МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное образовательное учреждение
высшего образования «Южный федеральный университет»
(ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ)
Институт компьютерных технологий и информационной безопасности

Лабораторная работа №4

Классификация изображений с использованием сверточных нейронных сетей на PyTorch

По дисциплине:
Современные методы разработки ИИ решений

Выполнил: студент гр. КТмо1-13
Едуш А.В.

Проверила: Старший преподаватель каф. ИАСБ
Никашина П.О.

Таганрог 2025

Введение

Цель лабораторной работы:

- освоить методы предобработки изображений, техники аугментации данных и построение моделей классификации CNN с использованием PyTorch
- оценить их качество с помощью метрик и сравнить производительность с аугментацией и без.

Задание:

1 Загрузить датасет CIFAR-10

2 Выполнить предобработку данных и аугментацию:

- Нормализация
- Аугментация данных (RandomCrop, RandomHorizontalFlip, RandomRotation)

3 Обучить CNN модели:

- Архитектура ResNet-18
- Сравнить производительность с аугментацией и без

4 Оценить качество модели:

- Accuracy, метрики потерь
- Кривые обучения/валидации
- Сравнение производительности

Инструменты:

При проектировании системы были использованы следующие инструменты:

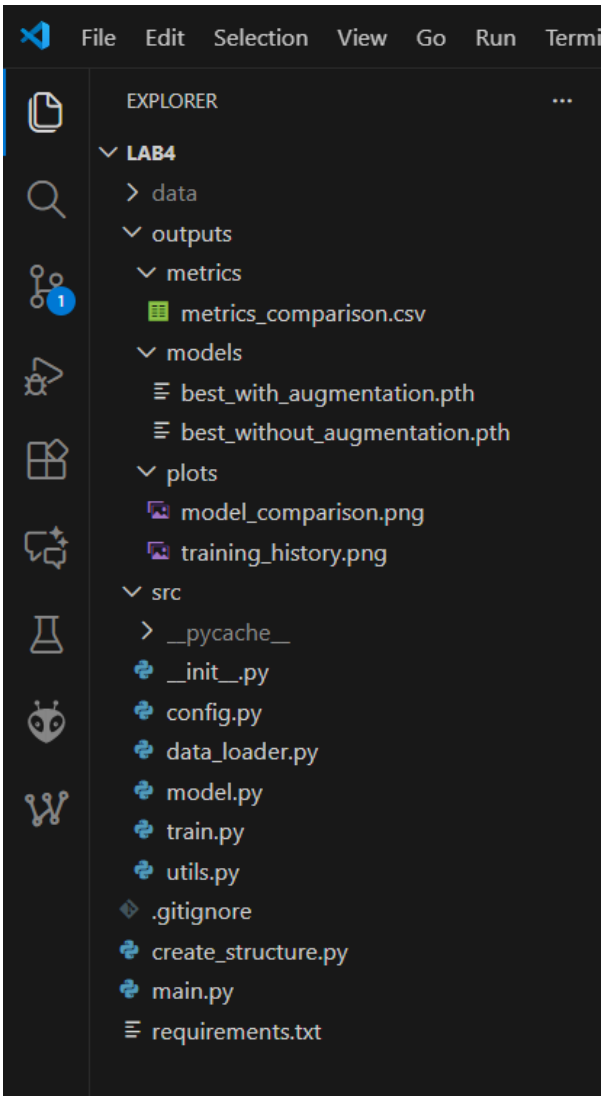
1 Редактор кода Visual Studio Code с установленными библиотеками и плагинами:

- pandas (загрузка и предобработка данных)
- numpy (численные операции и преобразования)
- torch (PyTorch основная библиотека для глубокого обучения)
- torchvision (дополнение для работы с изображениями и видео)
- tqdm (прогресс-бары для циклов и итераций)
- mathplotlib (анализ результатов и визуализация)

2 Git – ссылки на репозиторий:

- https://github.com/Anatoly-E/ImageRecognition-CNN-PyTorch

Структура проекта

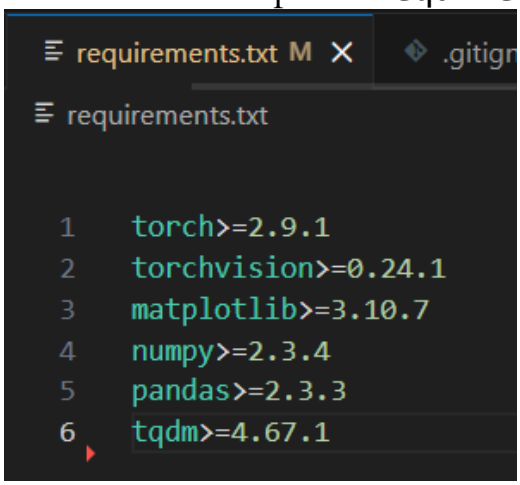


Основной файл **`main.py`**
Скрипты вынесены в папку **`src/`**
Выходные графики обучения моделей вынесены в **`plots/`**
Лучшие модели с аугментацией и без находятся в папке **`models/`**
Результат сравнения находится в таблице в папке **`metrics/`**

Задания

1. Загрузить датасет CIFAR-10

Актуальные версии устанавливаются в файле `requirements.txt`

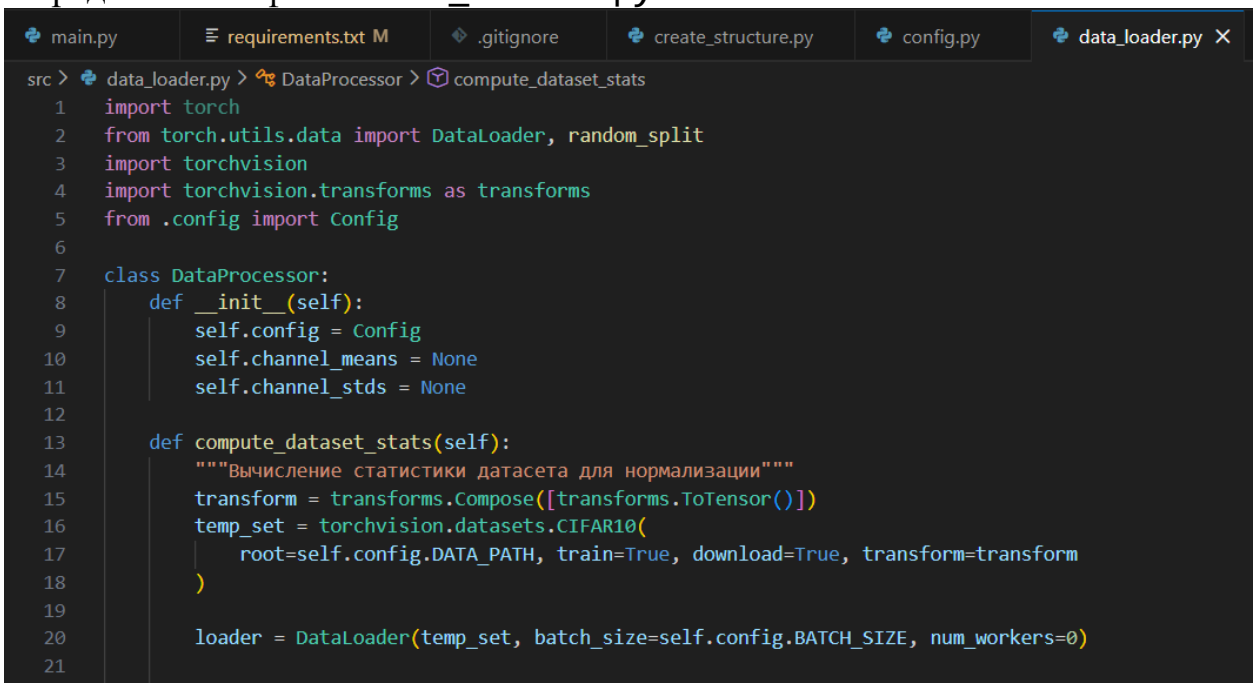


Командой в консоли:
`pip install -r requirements.txt`

```
PS E:\Management\Mara\ИИ\lab4> pip install -r requirements.txt
Requirement already satisfied: torch>=2.9.1 in c:\python\python314\lib\site-packages (from -r requirements.txt (line 1)) (2.9.1)
Requirement already satisfied: torchvision>=0.24.1 in c:\python\python314\lib\site-packages (from -r requirements.txt (line 2)) (0.24.1)
Requirement already satisfied: matplotlib>=3.10.7 in c:\python\python314\lib\site-packages (from -r requirements.txt (line 3)) (3.10.7)
Requirement already satisfied: numpy>=2.3.4 in c:\python\python314\lib\site-packages (from -r requirements.txt (line 4)) (2.3.4)
Requirement already satisfied: pandas>=2.3.3 in c:\python\python314\lib\site-packages (from -r requirements.txt (line 5)) (2.3.3)
Requirement already satisfied: tqdm>=4.67.1 in c:\python\python314\lib\site-packages (from -r requirements.txt (line 6)) (4.67.1)
Requirement already satisfied: filelock in c:\python\python314\lib\site-packages (from torch>=2.9.1->-r requirements.txt (line 1)) (3.20.0)
Requirement already satisfied: typing-extensions>=4.10.0 in c:\python\python314\lib\site-packages (from torch>=2.9.1->-r requirements.txt (line 1)) (4.15.0)
Requirement already satisfied: sympy>=1.13.3 in c:\python\python314\lib\site-packages (from torch>=2.9.1->-r requirements.txt (line 1)) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in c:\python\python314\lib\site-packages (from torch>=2.9.1->-r requirements.txt (line 1)) (3.5)
Requirement already satisfied: jinja2 in c:\python\python314\lib\site-packages (from torch>=2.9.1->-r requirements.txt (line 1)) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in c:\python\python314\lib\site-packages (from torch>=2.9.1->-r requirements.txt (line 1)) (2025.10.0)
```

CIFAR-10 - это классический датасет компьютерного зрения, содержащий 60 000 цветных изображений по 10 различным классам.

Определяется в файле `data_loader.py` в классе `DataProcessor`:



Основная загрузка происходит в функции `prepare_dataloaders()`:

onfig.py | data_loader.py | model.py | utils.py | train.py

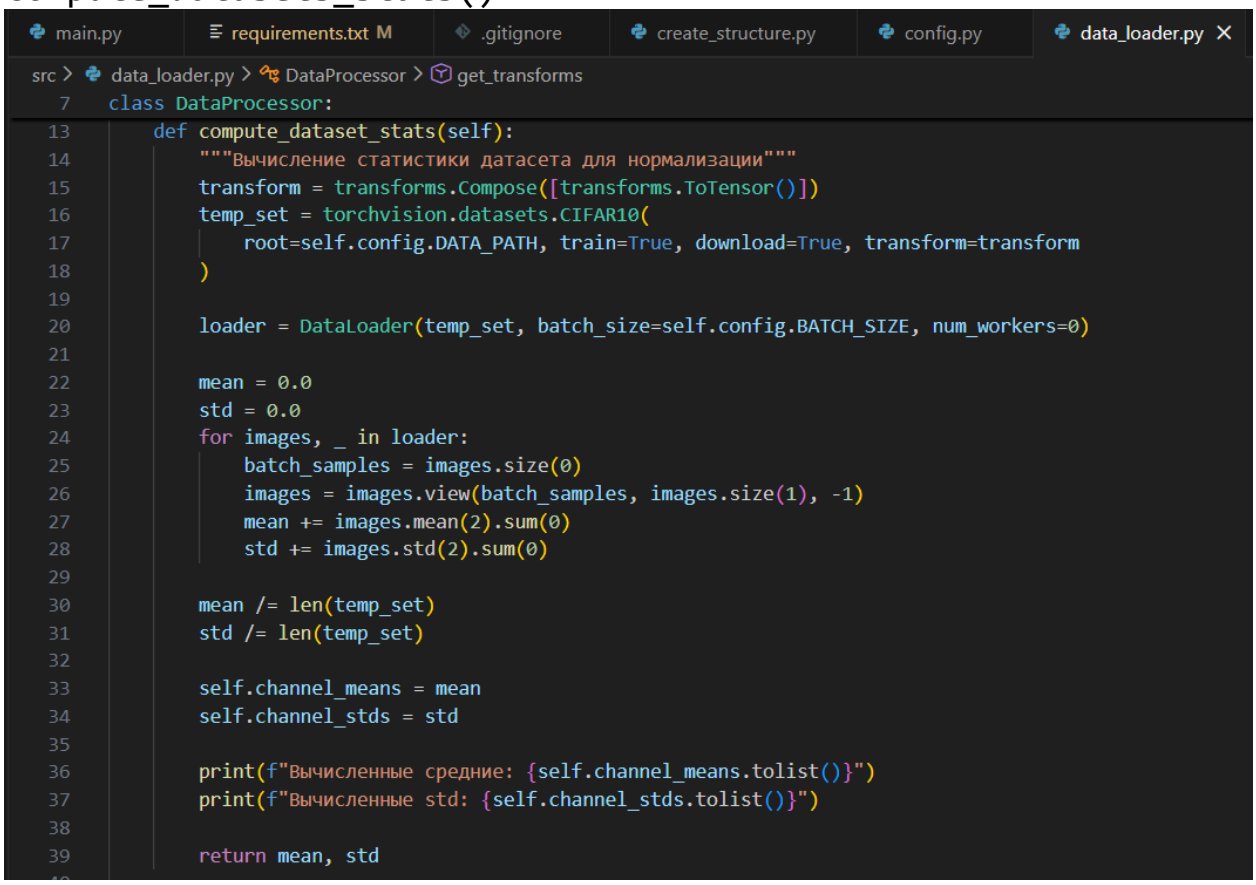
src > data_loader.py > DataProcessor > compute_dataset_stats

```
7   class DataProcessor:
41      def get_transforms(self):
61          return transform_basic, transform_augmented
62
63      def prepare_dataloaders(self):
64          """Подготовка всех DataLoader'ов"""
65          transform_basic, transform_augmented = self.get_transforms()
66
67          # Train set с аугментацией
68          trainset_augmented = torchvision.datasets.CIFAR10(
69              root=self.config.DATA_PATH, train=True, download=True,
70              transform=transform_augmented
71          )
72
73          # Train set без аугментации
74          trainset_basic = torchvision.datasets.CIFAR10(
75              root=self.config.DATA_PATH, train=True, download=True,
76              transform=transform_basic
77          )
78
79          # Test set (всегда без аугментации)
80          testset = torchvision.datasets.CIFAR10(
81              root=self.config.DATA_PATH, train=False, download=True,
82              transform=transform_basic
83          )
```

2. Выполнить предобработку данных и аугментацию

Нормализация

Происходит в файле `data_loader.py` в функции `compute_datasets_stats()`



Где:

- `mean` – это среднее арифметическое значение пикселей по каждому каналу (R, G, B)
- `std` – мера разброса значений пикселей вокруг среднего

После нормализации данные имеют:

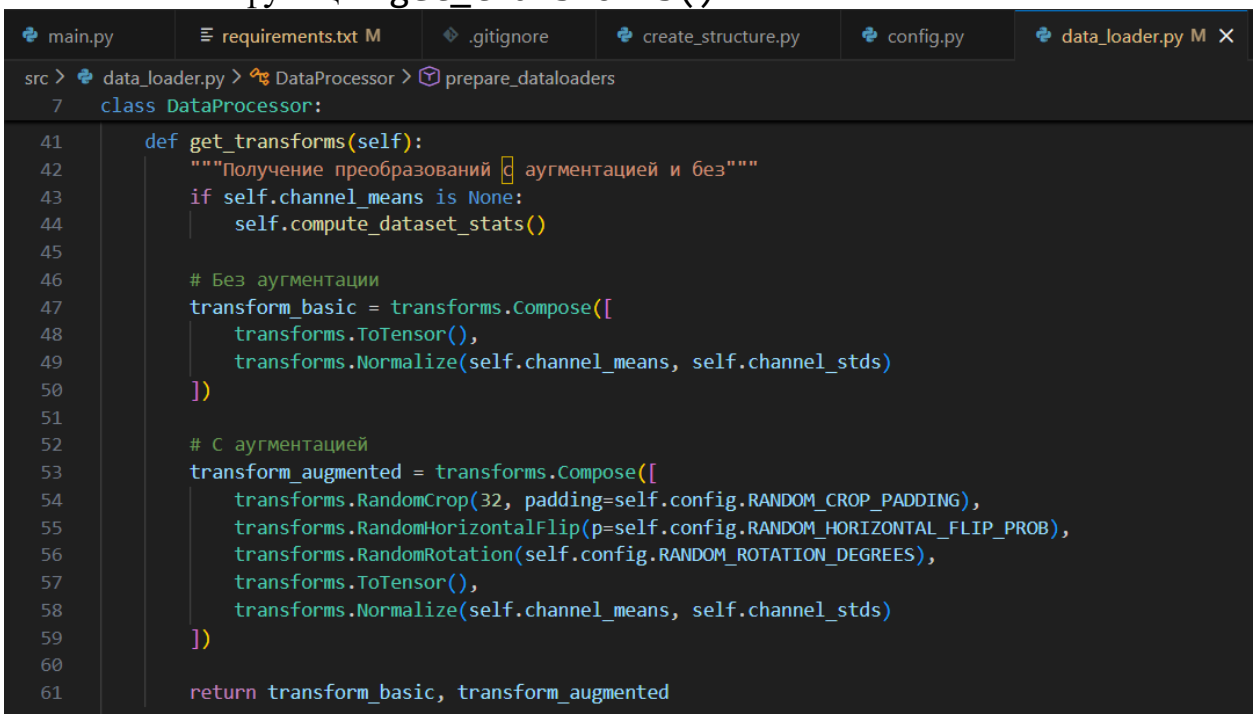
- Нулевое среднее (центрированы вокруг 0)
- Единичную дисперсию (стандартное отклонение = 1)
- Стандартное распределение (легче для обучения нейросетей)

Аугментация данных (RandomCrop, RandomHorizontalFlip, RandomRotation)

Нейронные сети легко переобучаются - запоминают конкретные примеры вместо изучения общих признаков.

Решение: Аугментация искусственно увеличивает разнообразие данных, заставляя модель учиться инвариантности к определенным преобразованиям.

Реализовано в функции `get_transforms()`



RandomCrop (Случайное обрезание изображения)

К примеру, берётся изображение 40x40 и из него берётся случайный фрагмент 32x32. В случайных изображениях объект может находиться где угодно, вверху изображения или, например, справа.

Если в тренировочных данных все объекты были в центре, без RandomCrop модель может не распознать искомый объект сбоку.

RandomHorizontalFlip (Случайное горизонтальное отражение)

Например, делаем изображение с вероятностью 50% отражённое по горизонтали.

Практическая польза – удвоение данных.

А так как многие объекты в мире часто симметричны – добавляет инвариантность к зеркальности. Машина может ехать в любую сторону, например.

RandomRotation (Случайный поворот)

Делает случайный поворот на случайный угол.

Практическая польза – модель учится распознавать объекты под разными углами.

Так как объекты в мире редко идеально выровнены, фото может быть сделано под углом.

В итоге аугментация делает модель более умной и универсальной.

3. Обучить CNN модели

Архитектура ResNet-18

Работа с моделью Вынесена в отдельный файл `model.py`

main.py | model.py | requirements.txt M | .gitignore | create_structure.py | config.py

src > model.py > ...

```python
import torch
import torch.nn as nn
import torchvision
from .config import Config

class ResNet18(nn.Module):
    def __init__(self, num_classes=Config.NUM_CLASSES):
        super(ResNet18, self).__init__()

        # ИСПРАВЛЕННЫЙ КОД - без предупреждений
        self.resnet = torchvision.models.resnet18(weights=None)

        # Адаптируем под CIFAR-10 (32x32 вместо 224x224)
        self.resnet.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.resnet.maxpool = nn.Identity()  # Убираем maxpool
        self.resnet.fc = nn.Linear(512, num_classes)

    def forward(self, x):
        return self.resnet(x)

def create_model(device=Config.DEVICE):
    """Создание и инициализация модели"""
    model = ResNet18().to(device)
    print(f"Модель создана и перемещена на {device}")
    print(f"Количество параметров: {sum(p.numel() for p in model.parameters()):,}")
    return model
```

Сравнить производительность с аугментацией и без.

Обучение модели с аугментацией заняло 198 минут и 7 секунд.

```
Epoch 15/15
--------------------------------------------------
✅ Train Loss: 0.2610 Acc: 0.9091
📊 Val Loss: 0.3465 Acc: 0.8810
💾 Сохранена лучшая модель: ./outputs/models\best_with_augmentation.pth

🏁 Обучение завершено за 198m 7s
🎯 Лучшая точность на валидации: 0.8810
```

Обучение модели без аугментаций заняло 186 минут и 6 секунд.

```
Epoch 15/15
--------------------------------------------------
✅ Train Loss: 0.0037 Acc: 0.9997
📊 Val Loss: 0.6084 Acc: 0.8531
💾 Сохранена лучшая модель: ./outputs/models\best_without_augmentation.pth

🏁 Обучение завершено за 186m 6s
🎯 Лучшая точность на валидации: 0.8531
```

Характеристики обучения моделей:

- Архитектура: ResNet-18 (адаптированная для CIFAR-10)
- Параметры обучения: 15 эпох, batch_size=128, Adam optimizer

Параметры вынесены в отдельный файл `config.py`

main.py | model.py | requirements.txt M | .gitignore | create_structure.py | config.py

src > config.py > Config

```python
import torch
import torchvision

class Config:
    # Пути
    DATA_PATH = './data'
    MODEL_SAVE_PATH = './outputs/models'
    PLOT_SAVE_PATH = './outputs/plots'
    METRICS_SAVE_PATH = './outputs/metrics'

    # Параметры данных
    BATCH_SIZE = 128
    NUM_WORKERS = 0  # Для избежания проблем с multiprocessing
    TRAIN_VAL_SPLIT = 0.8

    # Параметры модели
    NUM_CLASSES = 10
    LEARNING_RATE = 0.001
    WEIGHT_DECAY = 1e-4
    NUM_EPOCHS = 15

    # Аугментации
    RANDOM_CROP_PADDING = 4
    RANDOM_HORIZONTAL_FLIP_PROB = 0.5
    RANDOM_ROTATION_DEGREES = 10

    # Device
    DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # CIFAR-10 classes
    CLASSES = ('plane', 'car', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck')

```

Сравнение производительности:

- Модель с аугментацией: 198 минут 7 секунд
- Модель без аугментации: 186 минут 6 секунд
- Замедление из-за аугментации: ~6.5%
- Среднее время на эпоху: 13.2 минут против 12.4 минут

Наблюдения:

- Аугментация данных увеличивает время обучения на 12 минут
- Дополнительная вычислительная нагрузка обусловлена онлайн-преобразованиями
- Ожидается, что это замедление окупится улучшением качества модели (проверим в пункте 4)

4. **Оценить качество модели:**
   - Accuracy, метрики потерь
   - Кривые обучения/валидации

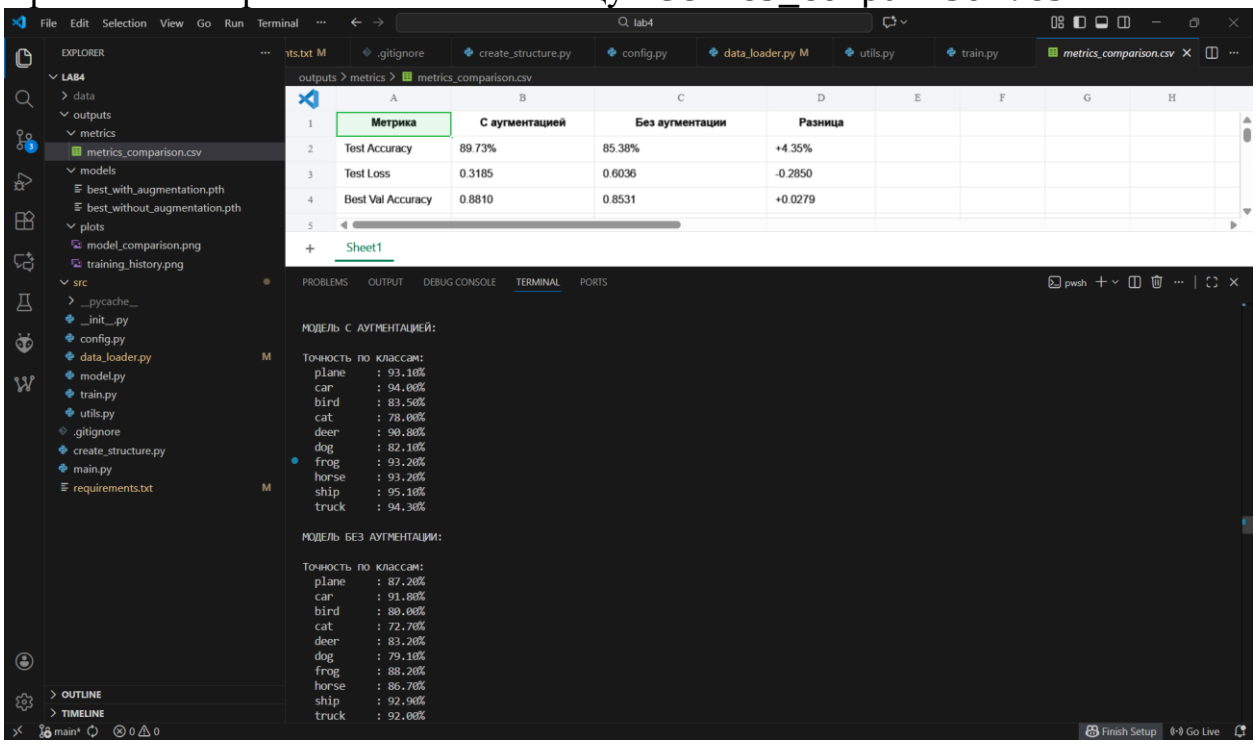
```
🧪 Тестирование модели: С аугментацией
----------------------------------------
📊 Test Loss: 0.3185
🎯 Test Accuracy: 89.73%
✅ Correct/Total: 8973/10000
```

```
🧪 Тестирование модели: Без аугментации
----------------------------------------
----------------------------------------
📊 Test Loss: 0.6036
🎯 Test Accuracy: 85.38%
✅ Correct/Total: 8538/10000
```

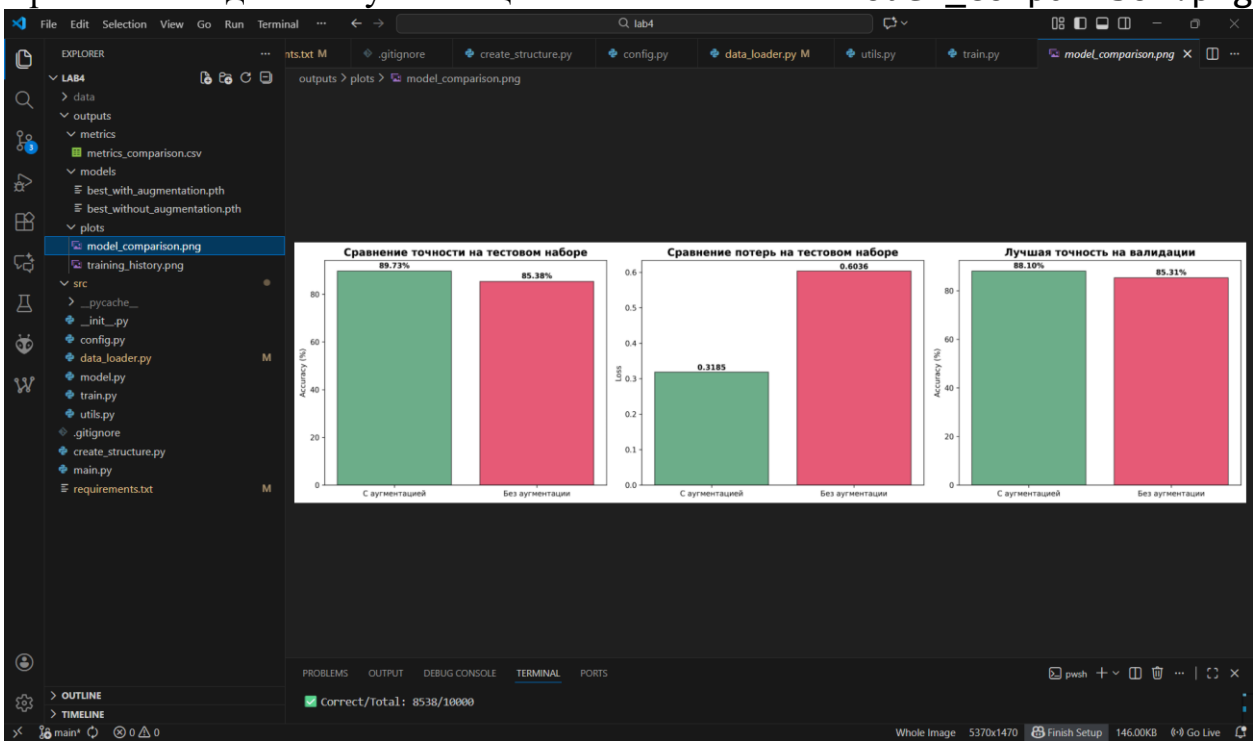
Аугментация данных положительно повлияла на модель:

- улучшение точности: +4.35%
- относительное улучшение: +5.09%

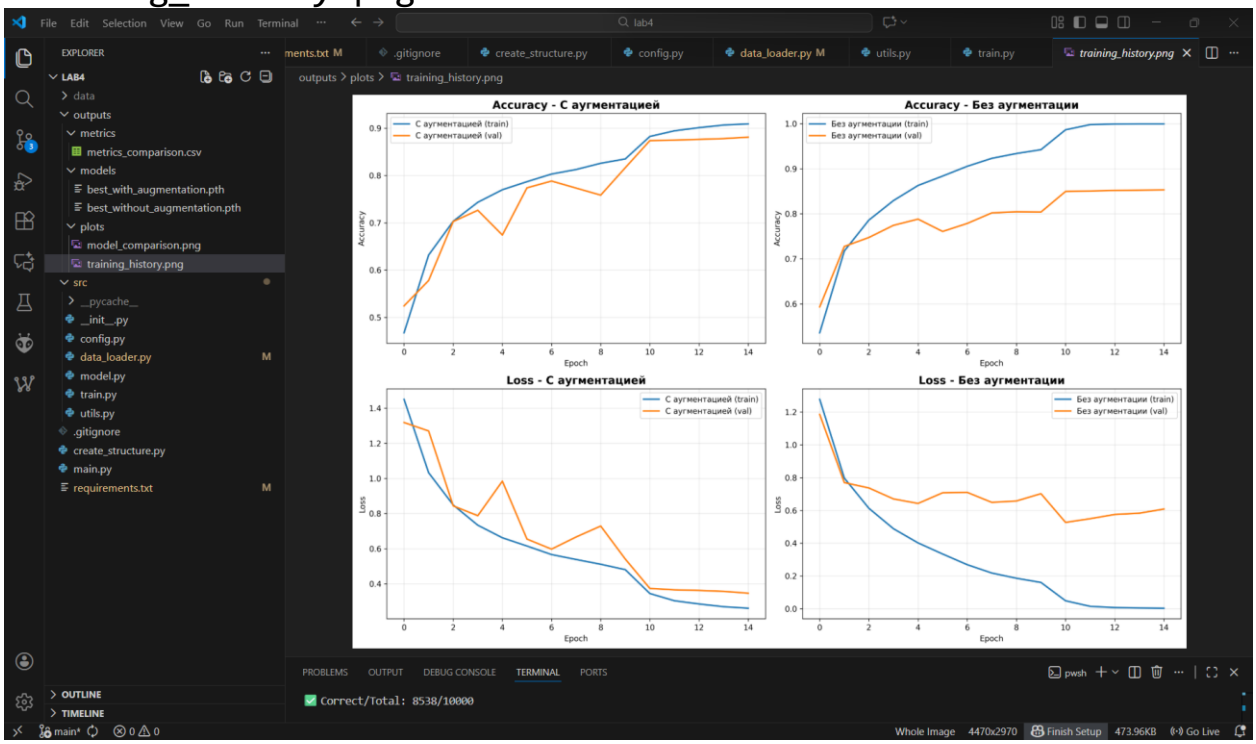
Сравнение метрик вынесено в таблицу `metrics_comparison.csv`



Сравнение моделей с аугментацией и без показано в `model_comparison.png`



На графики выведены accuracy и loss по эпохам в файле `training_history.png`



Выводы

Модель с аугментацией данных демонстрирует:

- В целом, плавную и стабильную сходимость - кривые обучения и потерь имеют монотонный характер
- Эффективную регуляризацию - малый разрыв между тренировочной и валидационной точностью
- Устойчивость к переобучению - validation loss продолжает уменьшаться на протяжении всех эпох
- Консистентное улучшение - обе метрики (accuracy и loss) улучшаются синхронно

Модель без аугментации показывает:

- Признаки переобучения - увеличивающийся разрыв между train и validation accuracy
- Риск преждевременной сходимости - validation loss стагнирует в последних эпохах

Заключение

В ходе лабораторной работы были успешно выполнены все поставленные задания

В ходе лабораторной работы успешно освоены методы предобработки изображений, техники аугментации данных и построение моделей классификации CNN с использованием PyTorch. Проведено комплексное сравнение производительности моделей с аугментацией и без.

Полученные результаты наглядно демонстрируют эффективность аугментации данных для улучшения обобщающей способности сверточных нейронных сетей и подтверждают их важность в практических задачах компьютерного зрения.