МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №3

Специальность ИИ24

Выполнил

Д. С. Сухаревич,

студент группы ИИ24

Проверила

К. В. Андренко,

Преподаватель-стажер кафедры ИИТ,

« » 2025 г.

Брест 2025

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

**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.

Сравнить полученные результаты с результатами, полученными на кастомных архитектурах из ЛР №1;

**Выполнение:**

# Код программы

**import** os  
**import** shutil  
**import** cv2  
**from** ultralytics **import** YOLO  
**import** yaml  
**import** random  
**import** torch  
**import** torchvision  
  
DOWNLOADED\_DATASET\_PATH = **'./rockpaperscissors\_raw'**MODEL\_TYPE = **'yolo12m.pt'**EPOCHS = 2  
IMGSZ = 416  
BATCH = 16  
OPTIMIZER = **'SGD'**MODEL\_NAME = **'yolo12m'**YOLO\_DATASET = **'./rockpaperscissors'**TEST\_IMAGES = [**"20220216\_222607\_jpg.rf.1483f40dac6a642b5bad42a24beeb52a.jpg"**, **"egohands-public-1620849869759\_png\_jpg.rf.c537eb75df29be09ebea80d01e0b4c76.jpg"**] *# Добавьте свои тестовые изображения***def** check\_gpu\_availability():  
 *"""Проверка доступности GPU и вывод информации"""* print(**"="** \* 50)  
 print(**"ПРОВЕРКА ДОСТУПНОСТИ GPU"**)  
 print(**"="** \* 50)  
  
 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"**)  
  
 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..."**)  
  
 **if not** os.path.exists(DOWNLOADED\_DATASET\_PATH):  
 **raise** FileNotFoundError(  
 **f"Папка с датасетом не найдена: {**DOWNLOADED\_DATASET\_PATH**}. Убедись, что ты распаковал архив."**)  
  
 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)  
  
 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**)  
  
 **def** copy\_split(split\_name, yolo\_img\_dir, yolo\_lbl\_dir):  
 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):  
 **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)  
  
 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 найден"**)  
  
 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)  
  
 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):  
 print(**f"\nОбучение модели YOLOv12m для {**num\_classes**} классов: {**class\_names**}..."**)  
 print(**f"Обучение на устройстве: {**device**}"**)  
  
 last\_weights = **f"D:/University/4kurs/OIIS/lab3/my/runs/detect/{**MODEL\_NAME**}/weights/last.pt"** best\_weights = **f"D:/University/4kurs/OIIS/lab3/my/runs/detect/{**MODEL\_NAME**}/weights/best.pt"** device\_param = 0 **if** str(device).startswith(**'cuda'**) **else 'cpu'  
  
 if** os.path.exists(last\_weights):  
 print(**f"Найдены предыдущие веса: {**last\_weights**}, продолжаем обучение с последней эпохи"**)  
 model = YOLO(MODEL\_TYPE)  
 results = model.train(  
 data=data\_yaml\_path,  
 epochs=EPOCHS,  
 imgsz=IMGSZ,  
 batch=BATCH,  
 name=MODEL\_NAME,  
 optimizer=OPTIMIZER,  
 patience=10,  
 lr0=0.001,  
 device=device\_param,  
 workers=4,  
 amp=**True**,  
 save\_period=1,  
 resume=**True** )  
 **elif** os.path.exists(best\_weights):  
 print(**f"Найдены лучшие веса: {**best\_weights**}, дообучаем с них"**)  
 model = YOLO(best\_weights)  
 results = model.train(  
 data=data\_yaml\_path,  
 epochs=EPOCHS,  
 imgsz=IMGSZ,  
 batch=BATCH,  
 name=MODEL\_NAME,  
 optimizer=OPTIMIZER,  
 patience=10,  
 lr0=0.001,  
 device=device\_param,  
 workers=4,  
 amp=**True**,  
 save\_period=1  
 )  
 **else**:  
 print(**f"Начинаем обучение с нуля: {**MODEL\_TYPE**}"**)  
 model = YOLO(MODEL\_TYPE)  
 results = model.train(  
 data=data\_yaml\_path,  
 epochs=EPOCHS,  
 imgsz=IMGSZ,  
 batch=BATCH,  
 name=MODEL\_NAME,  
 optimizer=OPTIMIZER,  
 patience=10,  
 lr0=0.001,  
 device=device\_param,  
 workers=4,  
 amp=**True**,  
 save\_period=1  
 )  
  
 print(**"Обучение завершено!"**)  
 **return** model  
  
  
  
**def** evaluate\_model(model, data\_yaml\_path, device):  
 *"""Оценка модели на тестовых данных"""* print(**"\nОценка модели..."**)  
  
 **if** str(device).startswith(**'cuda'**):  
 device\_param = 0  
 **else**:  
 device\_param = **'cpu'** metrics = model.val(  
 data=data\_yaml\_path,  
 split=**'test'**,  
 device=device\_param  
 )  
  
 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)  
  
 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..."**)  
  
 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:  
 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**}"**)  
  
 **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)  
  
 device = check\_gpu\_availability()  
  
 num\_classes, class\_names, yolo\_data\_yaml\_path = setup\_dataset\_from\_download()  
  
 print(**"\n"** + **"="** \* 50)  
 print(**"ЭТАП 2: ОБУЧЕНИЕ МОДЕЛИ"**)  
 print(**"="** \* 50)  
  
 monitor\_gpu\_usage()  
  
 model = train\_yolo\_model(num\_classes, class\_names, yolo\_data\_yaml\_path, device)  
  
 monitor\_gpu\_usage()  
  
 print(**"\n"** + **"="** \* 50)  
 print(**"ЭТАП 3: ОЦЕНКА МОДЕЛИ"**)  
 print(**"="** \* 50)  
 metrics = evaluate\_model(model, yolo\_data\_yaml\_path, device)  
  
 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()

Спецификация ввода

Python YOLORockPaperScissors <batch\_size>x3x<IMGSZ>x<IMGSZ>

Доп. обязательные параметры (заданы в коде переменными):

* DOWNLOADED\_DATASET\_PATH — путь к скачанному и распакованному датасету в формате YOLO (data.yaml присутствует).
* YOLO\_DATASET — директория, куда скрипт скопирует структуру train/valid/test (images + labels).
* MODEL\_TYPE, EPOCHS, BATCH, IMGSZ, OPTIMIZER, MODEL\_NAME — параметры обучения задаются в коде.

# Пример

Python YOLORockPaperScissors 16x3x416x416

(в коде по умолчанию BATCH=16, IMGSZ=416)

# Спецификация вывода

1. Файлы весов:

* runs/detect/<MODEL\_NAME>/weights/last.pt — последние веса (если сохранены).
* runs/detect/<MODEL\_NAME>/weights/best.pt — лучшие веса (если сохранены).

1. Метрики в консоли / объект metrics:

* mAP50-95 = <float> (например 0.1234)
* mAP50 = <float>

1. Логи обучения: информация об устройстве (CPU/GPU), использование памяти, прогресс эпох и пр.

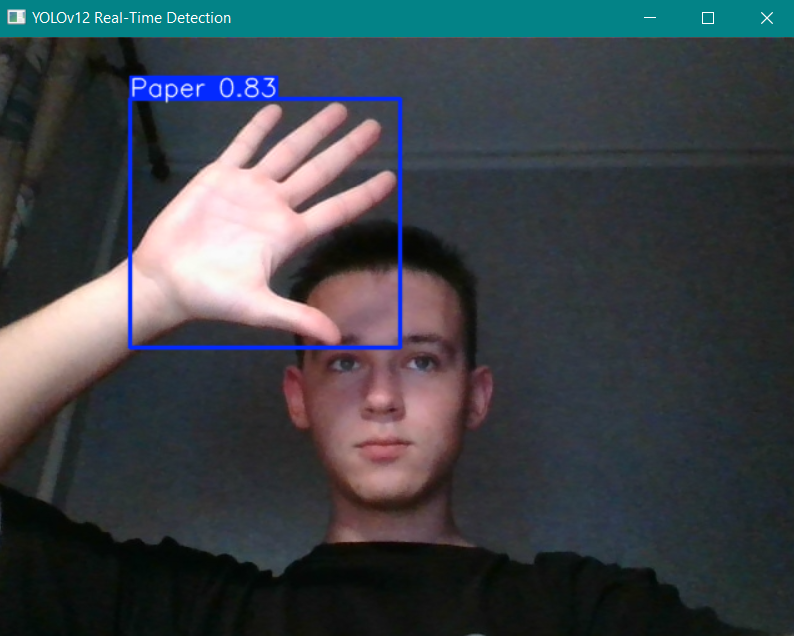
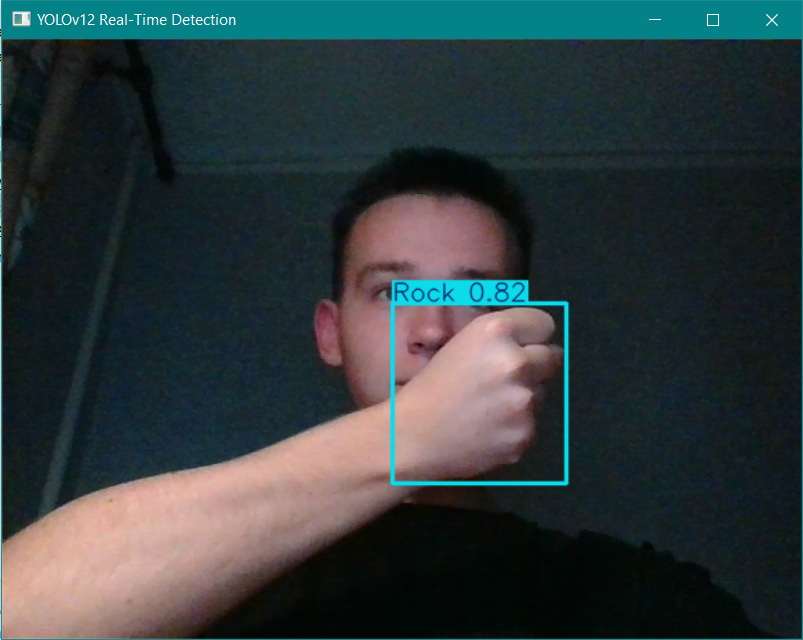
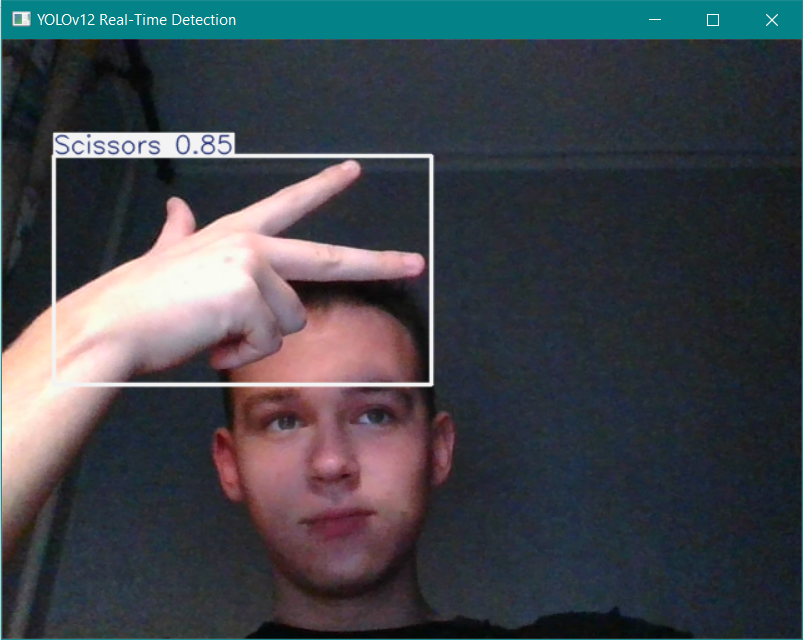
**Пример**

mAP50-95: 0.4321

mAP50: 0.6723

Модель сохранена в: runs/detect/yolo12m/

# Рисунки с результатами работы программы

****  

**Вывод:** осуществил обучение нейросетевого детектора для решения задачи обнаружения заданных объектов.