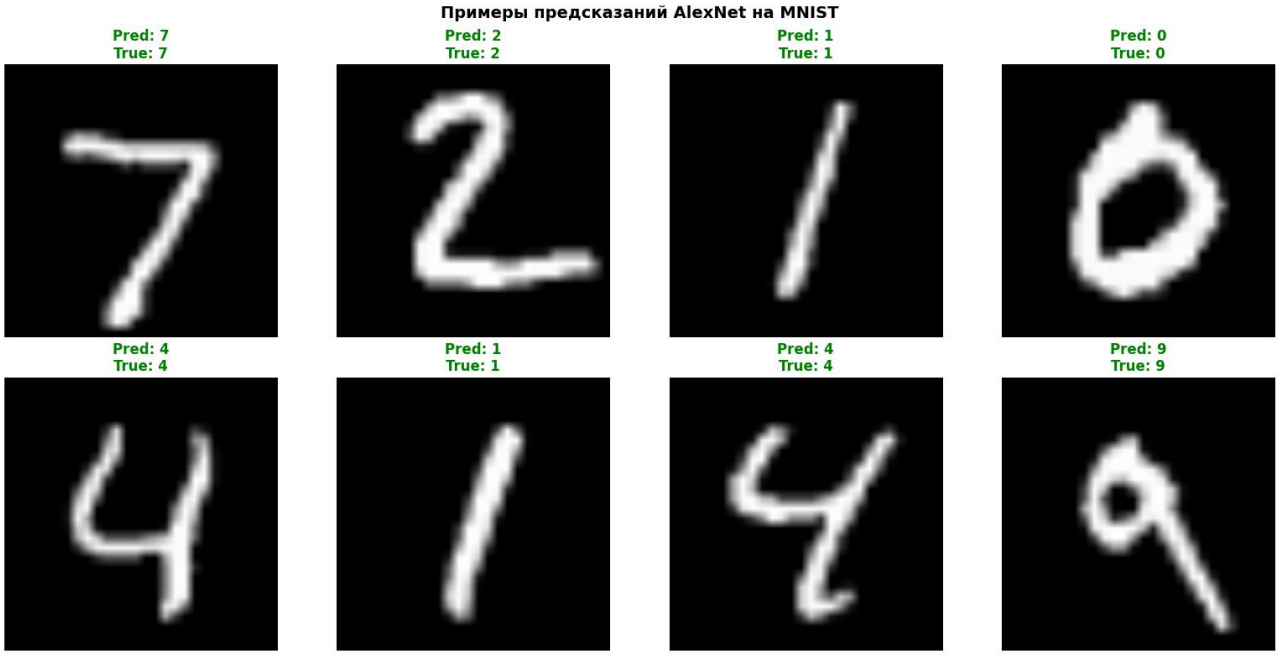
```
    color = "green" if predicted[i] == labels[i] else "red"
    plt.title(f"Pred: {pred_label}\nTrue: {true_label}",
              fontsize=12, color=color, fontweight='bold')
plt.suptitle("Примеры предсказаний AlexNet на MNIST", fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()

print("\n" + "="*50)
print("ФИНАЛЬНЫЕ РЕЗУЛЬТАТЫ:")
print("="*50)
print(f"Лучшая точность на обучении: {max(train_accuracies):.2f}%")
print(f"Лучшая точность на тесте: {max(test_accuracies):.2f}%")
print(f"Финальная точность на тесте: {test_accuracies[-1]:.2f}%")
print(f"Использовался оптимизатор: SGD (lr={0.01}, momentum={0.9})")
print(f"Всего эпох обучения: {num_epochs}")
print("="*50)

model = torchvision.models.alexnet(weights=None)

for param in model.features.parameters():
    param.requires_grad = False

num_classes = 10
model.classifier[6] = nn.Linear(4096, num_classes)

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

train_losses = []
test_losses = []
train_accuracies = []
test_accuracies = []

num_epochs = 10

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("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("Accuracy")
plt.legend()

plt.show()

classes = train_loader.dataset.classes

def imshow(img):
   img = img.cpu().numpy().transpose((1, 2, 0))
   mean = np.array((0.5071, 0.4865, 0.4409))
   std = np.array((0.2673, 0.2564, 0.2761))
   img = std * img + mean
   img = np.clip(img, 0, 1)
   plt.imshow(img)
   plt.axis("off")

dataiter = iter(test_loader)
images, labels = next(dataiter)
images, labels = images.to(device), labels.to(device)

model.eval()
with torch.no_grad():
   outputs = model(images)
   _, predicted = torch.max(outputs, 1)

plt.figure(figsize=(16,8))
for i in range(8):
   plt.subplot(2,4,i+1)
   imshow(images[i])
   plt.title(f"Pred: {classes[predicted[i]]}\nTrue: {classes[labels[i]]}",
          fontsize=9, color=("green" if predicted[i]==labels[i] else "red"))
plt.show()
```

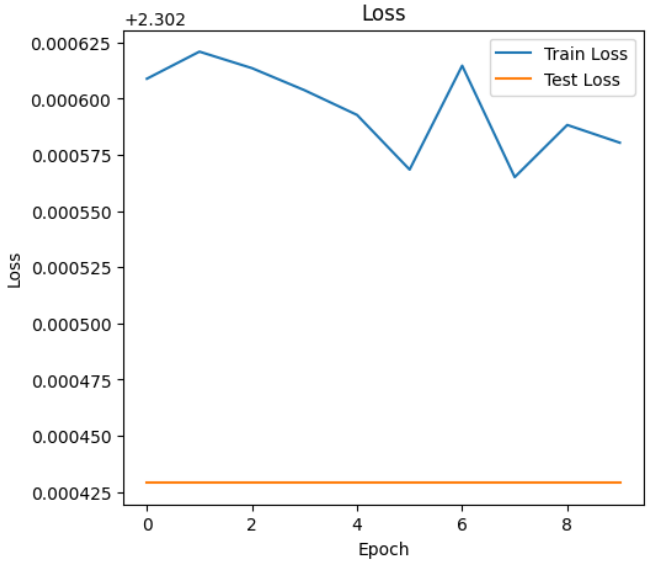
Модель с предобученными весами:

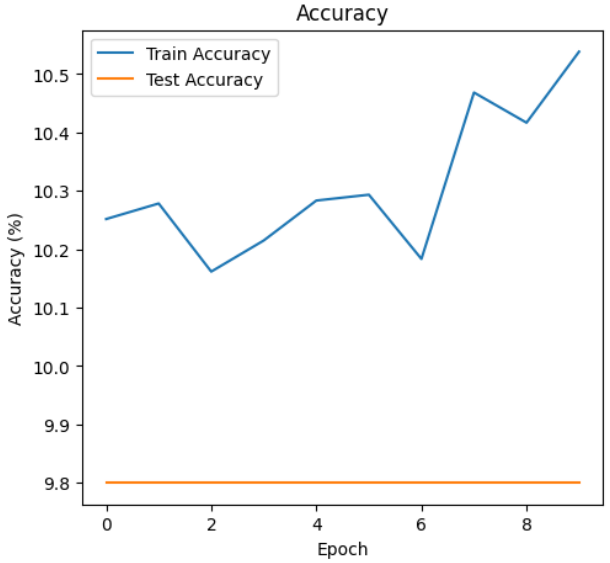


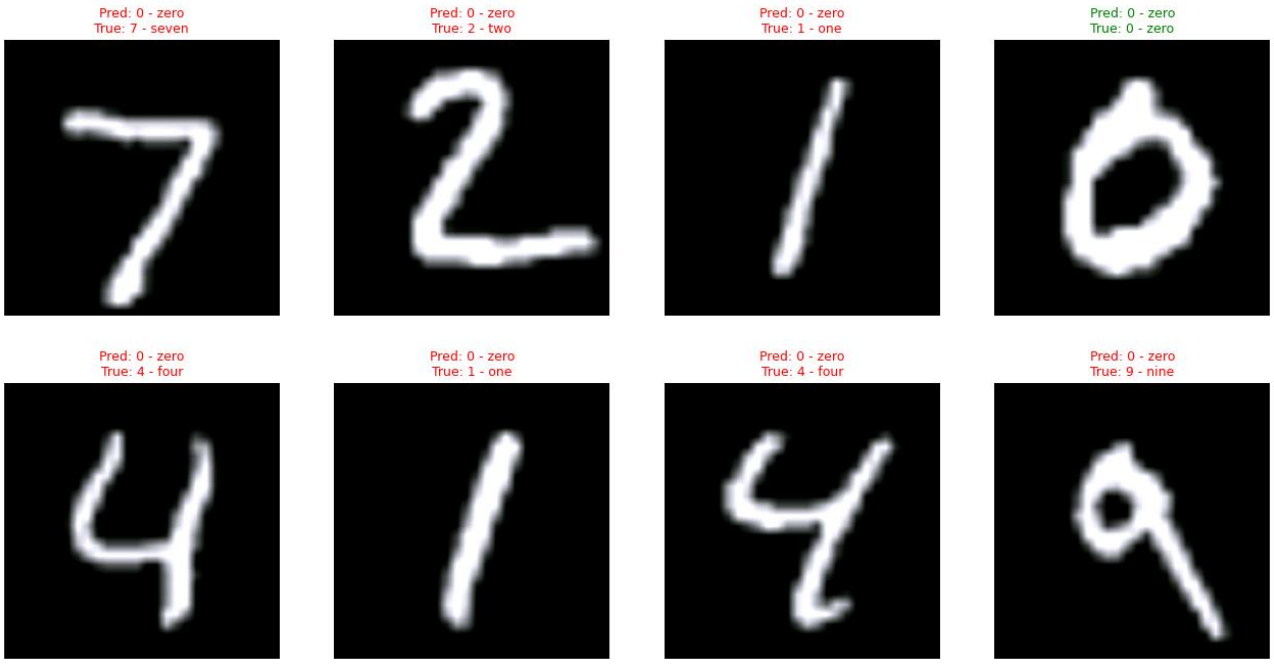




Модель без предобученных весов:







SOTA-результаты для выборки:

Для сравнения я взял классическую архитектуру **AlexNet**, адаптированную под одноканальные изображения MNIST. Модель обучалась N эпох с batch size 128 и стандартной оптимизацией на SGD. Несмотря на то что AlexNet изначально проектировался для работы с ImageNet и крупными RGB-изображениями, в уменьшенном варианте она демонстрирует высокую точность на MNIST. Полученные результаты составили:

Train Loss: 0.0857, Train Acc: 97.23% | Test Loss: 0.0473, Test Acc: 98.45%

Даже в адаптированном виде AlexNet остаётся относительно тяжёлой сетью по сравнению с тем, что обычно используют для MNIST. Благодаря глубокой структуре и нескольким последовательным свёрточным блокам модель уверенно сходится и демонстрирует сильную производительность на простом датасете, где локальные признаки хорошо выражены.

Однако современные SOTA-подходы для MNIST достигают существенно более высоких результатов. Например, ансамбли простых CNN с ядрами 3×3, 5×5 и 7×7 показывают до **99.87% точности**, а двухуровневые ансамбли — вплоть до **99.91%**, что фактически приближается к предельной точности для этого набора данных. Капсульные сети (CapsNet) и гибридные архитектуры CNN + MLP также стабильно достигают **99.8%+**, особенно при использовании расширенной аугментации и продвинутой регуляризации.

Разница в точности объясняется тем, что SOTA-модели для MNIST:

1. **Ориентированы специально на этот датасет**, учитывая его малую вариативность и простую структуру.
2. **Активно используют ансамбли**, что снижает ошибку за счёт агрегирования предсказаний разных архитектур.
3. **Применяют более современные механизмы представления данных**, например капсульные слои, позволяющие моделировать позные взаимоотношения между признаками.
4. **Оптимизированы по размеру, глубине и регуляризации**, что критично при работе на очень чистых и простых изображениях вроде MNIST.

Адаптированная AlexNet, напротив, остаётся относительно тяжёлой и «неглубоко-оптимизированной» под такие простые данные моделью. Она рассчитана на вычленение сложных пространственных структур из больших цветных изображений, а не на извлечение минимальных локальных признаков из цифр 28×28.

Поэтому она демонстрирует хорошую, но не предельную точность по сравнению с узкоспециализированными SOTA-архитектурами.

Вывод: осуществил обучение НС, сконструированных на базе предобученных архитектур НС