Голиков Михаил БВТ2402

Проект

Была взята следующая стратегия:

1. Предобработать картинку через изменение перспективы изображения с помощью OpenCV (изображение 1), изменить контрастность через white patch.
2. Использовать OCR (PyTesseract или PaddleOCR)+ OpenCV для поиска символов и линий подчеркивания (координаты bounding box + labeling).
3. С помощью трансформеров (как доминирующая модель для fine tuning рассматривалась LayoutLMv2 / LayoutLMv3, так как она была специально обучена для разделения входных данных на вопросы, ответы и другие данные, которые нельзя отнести к первым группам). Как альтернатива рассматривалась модель Donut, но так как для её использования не нужна OCR, то модель становится очень тяжелой и долгой для обучения.
4. Сохранение значений в формате JSON.

Как альтернативные стратегии рассматривались:

* Замена LayoutLMv2 простой моделью для обработки текста с запросом на разделение.
* Использование регулярных выражений для поиска специальных полей (например, дата или подпись)

Как основной датасет можно использовать FUNSD, который надо бы доработать. Изначально там хранятся картинки документов, слова для описания заполненных полей и ответы этих же полей в формате JSON. При помощи OpenCV. Заменить ответы на белые вставки и выделять линии для заполнения.

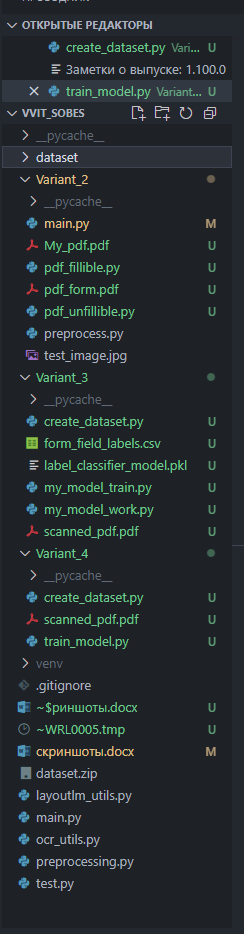
В качестве основных метрик были выбраны Recall как основная и F1 score для общего изучения.

Нам важнее всего находить все необходимые поля, в то время как поиск только правильных полей был второстепенной задачей (эти поля можно будет пропустить).

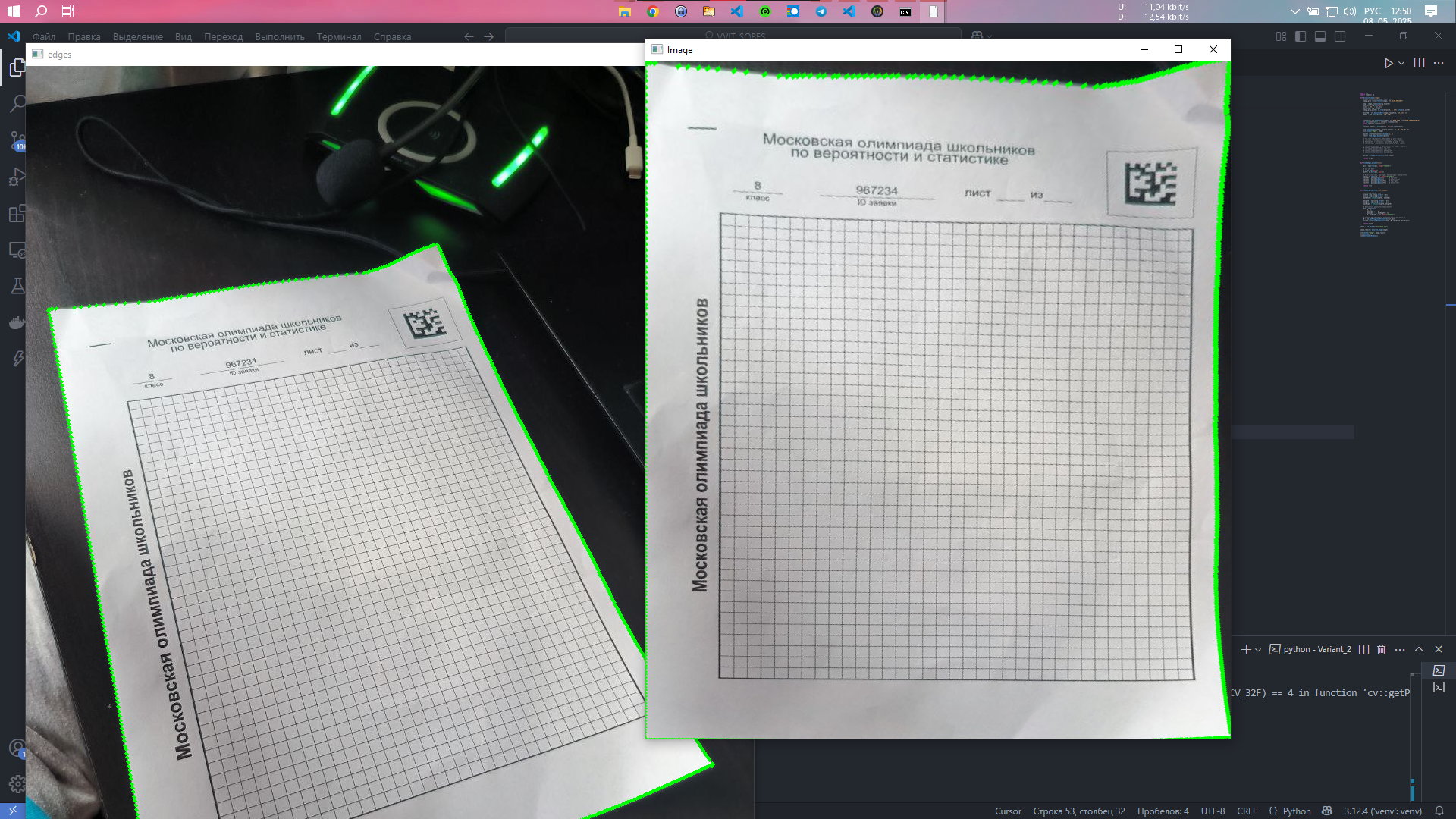
Далее задание было уточнено, из-за чего появилась следующая идея.

Обработка pdf файлов для определения полей для заполнения с помощью поиска определённых слов из словаря или слов с окончанием на “:”. При нахождении идёт поиск линий, на которые нужно писать ответ, выделяется bounding box с высотой текста. Далее идёт сканирование через OCR и сохранение данных в формате JSON для обучения модели. Пример можно наблюдать на изображении 2.

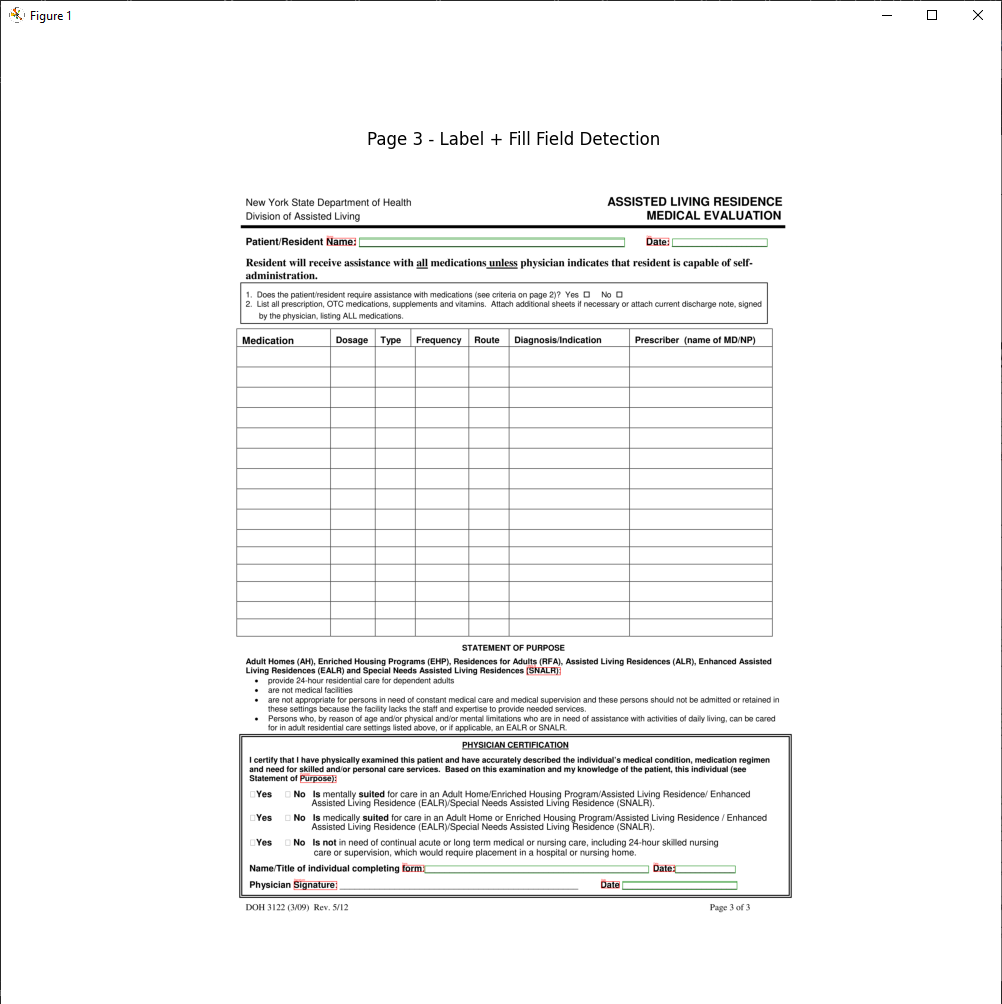
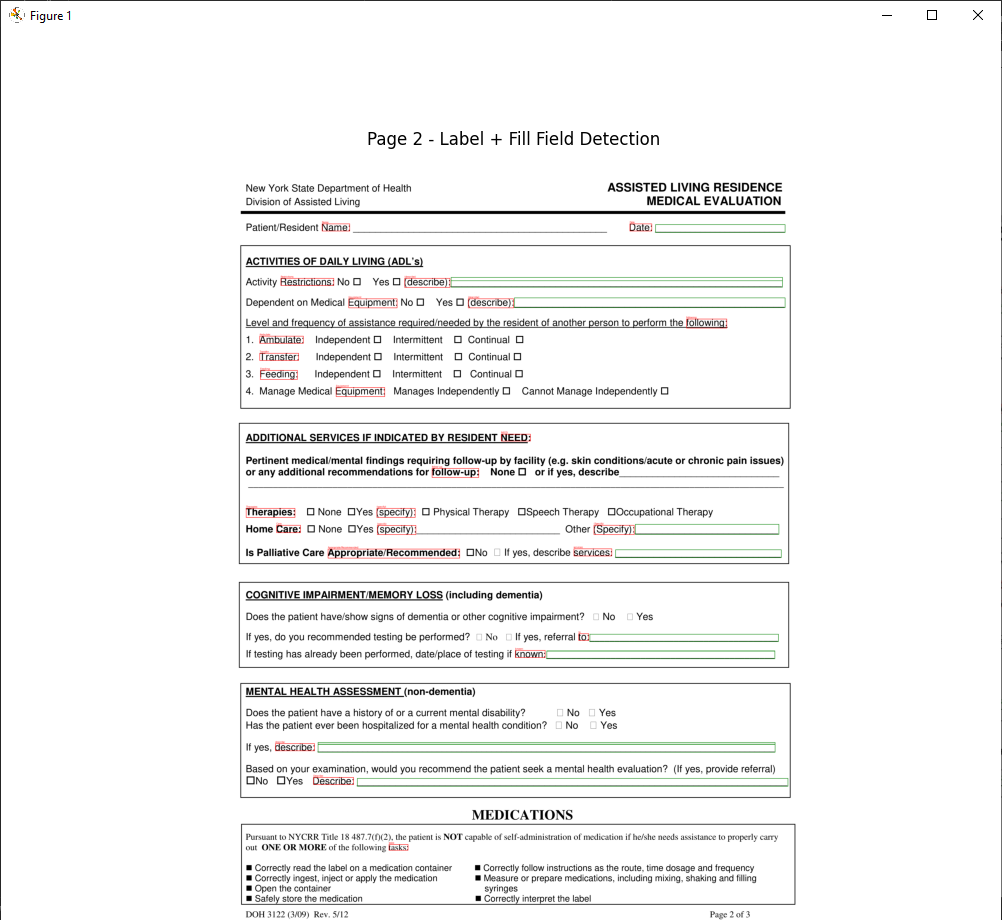
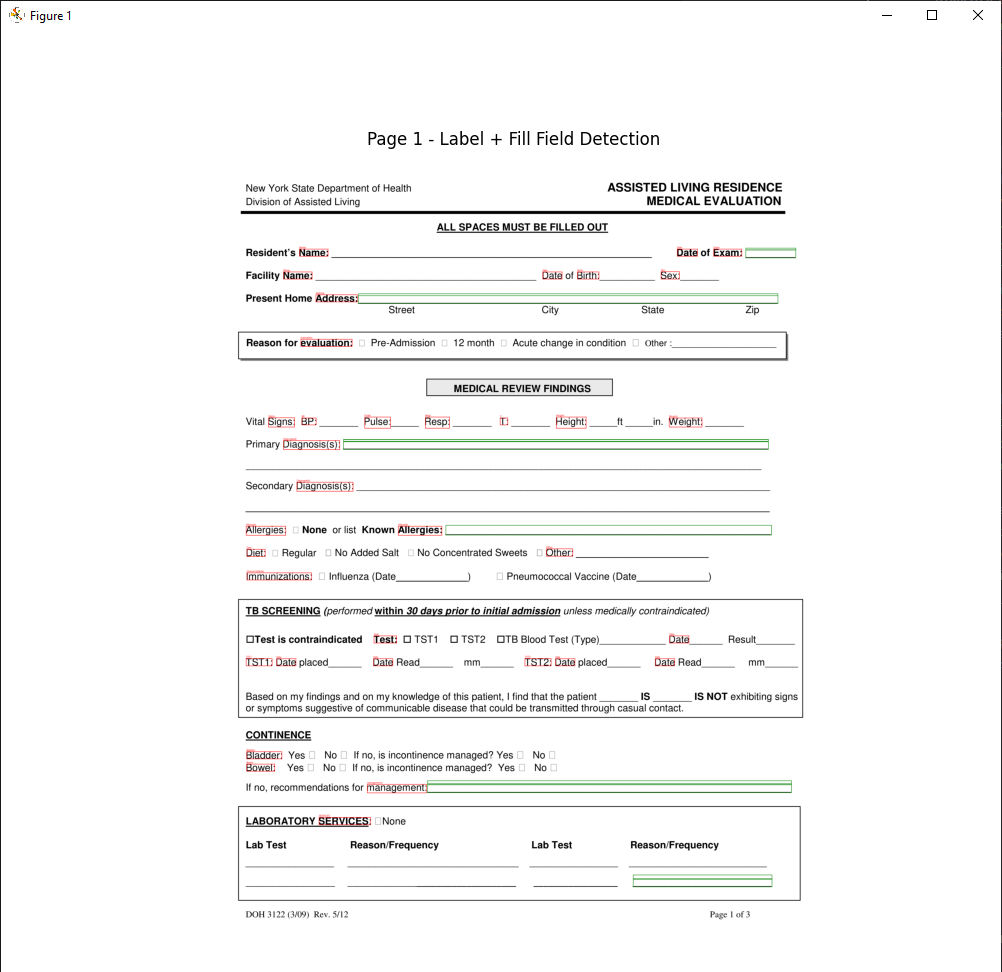
Во время всего проекта было много трудностей. Первая – при использовании фотографий PyTesseract распознавал не отдельные слова, а пары символов, из-за чего было принято решение перейти на PaddleOCR. Однако потом при обработке документов из pdf стало возможным вернуться к более легкой PyTesseract для создания JSON файлов с bounding box. Вторая ― несовместимость моей CUDA (12.4) с большинством моделей (CUDA 11.8). Третья ― долгое развёртывание моделей и малое количество подходящих датасетов. Всё это привело к тому, что запустить обучение модели так и не удалось. Однако была обучена модель RandomForest для поиска слов-вопросов для заполнения на 1 документе из трех страниц, размеченном с помощью словаря (полностью бесполезная модель, но при большой ручной разметке может заменить LayoutLMv2 для экономии вычислительных ресурсов)



Элемент 1 ― все файлы в проекте, разные варианты делают разные задачи



Изображение 1 ― Изменение перспективы документа через поиск наибольшего светлого контура



Изображение 2 ― Пример разметки подготовленных данных

import cv2

import numpy as np

def preprocess\_image(image):

    def create\_white\_patch\_image(image):

        image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

        img = image\_gray.astype(np.float32)

        max\_vals = img.max(axis=0)

        scale = 255.0 / max\_vals

        balanced = img \* scale

        image\_gray\_white = np.clip(balanced, 0, 255).astype(np.uint8)

        return image\_gray\_white

    def find\_edges\_document(pts):

        pts = np.array(pts, dtype="float32")

        s = pts.sum(axis=1)

        diff = np.diff(pts, axis=1)

        rect = np.zeros((4, 2), dtype="float32")

        rect[0] = pts[np.argmin(s)]       # top-left

        rect[2] = pts[np.argmax(s)]       # bottom-right

        rect[1] = pts[np.argmin(diff)]    # top-right

        rect[3] = pts[np.argmax(diff)]    # bottom-left

        return rect

    def change\_perspective(rect, image):

        (tl, tr, br, bl) = rect

        widthA = np.linalg.norm(br - bl)

        widthB = np.linalg.norm(tr - tl)

        maxWidth = int(max(widthA, widthB))

        heightA = np.linalg.norm(tr - br)

        heightB = np.linalg.norm(tl - bl)

        maxHeight = int(max(heightA, heightB))

        dst = np.array([

            [0, 0],

            [maxWidth - 1, 0],

            [maxWidth - 1, maxHeight - 1],

            [0, maxHeight - 1]], dtype="float32")

        # Compute the perspective transform matrix and apply it

        M = cv2.getPerspectiveTransform(rect, dst)

        warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))

        return warped

    image\_gray\_white = create\_white\_patch\_image(image)

    blurred = cv2.GaussianBlur(image\_gray\_white, (15, 15), 1)

    edges = cv2.Canny(blurred, 100, 200)

    contours = cv2.findContours(edges, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

    if len(contours) == 2: contours = contours[0];

    else: contours = contours[1];

    largest\_contour = max(contours, key=cv2.contourArea)

    #cv2.drawContours(image, largest\_contour, -1, (0, 255, 0), 3)

    #cv2.imshow("edges", image)

    points = largest\_contour.reshape(-1, 2)

    rect = find\_edges\_document(points)

    warped\_image = change\_perspective(rect, image)

    image\_result = create\_white\_patch\_image(warped\_image)

    # cv2.imshow("White\_patch", image\_result)

    #image\_result = cv2.resize(image\_result, (1000, 1000))

    return image\_result

Элемент 2 ― Программа для изменения перспективы изображения