Постановка задачи: обучить instance segmentation нейронную сеть для сегментации баркода в заранее найденном bbox’е.

Ожидаемый результат: нейронная сеть, сегментирующая баркоды внутри ранее найденного bbox’а с метрикой качества iou не менее 0.9. Входные данные для модели – RGB изображения, выходные данные – разметка для каждого изображения в формате {class_idx x_0 y_0 x_1 y_1 x_2 y_2 ...}, где class_idx – метка класса найденного объекта, x_i y_i – точки, из которых состоит граница объекта (т.е. полигон) из диапазона [0, 1], т.е. нормированные относительно размеров входного изображения.

Шаги решения задачи:

1) Были исследованы несколько SOTA решений. Исследованы модели OpenCV Barcode Detector, ZBAR Barcode Detector, Yolov8 segmentation. OpenCV и ZBAR показали плохое качество (не распознавали баркоды на простейшем датасете, OpenCV смог распознать 13 из 49 баркодов, ZBAR 17 из 49, yolov8 распознала 45 из 49 баркодов), поэтому было решено остановиться на модели Yolo.
2) Были исследованы разные архитектуры Yolov8 (nano(n), small(s), medium(m), large(l), extra_large(x)). Было решено остановиться на архитектуре yolov8x, поскольку имеются достаточные аппаратные средства для обучения большой модели, а качество результатов она показала лучше, чем остальные.
3) Возникла необходимость написания скрипта для приведения формата нашей разметке к разметке, необходимой yolo segmentation. Был написан скрипт json_to_yolo_txt.ipynb. (лежит в личном репозитории)
4) Поскольку нейронную сеть необходимо обучать для сегментации баркодов в bbox’е (потому что inference предполагается именно в таком формате), возникла необходимость написать скрипт для выделения bbox’ов с баркодами из размеченный изображений моего датасета. Задача также осложнялась тем, что необходимо было корректно автоматически разметить баркоды на получившихся изображениях. Данная подзадача была выполнена, скрипт лежит в моем личном репозитории (bbox_eraser.ipynb).
5) Было исследовано влияние входной размерности изображения на качество модели. Испытывались размерности: 128x128, 256x256, 512x512, 768x768. Валидация показала, что 128x128 – слишком маленькая размерность и модель, обученная на такие входные данные, показала плохое качество на инференсе (распознала только ~30 из 49 баркодов). Модель, обученная для входных картинок 256x256 показала отличное качество (49 из 49), при этом средняя iou составила ~0.82. Модели, обученные на размерностях 512x512 и 768x768 не показали качественно лучшего результата, однако обучение их заняло в ~8 раз больше времени. В итоге была выбрана архитектура yolo8x segmentation с входной размерностью картинки 256x256.

Результат: обучена модель yolov8x segmentation для сегментации баркодов внутри bbox’а. Качество по метрике iou составило ~0.88. Заявленного в ожидаемом результате качества добиться не удалось, поскольку использовались только данные, собранные мной лично. Были доступны несколько готовых датасетов, однако они не подходили для текущей архитектуры, поскольку имели разметку для детекции, а так же были сняты на неизвестные камеры (следовательно там была неподходящая для моей архитектуры разметка, а также домены изображений в этих датасетах отличались от того, в котором снимает моя камера). В качестве следующих шагов на мой взгляд необходимо было бы собрать все размеченные одногруппниками данные (на данный момент я не стал этого

делать, поскольку данных у всех мало, у каждого разные домены и слияние таких данных в один датасет могло только ухудшить качество). В репозиторий выложить обученную модель нет возможности, поскольку она превышает лимит размета файла в github, поэтому если она вам понадобится напишите мне (например в telegram) и я вам ее скину. В личном репозитории лежит датасет с баркодами в bbox'ах (bboxed_barcodes_dataset.zip), а также jupyter notebook (yolo_barcodes.ipynb), в котором можно повторить все шаги по получению iou метрики на валидационных данных. Для запуска модели на инференс можно воспользоваться двумя инструментами: 1) прямо из командной строки необходимо запустить команду yolo segment predict data=<путь до файла data.yaml в датасете> model=<путь до модели.pt>. Можно указать источником source='0' (проверено для windows), а также поставить флаг show=True чтобы видеть результат сегментации на экране. 2) Можно запустить через python, для этого необходимо создать модель (как это сделано в yolo_barcodes.ipynb) и запустить ее model.predict(source=<ваш источник>). В результате создастся папка runs/segment/predict в которой будут лежать изображения с наложенными масками (или видео, если вы указали его как источник) и папка labels, в которой будут лежать файлы с разметкой полигонами найденных баркодов, названия которых соответствуют названию картинок, на которых происходил инференс (например если делали предсказание для картинки my_awesome_barcode.jpg появится файл my_awesome_barcode.txt). Таким образом, считаю личную задачу решенной.

В семестре также возникали общие задачи, привожу список общих задач, решенных лично мной:

1) "Нулевой" задачей к каждой новой паре была разметка новых изображений. Мной размечено 50 изображений (меньше чем требовалось суммарно, однако на многих из них было сразу много баркодов, а также в процессе семестра приходилось переразмечать их все для удовлетворения критериев разметки).
2) Возникла задача поиска готовых датасетов и их описания. В общем репозитории лежит написанное мной исчерпывающее описание 15 датасетов: их структура, описание данных в них, включающее описание картинок, описание формата разметки. (https://github.com/dvpsun/mipt2024s-5-modern-cv/blob/main/Snatched%20datasets%20description.md)
3) Возникла необходимость написание стандарта разметки. В общем репозитории лежит папка labeling (https://github.com/dvpsun/mipt2024s-5-modern-cv/tree/main/labeling) с описание того, как осуществляется разметка в VGG Image Annotator с проиллюстрированными примерами для каждого шага, шаблон проекта разметки, скрипт для разделения разметки проекта из нескольких картинок на несколько отдельных проектов разметки, а также сам разметчик (via.html).
4) Возникла задача описания существующих видов баркодов. Данная задача выполнялась не только мной, но и некоторыми из одногруппников, однако многие ресурсы для описаний были предоставленны именно мной (сами описания я не делал). (https://github.com/dvpsun/mipt2024s-5-modern-cv/blob/main/barcode_types.md)