
POBR

Wykrywanie loga sklepu H&M

KORNEL SKÓRKA

JANUARY 21, 2022

Wstęp

Celem projektu jest stworzenie aplikacji, która będzie w stanie rozpoznać logo firmy H&M. Logo to przedstawione zostało na poniższym obrazku.



Logo składa się z koloru czerwonego, na podstawie tej informacji postanowiłem że najlepszą metodą będzie zaimplementowanie algorytmu progowania.

Obrazy Źródłowe

Zdjęcia użyte w projekcie zostały wykonane telefonem IPhone XS.



Figure 1: Zdjęcie 1



Figure 2: Zdjęcie 2

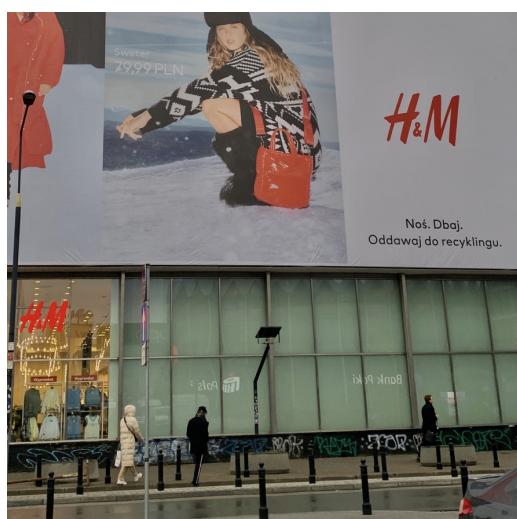


Figure 3: Zdjęcie 3

Wydajność

Pomimo zastosowania języka python, udało mi się uzyskać szybkie działanie programu dzięki zastosowaniu biblioteki Numba. Numba tłumaczy funkcje Pythona na zoptymalizowany kod maszynowy, wykorzystując standardową bibliotekę kompilatora LLVM. Skompilowane przez Numbę algorytmy numeryczne w Pythonie mogą zbliżyć się do prędkości C lub FORTRANu. Biblioteka umożliwia też zrównoleglenie operacji.

Algorytm przetwarzania

Dla ułatwienia eksperymentowania, początkowo implementacja wykorzystywała funkcje biblioteki Open CV. Dzięki temu mogłem przetestować wiele różnych kombinacji metod przetwarzania. Program został zaimplementowany w języku Python. Ostateczna wersja korzysta z open CV tylko do wczytania i wyświetlenia obrazu.

Przetwarzanie odbywa się według następującego algorytmu:

1. Wczytanie obrazu do pamięci
2. Filtracja dolnoprzepustowa za pomocą filtra konwolucyjnego
3. Konwersja do przestrzeni barw HSV
4. Progowanie
5. Domykanie (dilate i erode)
6. Wykrywanie obiektów za pomocą algorytmu flood fill
7. Wyliczenie cech obiektów
8. Jeśli obiekty spełniają wcześniej ustalone warunki dla wartości cech, na oryginalny obraz nakładane są prostokąty zawierające obraz
9. Wyświetlany jest oryginalny obraz z prostokątami zawierającymi loga H&M

W kolejnych sekcjach opiszę dokładniej istotne kroki algorytmu wraz z przykładami.

Filtracja

w celu usunięcia szumów z obrazów, zastosowana została filtracja dolnoprzepustowa za pomocą filtra konwolucyjnego i następującej macierzy:

$$\mathbf{F}_1 = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$



Figure 4: lewa strona: zdjęcie przed, prawa: po filtracji



Figure 5: Zdjęcie 2 przed zastosowaniem filtracji i progowania



Figure 6: Po zastosowaniu filtra dolnoprzepustowego i progowania

Metoda ta została wybrana po porównaniu z innymi metodami usuwania szumów, zależało mi na jak najlepszym wyniku segmentacji. Pod uwagę wziąłem również Gaussian Blur oraz Uśrednianie i Filtr Medianowy.



Figure 7: Po zastosowaniu Filtra Medianowego i progowania

Inne filtry nie dawały tak dobrych efektów więc postanowiłem ich nie implementować.

Konwersja do przestrzeni barw HSV

Obraz konwertowany jest z przestrzeni BGR do HSV. Model HSV nawiązuje do sposobu, w jakim widzi ludzki narząd wzroku, gdzie wszystkie barwy postrzegane są jako światło pochodzące z oświetlenia. Zdecydowałem się na implementację nie używającą liczb zmiennoprzecinkowych. Poniżej zakresy jakie przyjmują pixele:

H (Hue) zakres [0, 179]

S (Saturation) zakres [0, 255]

V (Value) zakres [0, 255]

Konwersja ułatwia operację progowania (łatwiej jest znaleźć przedział dla danego koloru).

Progowanie

Do znalezienia wartości koloru loga w przestrzeni barw skorzystałem z narzędzia Color Picker które dało mi wartość wybranego piksela na zdjęciu w postaci RGB.

Następnie skorzystałem z zaimplementowanej przez mnie funkcji `_bgr2hsv_pixel_fast` która zamienia wartość pojedynczego pixela na BGR. Tę operację powtórzyłem dla kilku obrazków. Otrzymałem następujące przedziały:

$(160 \leq H \leq 180) (100 \leq S \leq 255) (100 \leq V \leq 255)$

oraz

$(0 \leq H \leq 10) (100 \leq S \leq 255) (100 \leq V \leq 255)$

Wynikiem progowania jest mapa bitowa informująca czy dany piksel jest w zadanym zakresie.

Domykanie

Następną z zaimplementowanych metod przetwarzania jest algorytm Domykania (Rozszerzanie, a następnie Erozja). Metoda ta dała mieszane wyniki, dla pewnych obrazów poprawiała wynikowe loga, a dla niektórych nie dawała wymiernych efektów. Z tego powodu zdecydowałem się aby była to funkcjonalność opcjonalna, nie jest wykonywana dla każdego obrazka.

W obu przypadkach algorytm wykonywany jest dla macierzy 3 na 3.

Wykrywanie obiektów za pomocą algorytmu Flood Fill

Kolejnym krokiem jest wykrywanie obiektów na obrazku. Za obiekt uznaje zbiór pikseli sąsiadujących ze sobą, warunkiem jest odpowiednia liczebność takiego zbioru. Warunek ten w kodzie opisuje stała `PIXEL_COUNT_MIN`, aktualnie ustawiona na 1000 pikseli. Wartość tego parametru dobrana została w sposób eksperymentalny.

Algorytm flood fill (zalewania) działa wykorzystując strukturę kolejki. Poniżej wyniki algorytmu:







Wyliczenie cech obiektów

W celu odnalezienia obiektów zawierających logo lub jego części zastosowane są współczynniki kształtu i momenty geometryczne. W wyniku testów poszczególnych cech i momentów postanowiłem zastosować Współczynnik Kształtu Malinowskiej oraz momenty M1 i M7.

Klasyfikacja polega na sprawdzeniu, czy wszystkie parametry elementu mieszą się w odpowiednich zakresach. Wartości maksymalne i minimalne dobrano na podstawie eksperymentów. Są one opisane przez stałe zestawione poniżej:

pierwszy zestaw cech:

$$2 > w3 > 1.9$$

$$0.17 > M1 > 0.16$$

$$0.007 > M7 > 0.0068$$

drugi zestaw cech:

$$3.51 > w3 > 1.76$$

$$0.179 > M1 > 0.1670$$

$$0.008 > M7 > 0.00695$$

Obiekty w pierwszej grupie to litery H i M, natomiast w drugiej jest to całe logo.

Kolejnym krokiem jest oznaczenie czy jest to całe logo czy pojedyncza litera. pojedyncze litery występujące obok siebie są łączone.

Wyniki końcowe

Na koniec zwracane są współrzędne maksymalne dla obiektu wykrytego jako logo. Następnie rysowany jest prostokąt zawierający logo. Poniżej efekty końcowe programu:





