

Министерство образования Республики Беларусь  
Учреждение образования  
«Брестский Государственный технический университет»  
Кафедра ИИТ

**Лабораторная работа №1**

По дисциплине «Обработка изображений в интеллектуальных системах»

Тема: «Обучение классификаторов средствами библиотеки PyTorch»

**Выполнила:**

Студентка 4 курса

Группы ИИ-24

Максимович А. И.

**Проверила:**

Андренко К. В.

Брест 2025

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

### **Общее задание**

1. Выполнить конструирование своей модели СНС, обучить ее на выборке по заданию (использовать torchvision.datasets). Предпочтение отдавать как можно более простым архитектурам, базирующимся на базовых типах слоев (сверточный, полносвязный, подвыборочный, слой нелинейного преобразования). Оценить эффективность обучения на тестовой выборке, построить график изменения ошибки (matplotlib);
2. Ознакомьтесь с state-of-the-art результатами для предлагаемых выборок (из материалов в сети Интернет). Сделать выводы о результатах обучения СНС из п. 1;
3. Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата);
4. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

### **Задание по вариантам**

№ варианта	Выборка	Класс	Оптимизатор
11	MNIST	28X28	Adadelata

### **Код:**

```
import os
import time
from PIL import Image
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib

# Используем бэкенд, не требующий дисплея
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from torch.utils.data import Subset

# ===== ПАРАМЕТРЫ (можно менять прямо здесь) =====
EPOCHS = 10
BATCH_SIZE = 128
LR = 1.0 # Типичный learning rate для Adadelata
WEIGHT_DECAY = 1e-4
```

```

USE_CUDA = False
RESUME = True
SAVE_DIR = 'checkpoints_mnist'
VISUALIZE_IMAGE = None
DATA_RATIO = 1.0 # Использовать 100% данных
# =====

# Проверяем доступность CUDA
if USE_CUDA:
    USE_CUDA = torch.cuda.is_available()
    print(f"CUDA доступна: {USE_CUDA}")

device = torch.device('cuda' if USE_CUDA else 'cpu')
print(f"Используемое устройство: {device}")
os.makedirs(SAVE_DIR, exist_ok=True)

# -----
# Улучшенная CNN (VGG-подобная)
# -----
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.features = nn.Sequential(
            # Блок 1
            nn.Conv2d(1, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2), # 14x14

            # Блок 2
            nn.Conv2d(64, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2), # 7x7

            # Блок 3
            nn.Conv2d(128, 256, 3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2), # 3x3 (округление в меньшую сторону)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256 * 3 * 3, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes),
        )

    def forward(self, x):
        x = self.features(x)

```

```

        x = self.classifier(x)
        return x

def main():
    print("Начинаем загрузку данных MNIST...")

    # -----
    # Data (MNIST + аугментация)
    # -----
    transform_train = transforms.Compose([
        transforms.RandomAffine(degrees=10, translate=(0.1, 0.1), scale=(0.9,
1.1)),
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,)) # Стандартные значения
для MNIST
    ])

    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,)) # Стандартные значения
для MNIST
    ])

    try:
        # Загружаем полный датасет MNIST
        full_trainset = torchvision.datasets.MNIST(
            root='./data',
            train=True,
            download=True,
            transform=transform_train
        )

        # Выбираем только часть данных
        num_samples = int(len(full_trainset) * DATA_RATIO)
        indices = torch.randperm(len(full_trainset))[:num_samples]
        trainset = Subset(full_trainset, indices)

        trainloader = torch.utils.data.DataLoader(
            trainset,
            batch_size=BATCH_SIZE,
            shuffle=True,
            num_workers=0
        )

        testset = torchvision.datasets.MNIST(
            root='./data',
            train=False,
            download=True,
            transform=transform_test
        )
        testloader = torch.utils.data.DataLoader(
            testset,
            batch_size=100,
            shuffle=False,
            num_workers=0
        )

        print(f"Используется {num_samples} из {len(full_trainset)}
тренировочных образцов ({DATA_RATIO * 100}%)")
        print("Данные MNIST успешно загружены!")

```

```

except Exception as e:
    print(f"Ошибка при загрузке данных: {e}")
    return

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

# -----
# Модель/оптимизатор/критерий
# -----
print("Инициализация модели...")
model = SimpleCNN(num_classes=10).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta( # Заменяем RMSprop на Adadelta
    model.parameters(),
    lr=LR,
    weight_decay=WEIGHT_DECAY
)
# Убираем scheduler, так как для Adadelta он обычно не используется
# scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer,
T_max=EPOCHS)

# -----
# Загрузка из чекпоинта (если нужно)
# -----
start_epoch = 1
best_acc = 0.0
history = {'train_loss': [], 'test_loss': [], 'test_acc': []}

if RESUME:
    checkpoint_path = os.path.join(SAVE_DIR, 'best.pth')
    if os.path.isfile(checkpoint_path):
        print(f"Загрузка модели из {checkpoint_path} ...")
        try:
            checkpoint = torch.load(checkpoint_path, map_location=device)
            model.load_state_dict(checkpoint['model_state'])
            best_acc = checkpoint.get('acc', 0.0)
            start_epoch = checkpoint.get('epoch', 0) + 1
            # Загружаем историю если есть
            if 'history' in checkpoint:
                history = checkpoint['history']
            print(f"Модель загружена. Лучший acc={best_acc:.2f}% (эпоха
{start_epoch - 1})")
            print(f"История загружена: {len(history['train_loss'])}
эпох")
        except Exception as e:
            print(f"Ошибка при загрузке чекпоинта: {e}")
        else:
            print("Чекпоинт не найден, начинаем обучение с нуля.")

# -----
# Функция валидации
# -----
def evaluate(loader):
    model.eval()
    correct = 0
    total = 0
    running_loss = 0.0
    with torch.no_grad():
        for inputs, targets in loader:
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)

```

```

        loss = criterion(outputs, targets)
        running_loss += loss.item() * inputs.size(0)
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()
    return running_loss / total, 100.0 * correct / total

# -----
# Обучение
# -----
print("Начинаем обучение...")
start_time = time.time()

# Проверяем, нужно ли проводить обучение
if start_epoch <= EPOCHS:
    for epoch in range(start_epoch, EPOCHS + 1):
        model.train()
        running_loss = 0.0
        for i, (inputs, targets) in enumerate(trainloader, 1):
            inputs, targets = inputs.to(device), targets.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * inputs.size(0)

            if i % 50 == 0:
                print(
                    f'Эпоха {epoch}, Батч {i}/{len(trainloader)}, Loss: {loss.item():.4f}, Время: {time.time() - start_time:.2f}c')

        train_loss = running_loss / len(trainloader.dataset)
        test_loss, test_acc = evaluate(testloader)
        # scheduler.step() # Убрали scheduler

        history['train_loss'].append(train_loss)
        history['test_loss'].append(test_loss)
        history['test_acc'].append(test_acc)

        print(
            f'Epoch {epoch}/{EPOCHS} TrainLoss={train_loss:.4f} TestLoss={test_loss:.4f} TestAcc={test_acc:.2f}%'
        )

        if test_acc > best_acc:
            best_acc = test_acc
            checkpoint = {
                'model_state': model.state_dict(),
                'acc': best_acc,
                'epoch': epoch,
                'history': history # Сохраняем историю вместе с моделью
            }
            torch.save(checkpoint, os.path.join(SAVE_DIR, 'best.pth'))
            print(f"Новая лучшая модель сохранена с точностью {best_acc:.2f}%")
        else:

            total_time = time.time() - start_time
            print(f'Обучение завершено за {total_time / 60:.2f} минут. Лучшая точность: {best_acc:.2f}%'
            )

```

```

        print(f"Пропускаем обучение, так как начальная эпоха {start_epoch}
превышает EPOCHS {EPOCHS}")

# -----
# График изменения ошибки
# -----
if history['train_loss'] and history['test_loss']:
    print("Создание графика изменения ошибки...")

    # Создаем простой текстовый файл с данными для отладки
    debug_path = os.path.join(SAVE_DIR, 'loss_data.txt')
    with open(debug_path, 'w') as f:
        f.write("Epoch,Train_Loss,Test_Loss\n")
        for i in range(len(history['train_loss'])):
            f.write(f"{i +
1},{history['train_loss'][i]},{history['test_loss'][i]}\n")
        print(f"Данные для отладки сохранены в {debug_path}")

    # Создаем график с минимальными настройками
    try:
        # Создаем новую фигуру
        fig, ax = plt.subplots(figsize=(10, 6))

        # Данные для оси X (эпохи)
        epochs = range(1, len(history['train_loss']) + 1)

        # Простые линии без сложного форматирования
        ax.plot(epochs, history['train_loss'], 'b-', linewidth=2,
label='Train Loss')
        ax.plot(epochs, history['test_loss'], 'r-', linewidth=2,
label='Test Loss')

        # Простые подписи на английском
        ax.set_xlabel('Epoch')
        ax.set_ylabel('Loss')
        ax.legend()
        ax.set_title('Training and Test Loss - MNIST with Adadelata')

        # Сохраняем разными способами для надежности
        loss_path_png = os.path.join(SAVE_DIR, 'training_loss.png')
        loss_path_pdf = os.path.join(SAVE_DIR, 'training_loss.pdf')

        plt.savefig(loss_path_png, dpi=100, bbox_inches='tight')
        plt.savefig(loss_path_pdf, bbox_inches='tight')
        plt.close(fig) # Явно закрываем фигуру

        print(f'График ошибки сохранен в {loss_path_png} и
{loss_path_pdf}')

    except Exception as e:
        print(f"Ошибка при создании графика: {e}")
        # Альтернативный способ - сохранить данные в CSV
        csv_path = os.path.join(SAVE_DIR, 'loss_data.csv')
        try:
            import pandas as pd
            df = pd.DataFrame({
                'epoch': range(1, len(history['train_loss']) + 1),
                'train_loss': history['train_loss'],
                'test_loss': history['test_loss']
            })
            df.to_csv(csv_path, index=False)

```

```

        print(f"Данные сохранены в CSV: {csv_path}")
    except:
        print("Не удалось сохранить данные в CSV")

    # Выводим финальные значения ошибок
    print(f"Финальная ошибка обучения: {history['train_loss'][-1]:.4f}")
    print(f"Финальная ошибка тестирования:
{history['test_loss'][-1]:.4f}")
    else:
        print("Нет данных для построения графика ошибки")

# -----
# Визуализация предсказания для отдельного изображения (опционально)
# -----
def predict_image(img_path):
    img = Image.open(img_path).convert('L').resize((28, 28))
    x = transform_test(img).unsqueeze(0).to(device)
    model.eval()
    with torch.no_grad():
        logits = model(x)
        probs = torch.softmax(logits, dim=1).cpu().numpy()[0]
        pred = int(np.argmax(probs))
    return img, pred, probs

if VISUALIZE_IMAGE and os.path.exists(VISUALIZE_IMAGE):
    print(f"Визуализация изображения: {VISUALIZE_IMAGE}")
    img, pred_idx, probs = predict_image(VISUALIZE_IMAGE)
    plt.figure(figsize=(4, 4))
    plt.imshow(img, cmap='gray')
    plt.axis('off')
    plt.title(f'Prediction: {classes[pred_idx]}\nConfidence:
{probs[pred_idx] * 100:.1f}%')

    single_pred_path = os.path.join(SAVE_DIR, 'single_prediction.png')
    plt.savefig(single_pred_path, dpi=150, bbox_inches='tight')
    plt.close()
    print(f"Визуализация предсказания сохранена в {single_pred_path}")

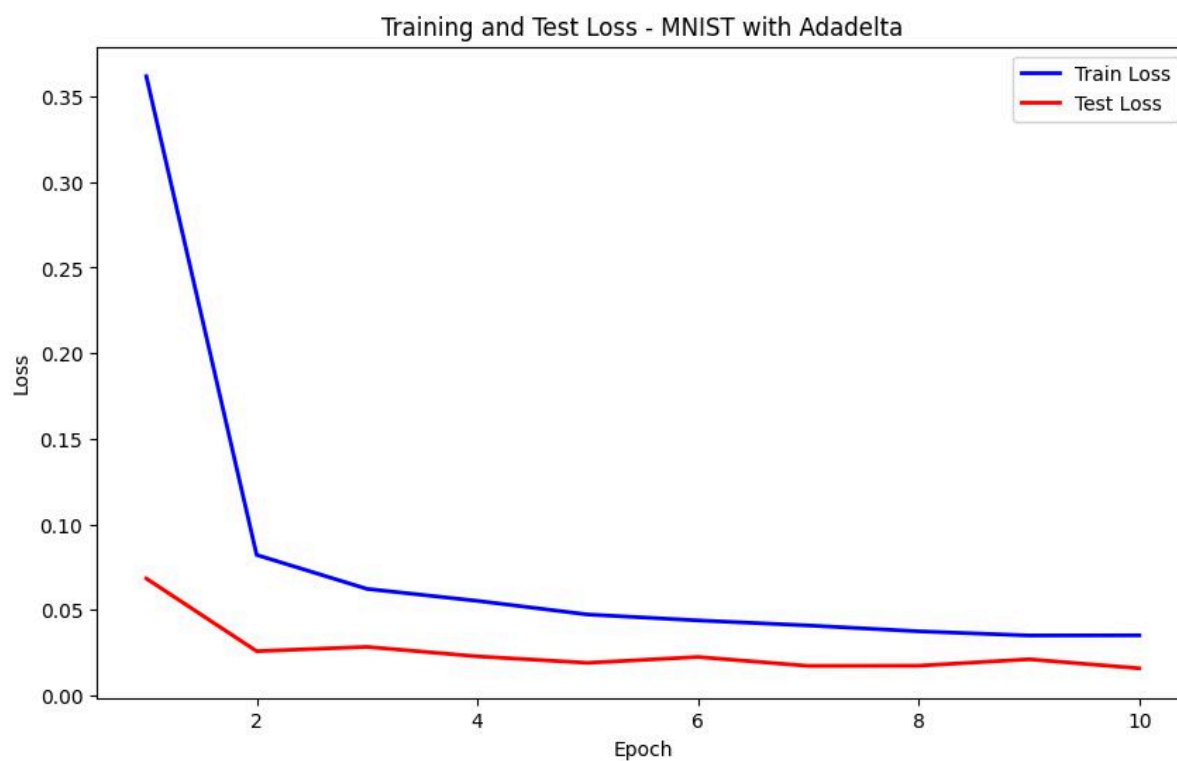
# -----
# Финальная оценка модели
# -----
print("Финальная оценка модели на тестовом наборе...")
final_test_loss, final_test_acc = evaluate(testloader)
print(f"Финальные результаты - Loss: {final_test_loss:.4f}, Accuracy:
{final_test_acc:.2f}%")

if __name__ == "__main__":
    main()
    print("Программа завершена!")

```

**Вывод:**





## State-of-art:

Table 7. Comparison of obtained results

	Fashion-MNIST		MNIST	
	Training accuracy	Testing accuracy	Training accuracy	Testing accuracy
Architecture 1	99.60%	89.65%	99.91%	98.48%
Architecture 2	92.02%	92.76%	98.86%	98.96%
Architecture 3	93.09%	93.56%	99.60%	99.37%
Architecture 4	93.17%	92.94%	99.02%	99.03%
Architecture 5 with Adam optimizer	93.12%	93.56%	99.48%	99.55%
Architecture 5 with RMSprop optimizer	92.67%	92.86%	99.26%	99.29%

[Ссылка на статью](#)

**Вывод:** научился конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения