Sprawozdanie z laboratorium: Komunikacja człowiek - komputer

Sprawozdanie I: Przetwarzanie obrazu – aplikacja

14 listopada 2017

Prowadzący: mgr inż. Paweł Liskowski

Autorzy: Mateusz Urbaniak 127345
Kajetan Zimniak 127229

Zajęcia wtorkowe, 8:00.

1 Wstęp

Celem projektu było znalezienie i zaimplementowanie stosunkowo szybkiej metody wykrywania i rozpoznawania cyfr drukowanych ze zdjęcia za pomocą algorytmu „template matchingu”. Wśród wielu możliwych dróg stworzenia programu, na bieżąco dodawaliśmy elementy poprawiające sprawność jego działania. Podział prac wynikał z kolejnych etapów projektu: implementacja algorytmu szukającego krawędzi obiektów, segmentacja obrazu, dopasowanie do wzorca, testowanie i modyfikacja programu. W testach wykorzystywaliśmy zdjęcia wykonywane pod różnym kątem i nachyleniem fotografowanej kartki, jednak nasz algorytm byłby w rzeczywistości wykorzystywany w fizycznym skanerze OCR, gdzie kartka leżałaby na płaskiej płaszczyźnie i w kontrolowanym oświetleniu skanera (tym samym, jak wynika z późniejszych eksperymentów i wskazanych powodów, przy takich założeniach algorytm działałby w 100% skutecznie, a działanie programu można by jeszcze bardziej przyspieszyć poprzez brak usuwania zanieczyszczeń obrazu).

2 Opis algorytmu i wykorzystanych filtrów, funkcji i metod

W pierwszej kolejności za pomocą funkcji *io.ImageCollection()* ładujemy obrazy do programu, które następnie, w kolejności,

poddawane są przekształceniu na odcienie szarości (*color.rgb2grey()*), w celu uzyskania dwuwymiarowej tablicy, na której dalej pracujemy. Fukcje *morphology.closing()* i *morphology.opening()* służą nam do usunięcia podstawowych zanieczyszczeń (noise) obrazu wejściowego. *Np.percentile()* i *exposure.rescale_intensity()* służą do zmiany intensywności. W tym momencie jesteśmy gotowi do użycia *measure.find_contours()* w celu znalezienia konturów, a dokładniej ich współrzędnych. Nie wystarczy to jednak do zakończenia pierwszej części programu. Na przykład dla cyfry „6”, funkcja wykryje nam dwa kontury, ponieważ znak składa się z dwóch kształtów. Przy użyciu minimalnych i maksymalnych współrzędnych x, y, usuwamy wewnętrzne części znaków, które nie pomogą nam w dalszej detekcji. W tym momencie jesteśmy gotowi do segmentacji, polegającej na wycięciu odpowiednich części macierzy obrazu. Jest to operacja nieskomplikowana, jednak potrzebna do użycia funkcji *match_template().* Zwraca ona liczbę przedstawiającą dokładność pokrycia, znalezienia szukanego elementu na obrazie. W naszym przypadku konieczne było początkowe wycięcie każdego znaku, aby zachować kolejność odczytywanych cyfr, ponieważ każdy znak sprawdzany jest w pętli, z każdym przygotowanym szablonem cyfr od 0 do 9.

W trakcie eksperymentów zdecydowaliśmy się dodać kolejną funkcjonalność. Za pomocą funkcji *draw.polygon_perimeter()* i *cv2.putText()* zaznaczana jest wykryta cyfra i wypisywana odczytana

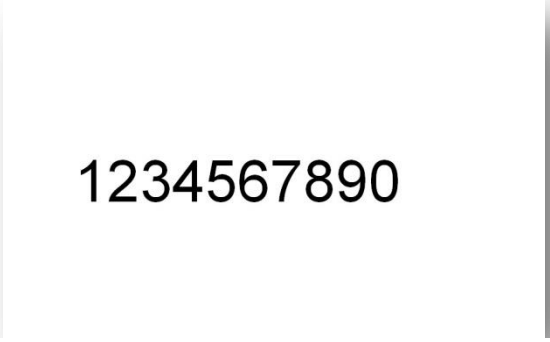
przez program wartość. Całość zapisywana jest w pdfie. Sprawia to, iż wynik programu przedstawiany jest w bardzo prosty i przejrzysty sposób w jednym miejscu, mimo wielu obrazków wejściowych.

3 Eksperymenty

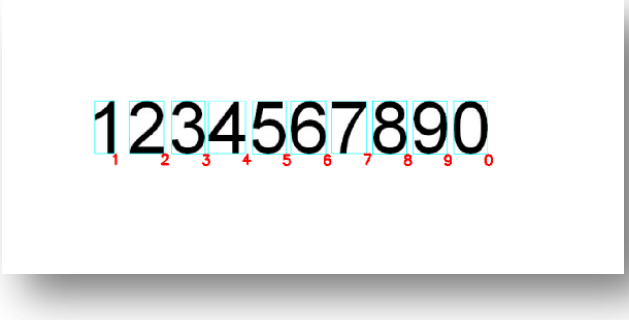
Program przystosowany jest do rozpoznania każdego rodzaju czcionki. W eksperymentach skupiliśmy się na czcionce Arial, po to, by dokładniej zbadać wpływ rozmiaru, koloru i tła zdjęcia.

Na początku pracowaliśmy nad tym, aby program ze stu-procentową skutecznością rozpoznawał cyfry ze zrzutu ekranu *(rys.1 i 2)*, na białym tle. Problemem okazały się różnice w rozmiarze zdjęcia i szablonu. Łatwo je jednak wyeliminowaliśmy skalując obrazy.

Rysunek 1:



Rysunek 2:



Po osiągnięciu sukcesu z powyżej przytoczoną sytuacją, zajęliśmy się badaniem przykładów nieco bardziej złożonych, mianowicie

mieszanki tekstu i cyfr (na przykład rozpoznawanie peselu wśród innych danych *Rys. 3, 4*)

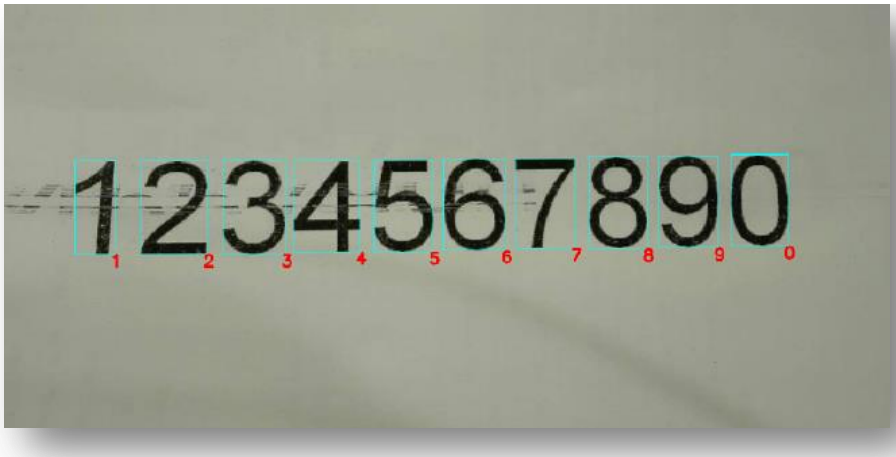
Rysunek 3, obraz wejściowy:

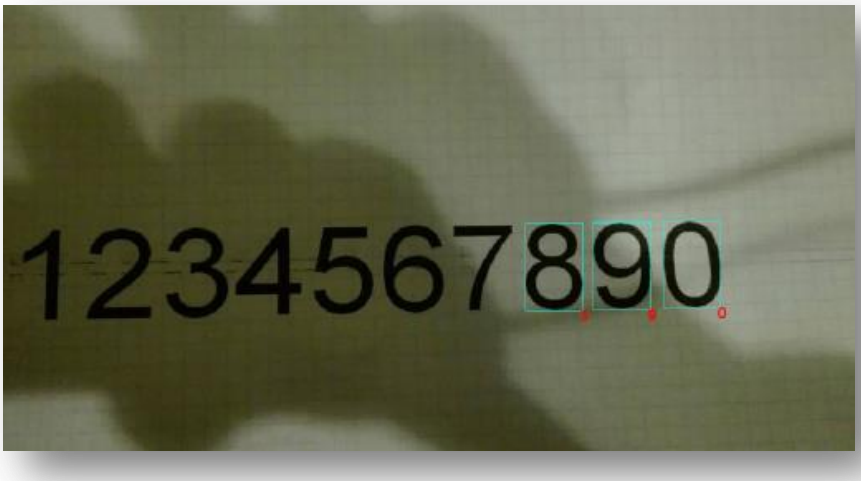
Rysunek 4 obraz wyjściowy:

Następnym etapem naszych testów było sprawdzanie skuteczności na zdjęciach zrobionych telefonem. Celowo używaliśmy papieru z niejednolitym tłem, oraz oświetlenia zakłócającego czytelność. Poniżej kilka przykładów. Można zaobserwować, że w większości sytuacji program działał bez zarzutu. Problemy pojawiały się tylko w miejscach bardzo zacienionych, co wynikało z braku możliwośći wykrycia zmian gradientu obrazu a przez to detekcji krawędzi a w konsekwencji braku możliwości segmentacji obrazu wejściowego. Brak wykrycia części wydrukowanych cyfr wynika również z obrotu i nachylenia fotografowanej kartki. Aby rozwiązać ten problem, musielibyśmy obracać nasze template'y o odpowiednie wartości, jednak w praktyce wiązałoby się to z rezygnacją z założonej metody

(gdyż wówczas większość algorytmów rozpoznawania działałby znacznie lepiej, podczas gdy template matching zakłada wykorzystanie prostych szablonów i ich porównań).

Rysunek 5, 6, 7:





4 Podsumowanie

Po wielu eksperymentach i wyborze odpowiednich metod, udało nam się zrealizować założony cel. Stworzyliśmy program wykrywający i rozpoznawający cyfry, nie tylko na zrzucie ekranu, ale również te zrobione aparatem w zwykłym telefonie. W domyśle, fizyczny skaner OCR wykonywałby skan kilkudziesięciu zadanych stron, a następnie dla wszystkich zeskanowanych stron uruchamiałby stworzony przez nas algorytm i dokonywał detekcji i rozpoznania znaków cyfrowych (optycznych, drukowanych czcionek) na zadanych kartkach papieru. Umożliwiałoby to konwersję obrazu zawierającego znaki cyfrowe do wynikowego pliku tekstowego w formacie wygodnym do modyfikacji. Jest to program, który już na tym poziomie zaawansowania może przydać się w wielu miejscach i z łatwością można przerobić go wg potrzeb określonego celu. W ramach laboratorium wykonaliśmy szablony tylko jednej czcionki, jednak mając na uwadze potrzeby praktyczne, program dostosowaliśmy do łatwego dodania każdej innej, nie zajmie to więcej niż kilkadziesiąt sekund na każdą z rodzaju czcionki.

Ważnym wnioskiem jest konieczność dobrania metody do konkretnego problemu. W przypadku braku konieczności wykorzystania znacznie bardziej skomplikowanych metod pokroju konwolucyjnych sieci neuronowych czy bardziej kosztownych obliczeniowo algorytmów rozpoznawania, nie ma sensu atakować

problemu czymś bardziej skomplikowanym, tym bardziej, że zależy nam na szybkości działania, a algorytmy odporne na obrót mogłyby wykrywać obrócone litery jako kompletnie inne znaki np. ‘A’ jako ‘V’ czy ‘9’ i ‘6’. Zakładamy inteligentne wykorzystanie programu przez użytkownika, który dąży do zeskanowania kartki w jak największym pionie.

Źródło

[1] http://scikit-image.org

[2] http://matplotlib.org

[3] docs.opencv.org