Обоснование выбора алгоритмов поворота изображения и оценка эффективности решения

Краткое описание решения

В данном проекте задача нахождения угла поворота изображения документа решается путем выделения границ горизонтальных объектов, чаще всего строк, так они чаще всего встречаются в документах и имеют строго горизонтальное направление.

Далее происходит поиск самой длинной части прямоугольного контура каждой строки и вычисление медианного угла наклона всех длин. После чего мы фильтруем контуры с экстремальными величинами углов. Затем на основании медианного угла, происходит разворот самого документа.

Далее, мы сравниваем найденный угол с углом записанным в папке «angles», и высчитываем точность алгоритма на 40 документах, угол считается верно определенным, если его величина отличается менее чем на один градус, от реального угла. Проверка записана в декораторе, который находится в отдельном модуле «utils.py». Разворот изображений для проверки реализован в файле «rotate_image.py».

Выбранный подход основывается на функционале библиотеки OpenCV, что обеспечивает высокую скорость, точность и надежность при обработке изображений. Реализована параллельная обработка файлов для ускорения работы.

Алгоритм корректно работает на файлах повернутых не более чем на 44 градуса в каждую сторону.

Ниже описаны ключевые этапы алгоритма, обоснование их выбора и оценка эффективности с точки зрения использования памяти и ресурсов.

Описание алгоритмов

- Исходное цветное изображение преобразуется в градации серого с помощью **cv2.cvtColor**. Затем применяется пороговая бинаризация с использованием метода Оцу (cv2.threshold с параметром THRESH_BINARY + THRESH_OTSU), что позволяет автоматически определить оптимальный порог для разделения фона и объекта (документа).
- Применяется дилатация (**cv2.dilate**) с использованием заданного структурного элемента (ядра), что позволяет объединить разрозненные участки и улучшить целостность границ.
- Метод **Canny** (**cv2.Canny**) выделяет резкие изменения яркости, определяя границы объектов на изображении.
- С помощью **cv2.findContours** находятся контуры, а **cv2.minAreaRect** вычисляет минимальный ограничивающий прямоугольник, который может быть повернут относительно осей. Для каждого контура определяется его ориентация (угол наклона).
- На основе медианного угла определяется корректирующий угол. При необходимости производится дополнительная корректировка (например, если угол меньше -45°, добавляется 90° для корректного ориентира). Поворот изображения осуществляется с помощью **cv2.getRotationMatrix2D** и **cv2.warpAffine**, что позволяет сохранить исходное разрешение изображения.

Скорость выполнения

Все используемые алгоритмы (преобразование в оттенки серого, бинаризация, дилатация, Canny, поиск контуров, вычисление минимальных прямоугольников, поворот) являются встроенными в OpenCV и оптимизированы на уровне C/C++. Это обеспечивает высокую скорость обработки даже для изображений высокого разрешения.

Потребление памяти

Снижение объема данных:

Преобразование изображения в оттенки серого и последующая бинаризация значительно уменьшают объем обрабатываемых данных, так как избавляют от цветовой компоненты и упрощают структуру изображения.

Оптимизация промежуточных вычислений:

Промежуточные результаты (например, бинаризованное изображение, дилатированное изображение, результаты детекции границ) хранятся в отдельных переменных, что позволяет избежать избыточного копирования данных. При этом использование библиотек OpenCV и NumPy гарантирует эффективное управление памятью.

Использование вычислительных ресурсов

Процессорное время:

Благодаря оптимизированной реализации алгоритмов в OpenCV и параллельной обработке изображений с использованием ThreadPoolExecutor для обработки нескольких файлов одновременно, алгоритм эффективно распределяет вычислительную нагрузку. Это позволяет сократить общее время обработки при наличии большого количества изображений.

Баланс между качеством и производительностью:

Выбранное решение обеспечивает высокий уровень точности (корректное определение угла поворота) при умеренном потреблении ресурсов. Это делает алгоритм подходящим как для локальных приложений, так и для серверных систем, где важна масштабируемость и эффективность использования ресурсов.