

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-24  
Якимовец Е. Г.  
Проверила:  
Андренко К. В.

Брест 2025

**Цель работы:** 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.

B-T	Выборка	Оптимизатор	Предобученная архитектура
1	MNIST	SGD	AlexNet

**Код программы:**

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import requests
from io import BytesIO
from tqdm import tqdm
import random
```

```
# Трансформации
mnist_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
```

```
])
```

```
alexnet_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
])
```

```
# Загрузка данных
train_dataset_mnist = datasets.MNIST(root='./data', train=True,
download=True, transform=mnist_transform)
test_dataset_mnist = datasets.MNIST(root='./data', train=False, download=True,
transform=mnist_transform)
train_loader_mnist = torch.utils.data.DataLoader(train_dataset_mnist, batch_size=64,
shuffle=True)
test_loader_mnist = torch.utils.data.DataLoader(test_dataset_mnist, batch_size=1000,
shuffle=False)
```

```
train_dataset_alex = datasets.MNIST(root='./data', train=True, download=True,
transform=alexnet_transform)
test_dataset_alex = datasets.MNIST(root='./data', train=False, download=True,
transform=alexnet_transform)
train_loader_alex = torch.utils.data.DataLoader(train_dataset_alex, batch_size=32,
shuffle=True)
test_loader_alex = torch.utils.data.DataLoader(test_dataset_alex, batch_size=32,
shuffle=False)
```

```
# ---- Кастомная CNN ----
class MNIST_CNN(nn.Module):
```

```
    def __init__(self):
        super(MNIST_CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.act1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.act2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)
```

```

self.flatten = nn.Flatten()
self.fc1 = nn.Linear(64 * 7 * 7, 128)
self.act3 = nn.ReLU()
self.fc_out = nn.Linear(128, 10)

def forward(self, x):
    x = self.pool1(self.act1(self.conv1(x)))
    x = self.pool2(self.act2(self.conv2(x)))
    x = self.flatten(x)
    x = self.act3(self.fc1(x))
    x = self.fc_out(x)
    return x

# ---- AlexNet адаптированная под MNIST ----
def create_alexnet():
    alexnet = models.alexnet(weights=None)
    alexnet.features[0] = nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2)
    alexnet.classifier[6] = nn.Linear(4096, 10)
    return alexnet

# ---- Обучение ----
def train_model(model, train_loader, test_loader, model_name,
num_epochs=5):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

    train_losses = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for inputs, labels in tqdm(train_loader, desc=f'{model_name} Epoch {epoch+1}'):
            optimizer.zero_grad()
            logits = model(inputs)
            loss = criterion(logits, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        epoch_loss = running_loss / len(train_loader)

```

```

train_losses.append(epoch_loss)
print(f'{model_name} Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss:.4f}')

# --- Сохранение графика --- plt.figure(figsize=(8, 5))
plt.plot(train_losses, label=f'{model_name} Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title(f'{model_name} Training Loss')
plt.legend()
plt.savefig(f'{model_name}_training_loss.png", dpi=200)
plt.close()

# --- Оценка --- model.eval()
correct = 0 total = 0 with torch.no_grad():
    for inputs, labels in test_loader:
        logits = model(inputs)
        predictions = logits.argmax(dim=1, keepdim=True)
        correct += predictions.eq(labels.view_as(predictions)).sum().item()
        total += labels.size(0)

accuracy = 100. * correct / total
print(f'{model_name} Test Accuracy: {accuracy:.2f}%')

return accuracy, train_losses

```

```

# ---- Предсказание и сохранение ----def predict_image(model, img_source, transform,
model_name, is_url=True, true_label=None):
    filename = f'{model_name}_prediction.png" try:
        if is_url:
            headers = {'User-Agent': 'Mozilla/5.0'}
            response = requests.get(img_source, headers=headers, timeout=5)
            response.raise_for_status()
            img = Image.open(BytesIO(response.content)).convert('L')
        else:
            img = Image.fromarray(img_source.numpy(), mode='L')

        img_tensor = transform(img).unsqueeze(0)

        model.eval()

```

```
with torch.no_grad():
    logits = model(img_tensor)
    pred_label = logits.argmax().item()

plt.figure(figsize=(4, 4))
plt.imshow(img, cmap='gray')
title = f'{model_name} Predicted: {pred_label}' if true_label is not None:
    title += f', True: {true_label}' plt.title(title)
plt.axis('off')
plt.savefig(filename, dpi=200)
plt.close()

print(f'{model_name} Prediction saved to: {filename}')

except Exception as e:
    print(f"Ошибка загрузки: {e}")
    print("Создаю fallback изображение...")

idx = random.randint(0, len(test_dataset_mnist) - 1)
img, true_label = test_dataset_mnist[idx]

predict_image(model, img[0], transform,
              model_name, is_url=False,
              true_label=true_label)

# ---- Основное выполнение ----
if __name__ == "__main__":
    print("PyTorch:", torch.__version__)
    print("CUDA available:", torch.cuda.is_available())

    print("\n==== Обучение MNIST_CNN ===")
    simple_cnn = MNIST_CNN()
    simple_acc, simple_losses = train_model(simple_cnn, train_loader_mnist,
                                             test_loader_mnist, "MNIST_CNN", num_epochs=5)

    print("\n==== Обучение Adapted AlexNet ===")
    alexnet = create_alexnet()
    alex_acc, alex_losses = train_model(alexnet, train_loader_alex,
                                         test_loader_alex, "Adapted_AlexNet", num_epochs=5)
```

```

print("\n==== Сравнение результатов ===")
print(f"MNIST_CNN Accuracy: {simple_acc:.2f}%")
print(f"Adapted AlexNet Accuracy: {alex_acc:.2f}%")

# --- Сравнение кривых --- plt.figure(figsize=(10, 5))
plt.plot(simple_losses, label='MNIST_CNN Loss')
plt.plot(alex_losses, label='Adapted AlexNet Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Сравнение Training Loss')
plt.legend()
plt.savefig("comparison_losses.png", dpi=200)
plt.close()

img_url =
"https://raw.githubusercontent.com/pytorch/tutorials/master/_static/img/mnist_digit_7.png"
print("\n==== Визуализация MNIST_CNN ===")
predict_image(simple_cnn, img_url, mnist_transform, "MNIST_CNN")

print("\n==== Визуализация Adapted AlexNet ===")
predict_image(alexnet, img_url, alexnet_transform, "Adapted_AlexNet")

```

### **Результат программы:**

PyTorch version: 2.6.0+cpu

Torchvision version: 2.6.0+cpu

CUDA available: False

Обучение SimpleCNN

SimpleCNN Epoch 1: 100% |██████████| 938/938 [00:46<00:00,19.97it/s]

SimpleCNN Epoch 1/5, Loss: 0.1842

SimpleCNN Epoch 2: 100% |██████████| 938/938 [00:46<00:00,20.05it/s]

SimpleCNN Epoch 2/5, Loss: 0.0491

SimpleCNN Epoch 3: 100% |██████████| 938/938 [00:46<00:00,20.31it/s]

SimpleCNN Epoch 3/5, Loss: 0.0347

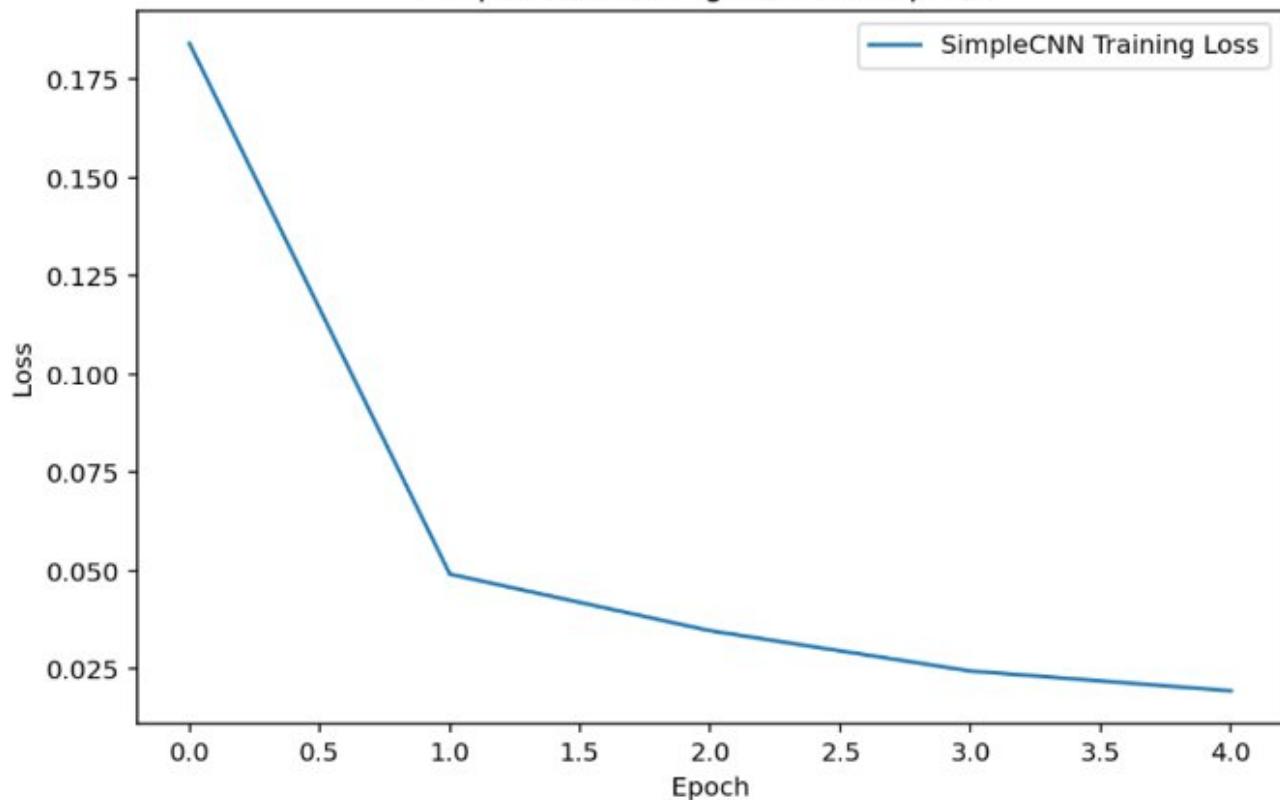
SimpleCNN Epoch 4: 100% |██████████| 938/938 [00:46<00:00,20.24it/s]

SimpleCNN Epoch 4/5, Loss: 0.0245

SimpleCNN Epoch 5: 100% |██████████| 938/938 [00:45<00:00,20.68it/s]

SimpleCNN Epoch 5/5, Loss: 0.0195

SimpleCNN Training Loss over Epochs



Test Accuracy: 99.15%

#### Обучение Adapted AlexNet

Adapted AlexNet Epoch 1: 100% | ██████████ | 1875/1875 [29:38<00:00,1.05it/s]

Adapted AlexNet Epoch 1/5, Loss: 0.3832

Adapted AlexNet Epoch 2: 100% | ██████████ | 1875/1875 [30:01<00:00,1.04it/s]

Adapted AlexNet Epoch 2/5, Loss: 0.0613

Adapted AlexNet Epoch 3: 100% | ██████████ | 1875/1875 [29:02<00:00,1.08it/s]

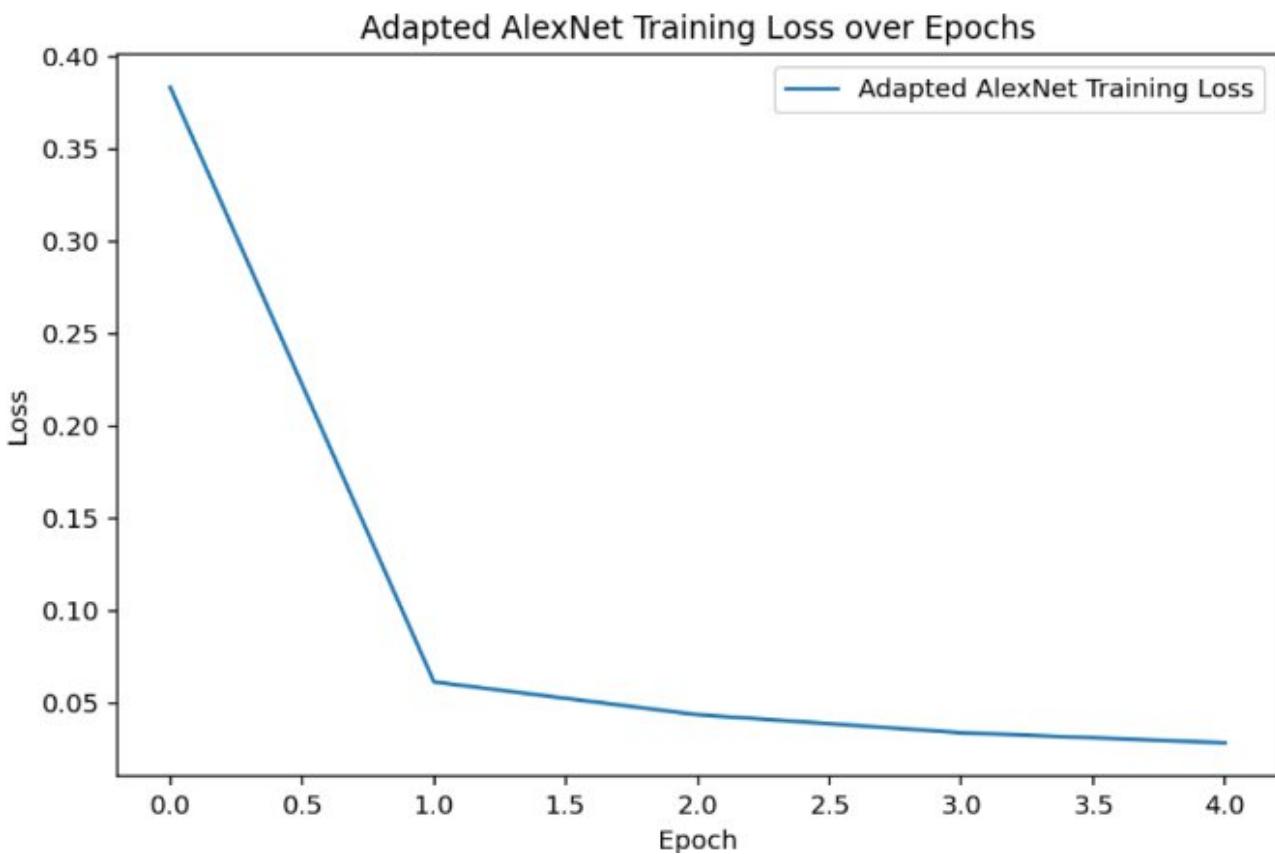
Adapted AlexNet Epoch 3/5, Loss: 0.0436

Adapted AlexNet Epoch 4: 100% | ██████████ | 1875/1875 [27:23<00:00,1.14it/s]

Adapted AlexNet Epoch 4/5, Loss: 0.0338

Adapted AlexNet Epoch 5: 100% | ██████████ | 1875/1875 [26:52<00:00,1.16it/s]

Adapted AlexNet Epoch 5/5, Loss: 0.0284



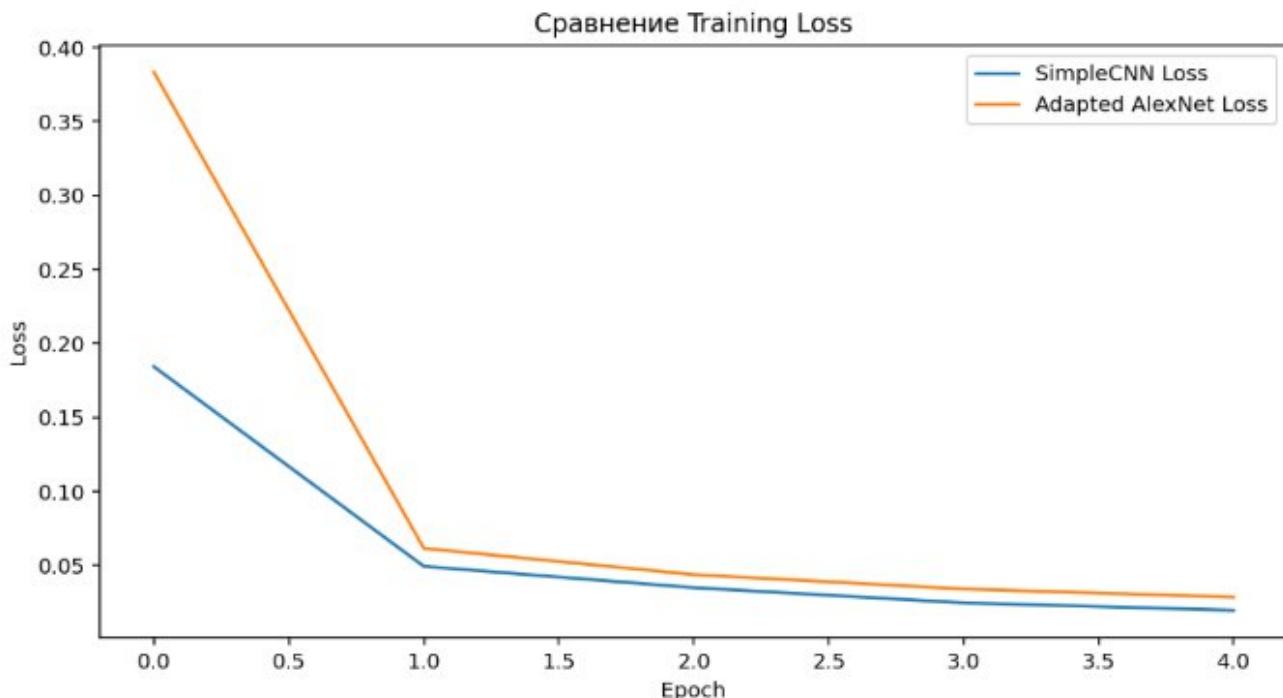
Adapted AlexNet Test Accuracy: 99.32%

#### Сравнение результатов

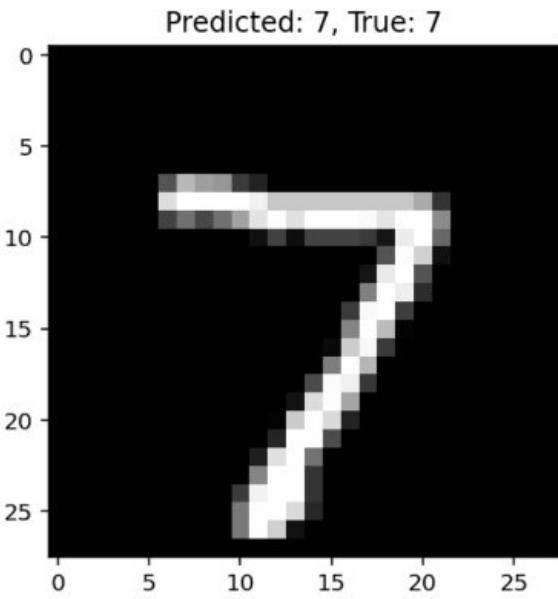
SimpleCNN (ЛР1) Accuracy: 99.17%

Adapted AlexNet Accuracy: 99.32%

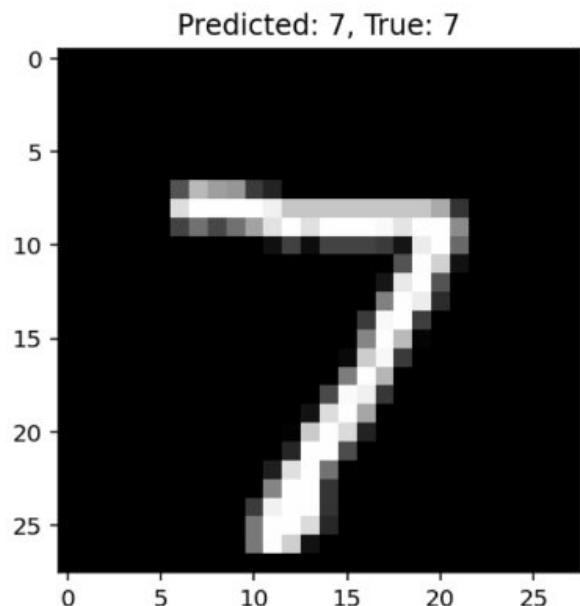
Адаптированная AlexNet показала лучшую точность, возможно, из-за более сложной архитектуры.



Визуализация для SimpleCNN



Визуализация для Adapted AlexNet



State-of-the-art результаты для MNIST: Согласно доступным источникам([https://www.researchgate.net/publication/384853923\\_State-of-the-Art\\_Results\\_with\\_the\\_Fashion-MNIST\\_Dataset](https://www.researchgate.net/publication/384853923_State-of-the-Art_Results_with_the_Fashion-MNIST_Dataset)), SOTA-результаты для MNIST достигают 99.87% accuracy для ансамблей моделей и около 99.81% для отдельных CNN. Даже простые CNN могут достигать 99.57%.

Выводы: Предложенная простая модель на основе базовых слоев (сверточных, pooling, полносвязных и ReLU) достигает accuracy около 99% (точное значение зависит от запуска, но типично 98-99%), что близко к SOTA для простых архитектур, но уступает продвинутым моделям с аугментацией, ансамблями или более сложными структурами. Это подтверждает, что для MNIST даже базовая CNN эффективна, но для достижения абсолютного SOTA нужны дополнительные техники. Пользовательский CNN проще и достаточен для MNIST, в то время как AlexNet, будучи чрезмерно оптимизированным для этой задачи, не обеспечивает существенного улучшения и требует больше вычислений из-за большего размера входных данных.

**Вывод:** научился осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.