Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «Обработка изображений в интеллектуальных системах»

Тема: «Конструирование моделей на базе предобученных нейронных сетей»

Выполнила:

Студентка 4 курса

Группы ИИ-24

Максимович А. И.

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.

Задание по вариантам

№ варианта	Выборка	Оптимизатор	Предобученная архитектура
11	MNIST	Adadelta	ResNet34

Код:

```
import os
import time
from datetime import datetime
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision
from torchvision import transforms, models

# ======================
```

```
# ПАРАМЕТРЫ ОБУЧЕНИЯ
# ======================
# Основные параметры
EPOCHS = 20
BATCH_SIZE = 128
LEARNING_RATE = 1.0  # Типичный learning rate для Adadelta
WEIGHT_DECAY = 1e-4
FREEZE_BACKBONE = False

# Пути
SAVE_DIR = 'checkpoints_mnist_resnet'
RESUME_TRAINING = False
RESUME_PATH = None
VISUALIZE_IMAGE = None  # Путь к изображению для предсказания

# Настройки данных
NUM_WORKERS = 4
INPUT_SIZE = 224
LOG_INTERVAL = 50  # Интервал логирования (в батчах)


# ======================
# МОДЕЛЬ И ФУНКЦИИ
# ======================
def create_resnet34(num_classes=10, freeze_backbone=False):
    """
    Создает модель ResNet34 с предобученными весами ImageNet
    и заменяет классификатор для нужного количества классов
    """
    # Загрузка предобученной модели ResNet34
    model = models.resnet34(weights=models.ResNet34_Weights.IMAGENET1K_V1)

    # Замена последнего полносвязного слоя
    in_features = model.fc.in_features
    model.fc = nn.Linear(in_features, num_classes)

    if freeze_backbone:
        # Заморозка всех параметров, кроме последнего слоя
        for name, param in model.named_parameters():
            if 'fc' not in name:
                param.requires_grad = False

    return model


def evaluate(model, loader, device, criterion):
    """Функция оценки модели на валидационной выборке"""
    model.eval()
    running_loss, correct, total = 0.0, 0, 0
    with torch.no_grad():
        for x, y in loader:
            x, y = x.to(device), y.to(device)
            out = model(x)
            loss = criterion(out, y)
            running_loss += loss.item() * x.size(0)
            preds = out.argmax(dim=1)
            correct += (preds == y).sum().item()
            total += x.size(0)
    return running_loss / total, 100.0 * correct / total
```

```
def predict_image(path, model, device, transform, input_size, classes):
    """Функция для предсказания на одном изображении"""
    img = Image.open(path).convert('RGB')
    img_resized = img.resize((input_size, input_size))
    x = transform(img_resized).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


# ======================
# ОСНОВНАЯ ФУНКЦИЯ
# ======================
def main():
    # Настройка устройства
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f'Using device: {device}')
    if torch.cuda.is_available():
        print(f'GPU: {torch.cuda.get_device_name(0)}')

    # Создание директории для сохранения
    os.makedirs(SAVE_DIR, exist_ok=True)

    # Параметры нормализации для ImageNet
    imagenet_mean = (0.485, 0.456, 0.406)
    imagenet_std = (0.229, 0.224, 0.225)

    # Преобразования для обучающей выборки
    train_transform = transforms.Compose([
        transforms.Grayscale(num_output_channels=3),  # Конвертируем в 3
канала
        transforms.RandomResizedCrop(INPUT_SIZE, scale=(0.8, 1.0)),
        transforms.RandomRotation(10),  # Небольшие повороты для цифр
        transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),  #
Случайные сдвиги
        transforms.ToTensor(),
        transforms.Normalize(imagenet_mean, imagenet_std)
    ])

    # Преобразования для тестовой выборки
    test_transform = transforms.Compose([
        transforms.Grayscale(num_output_channels=3),  # Конвертируем в 3
канала
        transforms.Resize(256),
        transforms.CenterCrop(INPUT_SIZE),
        transforms.ToTensor(),
        transforms.Normalize(imagenet_mean, imagenet_std)
    ])

    # Загрузка датасета MNIST
    print("Loading MNIST dataset...")
    train_set = torchvision.datasets.MNIST(
        root='./data',
        train=True,
        download=True,
        transform=train_transform
    )
    test_set = torchvision.datasets.MNIST(
```

```
        root='./data',
        train=False,
        download=True,
        transform=test_transform
    )

    # Создание загрузчиков данных
    train_loader = DataLoader(
        train_set,
        batch_size=BATCH_SIZE,
        shuffle=True,
        num_workers=NUM_WORKERS,
        pin_memory=True
    )
    test_loader = DataLoader(
        test_set,
        batch_size=BATCH_SIZE,
        shuffle=False,
        num_workers=NUM_WORKERS,
        pin_memory=True
    )

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

    # Создание модели
    print("Creating ResNet34 model...")
    model = create_resnet34(
        num_classes=10,
        freeze_backbone=FREEZE_BACKBONE
    ).to(device)

    # Вывод информации о модели
    print(f"Model: ResNet34")
    print(f"Trainable parameters: {sum(p.numel() for p in model.parameters()
if p.requires_grad)}")
    print(f"Total parameters: {sum(p.numel() for p in model.parameters())}")

    # Функция потерь и оптимизатор Adadelta
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adadelta(
        [p for p in model.parameters() if p.requires_grad],
        lr=LEARNING_RATE,
        weight_decay=WEIGHT_DECAY,
        rho=0.9,  # Параметр rho для Adadelta
        eps=1e-6  # Параметр epsilon для Adadelta
    )
    # Для Adadelta обычно не используется scheduler, но можно оставить
    scheduler = optim.lr_scheduler.StepLR(
        optimizer,
        step_size=max(5, EPOCHS // 2),
        gamma=0.1
    )

    # Возобновление обучения (если нужно)
    start_epoch = 1
    history = {'train_loss': [], 'test_loss': [], 'test_acc': []}

    if RESUME_TRAINING or RESUME_PATH:
        ckpt_path = RESUME_PATH
        if RESUME_TRAINING and ckpt_path is None:
```

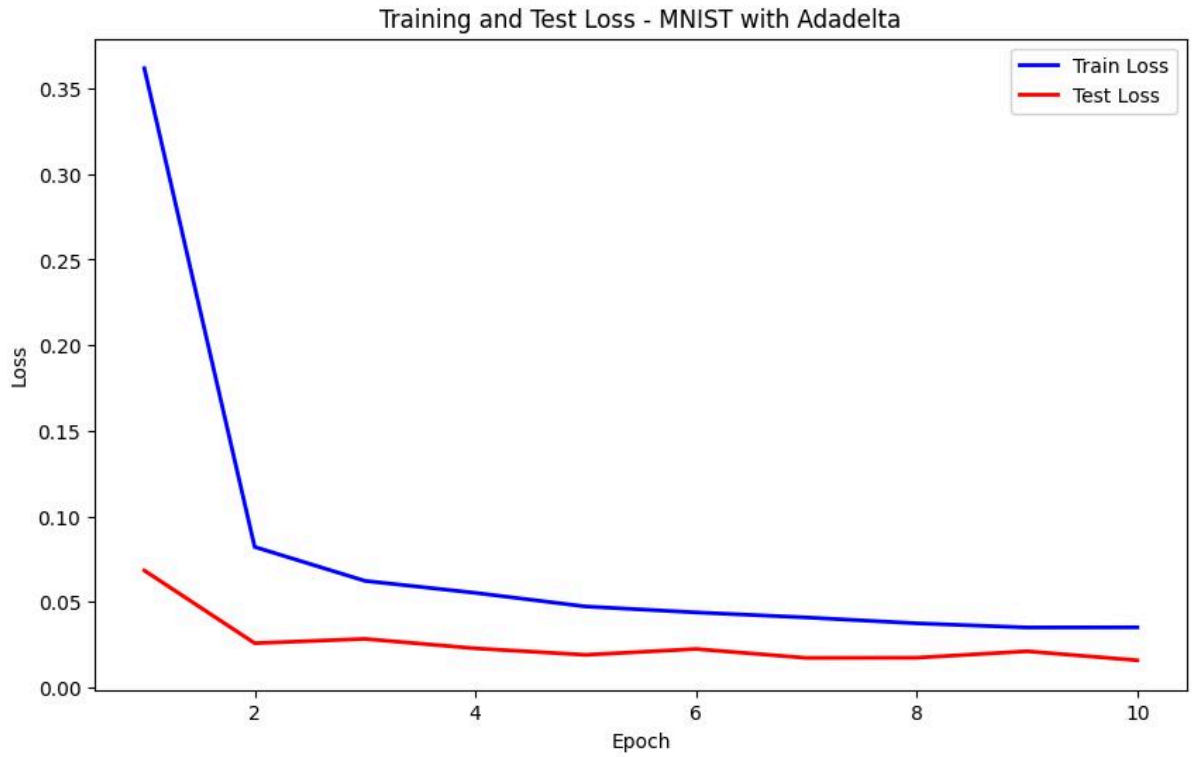
```
            files = [f for f in os.listdir(SAVE_DIR) if f.endswith('.pth')]
            if files:
                files = sorted(files, key=lambda x:
int(x.split('epoch')[-1].split('.')[0]))
                ckpt_path = os.path.join(SAVE_DIR, files[-1])

        if ckpt_path and os.path.isfile(ckpt_path):
            print(f"Loading checkpoint {ckpt_path} ...")
            ckpt = torch.load(ckpt_path, map_location=device)
            model.load_state_dict(ckpt['model_state'])
            optimizer.load_state_dict(ckpt['optimizer_state'])
            start_epoch = ckpt['epoch'] + 1
            if 'history' in ckpt:
                history = ckpt['history']
            print(f"Resumed from epoch {ckpt['epoch']}")
        else:
            print("⚠️ Checkpoint not found, starting from scratch")

    # Обучение модели
    print(f"Starting training for {EPOCHS} epochs...")
    global_start_time = time.time()

    for epoch in range(start_epoch, EPOCHS + 1):
        epoch_start_time = time.time()
        model.train()
        running_loss = 0.0

        # Логирование времени для батчей
        batch_times = []
        batch_start_time = time.time()

        for batch_idx, (xb, yb) in enumerate(train_loader, 1):
            xb, yb = xb.to(device), yb.to(device)
            optimizer.zero_grad()

            # Прямой проход
            logits = model(xb)
            loss = criterion(logits, yb)

            # Обратный проход
            loss.backward()
            optimizer.step()

            # Статистика
            running_loss += loss.item() * xb.size(0)

            # Логирование каждые LOG_INTERVAL батчей
            if batch_idx % LOG_INTERVAL == 0:
                batch_time = time.time() - batch_start_time
                batch_times.append(batch_time)

                # Прогноз оставшегося времени эпохи
                avg_batch_time = np.mean(batch_times[-10:]) if
len(batch_times) > 10 else batch_time
                remaining_batches = len(train_loader) - batch_idx
                eta_seconds = avg_batch_time * remaining_batches
                eta_str = time.strftime("%H:%M:%S", time.gmtime(eta_seconds))

                current_time = datetime.now().strftime("%H:%M:%S")
                print(f"[{current_time}] Epoch {epoch}/{EPOCHS} | "
                      f"Batch {batch_idx}/{len(train_loader)} | "
```

```
                        f"Loss: {loss.item():.6f} | "
                        f"Batch Time: {batch_time:.3f}s | "
                        f"ETA: {eta_str}")

                batch_start_time = time.time()

        # Статистика эпохи
        epoch_time = time.time() - epoch_start_time
        train_loss = running_loss / len(train_loader.dataset)
        test_loss, test_acc = evaluate(model, test_loader, device, criterion)

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

        print(f"Epoch {epoch}/{EPOCHS} completed in {epoch_time:.2f}s | "
              f"TrainLoss {train_loss:.4f} | TestLoss {test_loss:.4f} | "
              f"TestAcc {test_acc:.2f}%")

        # Сохранение контрольной точки
        checkpoint_path = os.path.join(SAVE_DIR,
f'resnet34_epoch{epoch}.pth')
        torch.save({
            'epoch': epoch,
            'model_state': model.state_dict(),
            'optimizer_state': optimizer.state_dict(),
            'history': history,
            'test_acc': test_acc
        }, checkpoint_path)
        print(f"Checkpoint saved: {checkpoint_path}")

    # Итоговое время обучения
    total_time = time.time() - global_start_time
    print(f'Training finished in {total_time / 60:.2f} minutes')

    # Построение графиков обучения
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(range(1, len(history['train_loss']) + 1), history['train_loss'],
label='train')
    plt.plot(range(1, len(history['test_loss']) + 1), history['test_loss'],
label='test', linestyle='--')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training and Test Loss - MNIST with ResNet34')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(range(1, len(history['test_acc']) + 1), history['test_acc'],
label='test acc', color='green')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.title('Test Accuracy - MNIST with ResNet34')
    plt.legend()

    plt.tight_layout()
    history_path = os.path.join(SAVE_DIR, 'training_history.png')
    plt.savefig(history_path, dpi=150)
    print(f'Saved history plot to {history_path}')
```

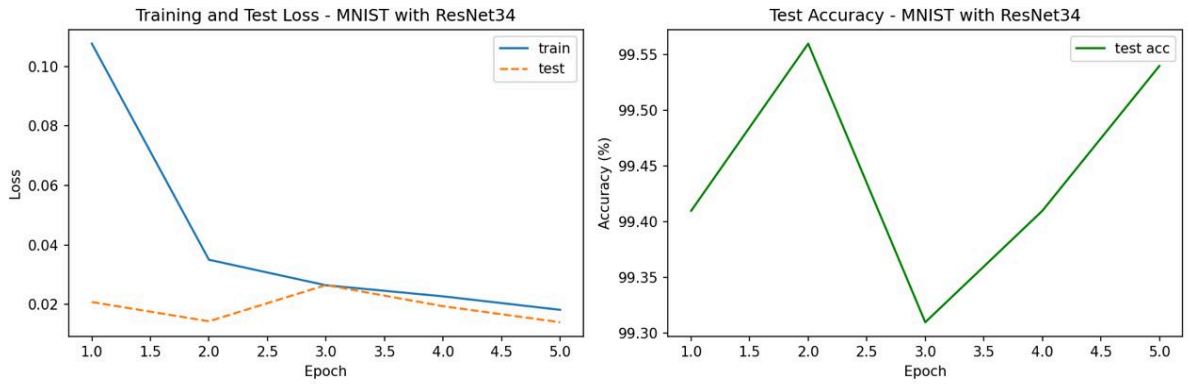
```
    # Визуализация (если указан путь к изображению)
    if VISUALIZE_IMAGE and os.path.exists(VISUALIZE_IMAGE):
        print(f"Making prediction for image: {VISUALIZE_IMAGE}")
        img, pred_idx, probs = predict_image(
            VISUALIZE_IMAGE,
            model,
            device,
            test_transform,
            INPUT_SIZE,
            classes
        )
        plt.figure(figsize=(6, 6))
        plt.imshow(img)
        plt.axis('off')
        plt.title(f'Prediction: {classes[pred_idx]} ({probs[pred_idx] *
100:.1f}%)')
        pred_path = os.path.join(SAVE_DIR, 'prediction.png')
        plt.savefig(pred_path, dpi=150, bbox_inches='tight')
        print(f'Saved prediction visualization to {pred_path}')
    elif VISUALIZE_IMAGE:
        print(f"Image {VISUALIZE_IMAGE} not found, skipping prediction.")


if __name__ == "__main__":
    main()
```

Вывод:

Л. Р. №1



Л. Р. № 2



State-of-art:

Regarding the models, ResNet-18 performed the best, followed by VGG-16, which is what we expected, as they are architecturally much more extensive networks. Our two custom CNNs did not perform as well, but still gave 91-92% accuracy.

We extensively trained CNN-7-BN for 120 epochs, recorded in Figure 2, expecting overtraining to occur. To our surprise, validation loss did not increase and validation accuracy did not decrease. We believe overtraining was prevented in CNN-7-BN due to batch normalization, dropout regularization, and max-pooling.

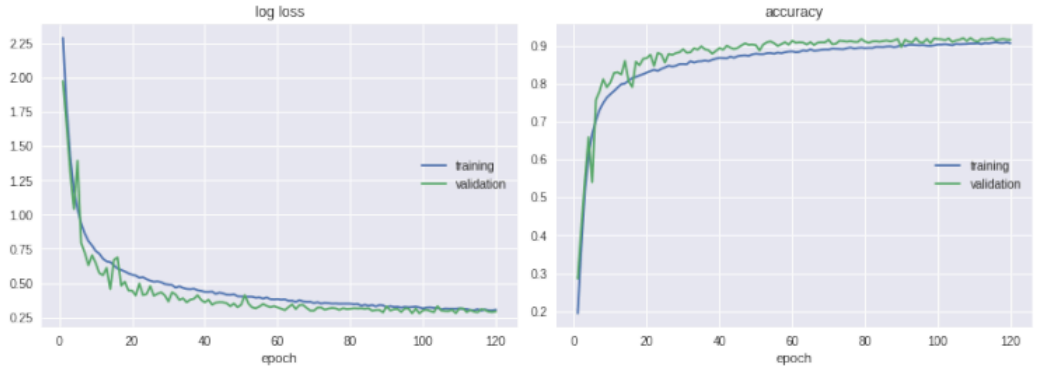


Figure 2: Batch normalization prevents overtraining CNN-7-BN

Possible directions for future investigation include training deeper ResNet models, such as ResNet-34 or ResNet-50. We would also like to perform ensembles, such as boosting CNN-6 or CNN-7-BN to reduce variance, and stacking on multiple ResNet models to minimize loss and achieve state-of-the-art accuracies.

[Ссылка на статью]

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