

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

Лабораторная работа №3

По дисциплине «Обработка изображений в интеллектуальных системах»

Тема: «Обучение детекторов объектов»

Выполнил:

Студент 4 курса

Группы ИИ-24

Супрунович И. С.

Проверила:

Андренко К. В.

Брест 2025

Цель: осуществлять обучение нейросетевого детектора для решения задачи обнаружения заданных объектов

Общее задание

1. Базируясь на своем варианте, ознакомится с выборкой для обучения детектора, выполнить необходимые преобразования данных для организации процесса обучения (если это нужно!);
2. Для заданной архитектуры нейросетевого детектора организовать процесс обучения для своей выборки. Оценить эффективность обучения на тестовой выборке (mAP);
3. Реализовать визуализацию работы детектора из пункта 1 (обнаружение знаков на отдельных фотографиях из сети Интернет);
4. Оформить отчет по выполненной работе, залить исходный код и отчет в соответствующий репозиторий на github.

Задание по вариантам

№ в-а	Дететктор	Датасет
5	YOLOv10m	Счетчики расхода воды: https://universe.roboflow.com/koer3741-gmail-com/watermeteramrv2/dataset/1

Код:

```
import os
import shutil
import cv2
from ultralytics import YOLO
import yaml
import random
import torch
import torchvision

# ===== КОНФИГУРАЦИЯ =====
DOWNLOADED_DATASET_PATH = './watermeteramrv2' # Путь к распакованной папке

# Настройки модели и обучения
MODEL_TYPE = 'yolov10m.pt'
EPOCHS = 50
IMGSZ = 640
BATCH = 16
OPTIMIZER = 'AdamW'
MODEL_NAME = 'yolov10m_water_meter'

# Куда подготовить данные
YOLO_DATASET = 'watermeteramrv2'

# Пути к тестовым изображениям для демонстрации
```

```

TEST_IMAGES =
["0e0379486aae0438e256cc54a8477c80_JPG.rf.61b240dea3018173f9049b451f6feb98.jp
g",
"2e38bb1d1c9f653aa4a6be622313c77a_JPG.rf.4036e973f9d4710980a6d653ab34c929.jpg
"] # Добавьте свои тестовые изображения

# =====

def check_gpu_availability():
    """Проверка доступности GPU и вывод информации"""
    print("=" * 50)
    print("ПРОВЕРКА ДОСТУПНОСТИ GPU")
    print("=" * 50)

    # Проверяем доступность CUDA
    cuda_available = torch.cuda.is_available()
    print(f"CUDA доступен: {cuda_available}")

    if cuda_available:
        gpu_count = torch.cuda.device_count()
        print(f"Количество GPU: {gpu_count}")

        for i in range(gpu_count):
            gpu_name = torch.cuda.get_device_name(i)
            gpu_memory = torch.cuda.get_device_properties(i).total_memory /
1024 ** 3 # в GB
            print(f"GPU {i}: {gpu_name}")
            print(f"Память: {gpu_memory:.2f} GB")

            # Устанавливаем устройство по умолчанию
            device = torch.device('cuda:0')
            print(f"Используется устройство: {device}")
        else:
            device = torch.device('cpu')
            print("ВНИМАНИЕ: CUDA не доступен, обучение будет на CPU!")
            print("Это может занять значительно больше времени.")

    print()
    return device

def setup_dataset_from_download():
    """Настройка датасета, скачанного вручную с RoboFlow"""
    print("Настройка скачанного датасета...")

    # Проверяем, существует ли папка с датасетом
    if not os.path.exists(DOWNLOADED_DATASET_PATH):
        raise FileNotFoundError(
            f"Папка с датасетом не найдена: {DOWNLOADED_DATASET_PATH}.
Убедись, что ты распаковал архив.")

    # Читаем конфигурационный файл датасета
    original_data_yaml_path = os.path.join(DOWNLOADED_DATASET_PATH,
'data.yaml')
    with open(original_data_yaml_path, 'r') as f:
        data_config = yaml.safe_load(f)

    # Создаем структуру папок для YOLO
    yolo_train_img_dir = os.path.join(YOLO_DATASET, 'train', 'images')
    yolo_train_lbl_dir = os.path.join(YOLO_DATASET, 'train', 'labels')

```

```

yolo_val_img_dir = os.path.join(YOLO_DATASET, 'valid', 'images')
yolo_val_lbl_dir = os.path.join(YOLO_DATASET, 'valid', 'labels')
yolo_test_img_dir = os.path.join(YOLO_DATASET, 'test', 'images')
yolo_test_lbl_dir = os.path.join(YOLO_DATASET, 'test', 'labels')

for dir_path in [yolo_train_img_dir, yolo_train_lbl_dir,
                 yolo_val_img_dir, yolo_val_lbl_dir,
                 yolo_test_img_dir, yolo_test_lbl_dir]:
    os.makedirs(dir_path, exist_ok=True)

# Копируем данные из скачанного датасета
def copy_split(split_name, yolo_img_dir, yolo_lbl_dir):
    # Используем пути, указанные в data.yaml скачанного датасета
    original_img_dir = os.path.join(DOWNLOADED_DATASET_PATH,
data_config[split_name])
    original_lbl_dir = original_img_dir.replace('images', 'labels')

    if os.path.exists(original_img_dir):
        # Копируем изображения
        for img_file in os.listdir(original_img_dir):
            if img_file.lower().endswith(('.png', '.jpg', '.jpeg')):
                src_img = os.path.join(original_img_dir, img_file)
                dst_img = os.path.join(yolo_img_dir, img_file)
                shutil.copy2(src_img, dst_img)

                # Копируем соответствующие файлы разметки
                lbl_file = os.path.splitext(img_file)[0] + '.txt'
                src_lbl = os.path.join(original_lbl_dir, lbl_file)
                dst_lbl = os.path.join(yolo_lbl_dir, lbl_file)
                if os.path.exists(src_lbl):
                    shutil.copy2(src_lbl, dst_lbl)
                else:
                    print(f"Предупреждение: файл разметки {src_lbl} не
найдён")

# Копируем данные для обучения, валидации и тестирования
copy_split('train', yolo_train_img_dir, yolo_train_lbl_dir)
copy_split('val', yolo_val_img_dir, yolo_val_lbl_dir)
copy_split('test', yolo_test_img_dir, yolo_test_lbl_dir)

# Создаем обновленный data.yaml файл
updated_data_yaml = {
    'train': os.path.abspath(yolo_train_img_dir),
    'val': os.path.abspath(yolo_val_img_dir),
    'test': os.path.abspath(yolo_test_img_dir),
    'nc': data_config['nc'],
    'names': data_config['names']
}

yolo_data_yaml_path = os.path.join(YOLO_DATASET, 'data.yaml')
with open(yolo_data_yaml_path, 'w') as f:
    yaml.dump(updated_data_yaml, f, default_flow_style=False)

print("Настройка датасета завершена!")
print(f"Количество классов: {data_config['nc']}")
print(f"Классы: {data_config['names']}")

return data_config['nc'], data_config['names'], yolo_data_yaml_path

def train_yolo_model(num_classes, class_names, data_yaml_path, device):

```

```

    """Обучение модели YOLOv10 на GPU"""
    print(f"\nОбучение модели YOLOv10 для {num_classes} классов:
{class_names}...")
    print(f"Обучение на устройстве: {device}")

    # Загружаем модель YOLOv10
    model = YOLO(MODEL_TYPE)

    # Определяем параметры устройства для обучения
    if str(device).startswith('cuda'):
        # Для GPU указываем device=0 или список GPU
        device_param = 0
        print("Используется GPU для обучения")
    else:
        device_param = 'cpu'
        print("Используется CPU для обучения")

    # Обучаем модель с явным указанием устройства
    results = model.train(
        data=data_yaml_path,
        epochs=EPOCHS,
        imgsz=IMGSZ,
        batch=BATCH,
        name=MODEL_NAME,
        optimizer=OPTIMIZER,
        patience=10, # Ранняя остановка при отсутствии улучшений
        lr0=0.001, # Начальная скорость обучения
        device=device_param, # Явно указываем устройство
        workers=4, # Количество workers для загрузки данных
        amp=True, # Включаем автоматическое смешение точности для экономии
памяти
    )

    print("Обучение завершено!")
    return model

def evaluate_model(model, data_yaml_path, device):
    """Оценка модели на тестовых данных"""
    print(f"\nОценка модели...")

    # Определяем параметры устройства для валидации
    if str(device).startswith('cuda'):
        device_param = 0
    else:
        device_param = 'cpu'

    metrics = model.val(
        data=data_yaml_path,
        split='test',
        device=device_param # Явно указываем устройство для валидации
    )

    print(f"mAP50-95: {metrics.box.map:.4f}")
    print(f"mAP50: {metrics.box.map50:.4f}")
    return metrics

def visualize_detection(model, image_path, output_dir="detection_results"):
    """Визуализация детекции на изображении"""
    print(f"\nВизуализация детекции на {image_path}...")

```

```

os.makedirs(output_dir, exist_ok=True)

results = model(image_path)

# Сохраняем результат
output_path = os.path.join(output_dir,
f"detected_{os.path.basename(image_path)}")
results[0].save(filename=output_path)

print(f"Результат сохранен в: {output_path}")
return results, output_path

def demonstrate_on_test_images(model):
    """Демонстрация работы модели на тестовых изображениях"""
    print("\nДемонстрация работы детектора на тестовых изображениях...")

    # Используем изображения из тестовой выборки для демонстрации
    test_images_dir = os.path.join(YOLO_DATASET, 'test', 'images')

    if os.path.exists(test_images_dir):
        test_images = [f for f in os.listdir(test_images_dir) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))]

        if test_images:
            # Выбираем несколько случайных изображений для демонстрации
            demo_images = random.sample(test_images, min(3,
len(test_images)))

            for img_name in demo_images:
                img_path = os.path.join(test_images_dir, img_name)
                visualize_detection(model, img_path)
            else:
                print("В тестовой директории нет изображений")
        else:
            print(f"Тестовая директория не найдена: {test_images_dir}")

    # Также пробуем на пользовательских изображениях
    for test_img in TEST_IMAGES:
        if os.path.exists(test_img):
            visualize_detection(model, test_img)
        else:
            print(f"Тестовое изображение не найдено: {test_img}")

def monitor_gpu_usage():
    """Мониторинг использования GPU во время обучения"""
    if torch.cuda.is_available():
        print("\nМониторинг GPU:")
        for i in range(torch.cuda.device_count()):
            allocated = torch.cuda.memory_allocated(i) / 1024 ** 3
            reserved = torch.cuda.memory_reserved(i) / 1024 ** 3
            print(f"GPU {i}: выделено {allocated:.2f} GB, зарезервировано
{reserved:.2f} GB")

def main():
    """Основная функция"""
    try:
        print("=" * 50)

```

```

print("Лабораторная работа №3. Обучение детекторов объектов")
print("Версия с поддержкой GPU")
print("=" * 50)

# 0. Проверка доступности GPU
device = check_gpu_availability()

# 1. Настройка датасета
num_classes, class_names, yolo_data_yaml_path =
setup_dataset_from_download()

# 2. Обучение модели
print("\n" + "=" * 50)
print("ЭТАП 2: ОБУЧЕНИЕ МОДЕЛИ")
print("=" * 50)

# Мониторинг памяти до обучения
monitor_gpu_usage()

model = train_yolo_model(num_classes, class_names,
yolo_data_yaml_path, device)

# Мониторинг памяти после обучения
monitor_gpu_usage()

# 3. Оценка модели
print("\n" + "=" * 50)
print("ЭТАП 3: ОЦЕНКА МОДЕЛИ")
print("=" * 50)
metrics = evaluate_model(model, yolo_data_yaml_path, device)

# 4. Демонстрация работы
print("\n" + "=" * 50)
print("ЭТАП 4: ДЕМОНСТРАЦИЯ РАБОТЫ")
print("=" * 50)
demonstrate_on_test_images(model)

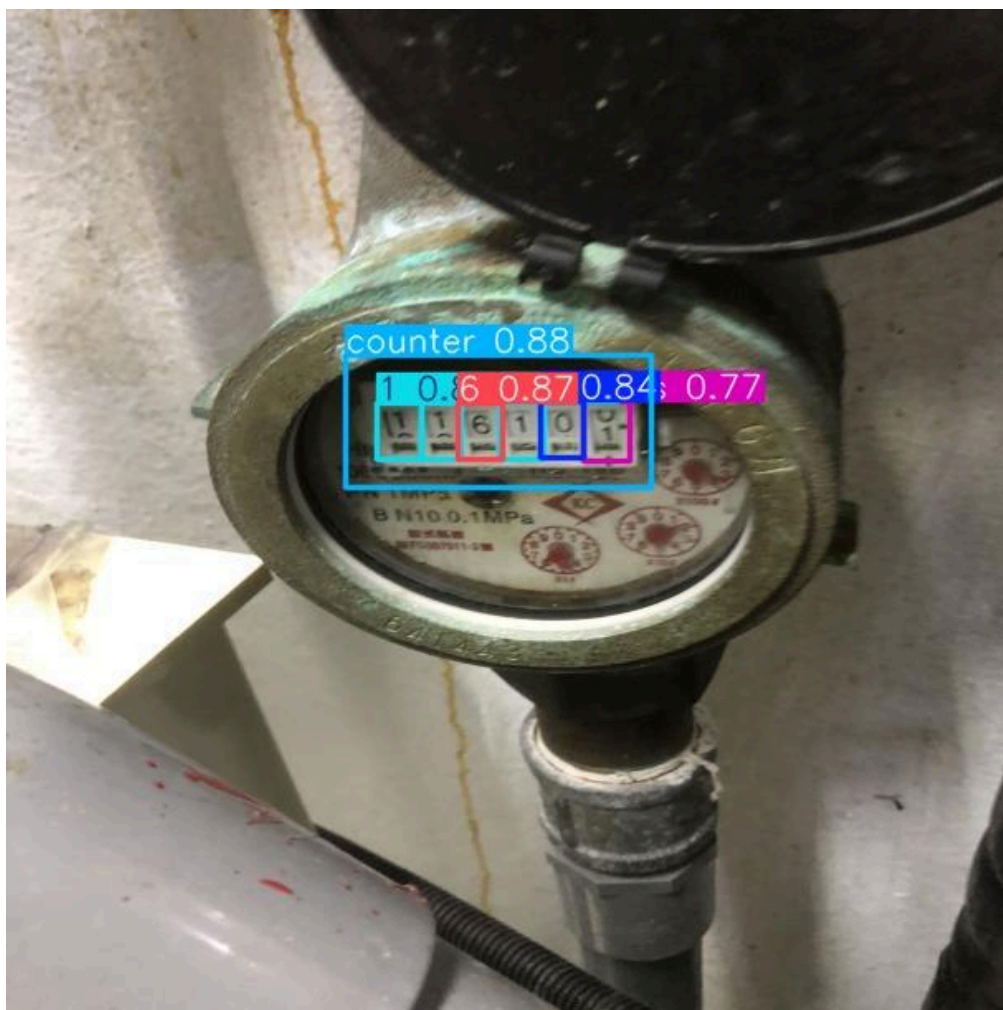
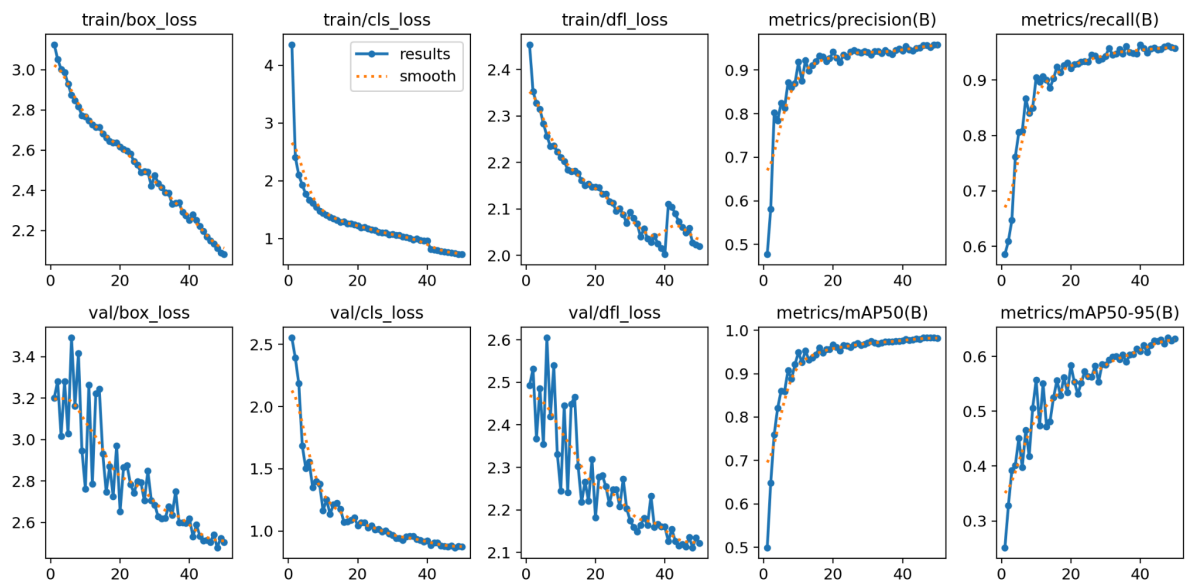
print("\n" + "=" * 50)
print("Лабораторная работа завершена!")
print("=" * 50)
print(f"Модель сохранена в: runs/detect/{MODEL_NAME}/")
print(f"mAP50-95: {metrics.box.map:.4f}")
print(f"mAP50: {metrics.box.map50:.4f}")
print(f"Количество классов: {num_classes}")
print(f"Классы: {class_names}")
print(f"Использованное устройство: {device}")

except Exception as e:
    print(f"\nПроизошла ошибка: {e}")
    print("\nУбедись, что:")
    print("1. Ты скачал датасет с RoboFlow в формате YOLOv8")
    print("2. Распаковал архив в папку с проектом")
    print("3. Указал правильный путь в переменной
DOWNLOADED_DATASET_PATH")
    print("4. Установлены все зависимости: pip install ultralytics
opencv-python pyyaml torch torchvision")
    print("5. Для GPU: установлены CUDA и cuDNN совместимых версий")

if __name__ == "__main__":
    main()

```

Вывод:



Вывод: осуществлять обучение нейросетового детектора для решения задачи обнаружения заданных объектов