PRZETWARZANIE CYFROWE OBRAZÓW - PROJEKT

Rozpoznawanie loga marki Shell

Opracował: Maciej Pęśko
R.A. 2017/2018
Prowadzący: dr inż. Tomasz Trzciński

1. Wstęp

Celem projektu jest opracowanie programu który poprawnie identyfikuje wybranej klasy obrazy. W tym przypadku jest to rozpoznawanie w obrazach loga marki Shell. Do wykonana zadania użyto języka Python oraz środowiska Jupyter Notebook. Wykorzystane biblioteki to Numpy, Pandas, Matplotlib, OpenCV, Pickle oraz tqdm. Pierwszą część stanowiło przygotowanie danych do uczenia, walidacji oraz testowych. Druga część składała się z wyboru i zaimplementowania algorytmu przetwarzającego obraz i wyciągającego z niego cechy niezbędne do uczenia klasyfikatora. Początkowo wybrano algorytm Canny'ego po wcześniejszym wyrównaniu histogramu dla obrazów wejściowych, jednak słabe wyniki klasyfikacji opartej tylko na krawędziach skłoniły autora do zmiany algorytmu i wyboru algorytmu HOG (Histogram of Oriented Gradients). Trzecią część stanowiło wytrenowanie i ocena klasyfikatora, użyto SVM (Support Vector Machine) dostępnego w pakiecie Opencv. Ostatnią częścią było zaimplemntowanie wykrywania różnej wielkości loga na różnych rozmiarach obrazów I ewaluacja wyników na przykładowym, reprezentatywnym zbiorze obrazów. Dokonano tego przy pomocy implementacji zmodyfikowanej wersji algorytmu Multiscale Detection.

2. Wybór danych

Do uczenia postanowiono wykorzystać obrazy o wielkości 64x64 px. Pierwszą klasę stanowiło 7500 obrazów loga firmy Shell wygenerowanych z jednego wzorca (załączony w katalogu data/logo/logo0.jpg). Przy generowaniu różnych obrazów loga wykorzystano skrypt który w losowy sposób obracał logo o kąt od -20 do 20 stopni, zmieniał jego wielkość oraz wykonywał losowej wartości transformację w poziomie i pionie o wartości od -3 do + 3 px. Drugą klasę stanowiły zdjęcia różnych obiektów nieprzedstawiające loga, wykorzystano tu losowe 7500 obrazów ze zbioru Caltech-256 Object Category Dataset dostępnego pod adresem: http://www.vision.caltech.edu/Image_Datasets/Caltech256/. Zdjęcia do klasyfikacji wczytywano jako czarnobiałe oraz poddano także operacji wyrównania histogramu jednak w końcowym rozrachunku nie przynosiła ona żadnych dodatkowych korzyści więc zrezygnowano z niej.

3. Zaimplementowane algorytmy

3.1. Podejście pierwsze - Wyrównanie histogramu + Canny

Ze względu na fakt iż logo ma dość charakterystyczny wzór w pierwszej kolejności postanowiono jako deskryptora cech użyć algorytmu Canny'ego oraz uzyskanych z niego obrazów krawędzi. Wszystko poprzedzone zostało operacją wyrównania histogramu.

3.1.1. Wyrównanie histogramu

Wyrównywanie histogramu polega na takiej zmianie wartości punktów obrazu aby w równych przedziałach histogramu ilość punktów obrazu była w przybliżeniu taka sama. Operacja ta pozwala na uwypuklenie mało kontrastowych szczegółów w obrazie i ogólną poprawę kontrastu.

3.1.2. Algorytm Cannego

Algorytm Canny'ego polega na przeprowadzeniu szeregu operacji na obrazie skutkujących otrzymaniem obrazu wybranych krawędzi, których szerokość wynosi dokładnie 1px. Kroki algorytmu:

1. Redukcja szumów przy pomocy wygładzającego filtru Gaussa.
   Zastosowano filtr 5x5 o wartościach:

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

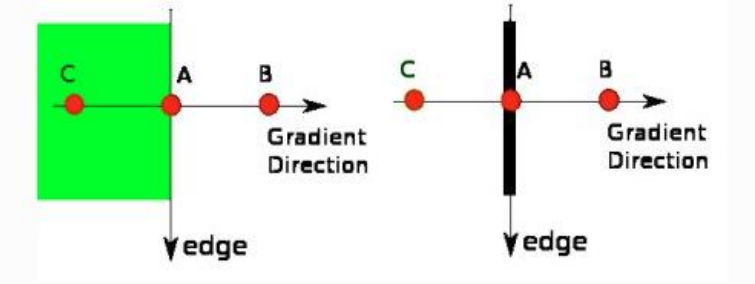
2. Znalezienie kierunku oraz wartości gradientów w obrazie dla każdego piksela. Najpierw używa się dwóch filtrów Sobela (poziomego I pionowego) do znalezienia dwóch pochodnych kierunkowych obrazu Gx i Gy.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

   Następnie przy pomocy poniższych wzorów dla każdego piksela oblicza się wartość gradientu G oraz kierunek θ:

$$G = \sqrt{G_x^2 + G_y^2} \qquad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

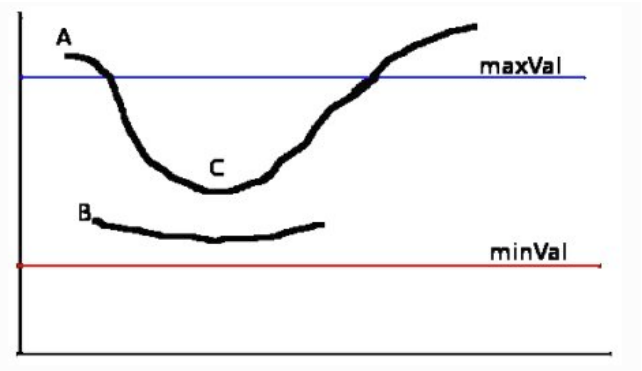
3. Non-maximum Suppression.
   Kolejnym krokiem jest usunięcie tych pikseli które nie stanowią krawędzi. Dokładniej mówiąc pozostawiamy te piksele które stanowią lokalne minimum w kierunku gradientu, usuwając wszystkie inne.
   Przykład:



Punkt A jest na krawędzi w kierunku pionowym. Kierunek gradientu jest prostopadły do krawędzi. Punkty B i C znajdują się w kierunku gradientu. Zatem punkt A jest sprawdzany z punktami B i C czy stanowi lokalne minimum. Jeżeli tak to jest traktowany jako krawędź, odpowiadającemu mu pikselowi przypisywana jest wartość 1 i przechodzi on do następnego kroku algorytmu. W przeciwnym przypadku zostaje usuwany z obrazu i zastępowany wartością 0. W rezultacie tego kroku algorytmu

otrzymujemy obraz binarny z wartościami 1 lub 0 z krawędziami o grubości dokładnie 1 piksela.

4. Hysteresis Thresholding.
   Ostatnim krokiem jest wykonanie progowania używając dwóch progów minVal and maxVal. Operacja ta ma na celu pozostawienie tylko właściwych krawędzi obiektów oraz pozbycie tych które są nieporządane. Polega to na:
   - Pozostawieniu wszystkich krawędzi o wartościach gradientu powyżej górnego progu maxVal
   - Usunięciu wszystkich krawędzi poniżej dolnego progu minVal
   - Pozostawieniu tylko tych krawędzi o wartościach gradientu pomiędzy minVal a maxVal, które są połączone z krawędziami o wartości gradient większej niż maxVal. Przykład:



   Krawędź A znajduje się powyżej wartości maxVal, zatem jest traktowana jako pewna krawędź pomimo faktu iż jej część C jest poniżej wartości maxVal. Krawędź B jest powyżej wartości minVal, ale jest poniżej maxVal oraz nie jest połączona z żadną pewną krawędzią, a zatem jest odrzucana. Prawidłowe ustawienie parametrów minVal I maxVal jest kluczowe do otrzymania oczekiwanego obrazu krawędzi.

5. Zamiana dwuwymiarowej tablicy o rozmiarze 64x64 na jednowymiarowy wektor o długości 4096.

Ostatecznie model SVM wytrenowany na cechach otrzymanych przy pomocy algorytmu Cannego radził sobie dobrze w przypadku klasyfikacji obrazów rozmiaru 64x64 osiągając skuteczność 95%, jednak kompletnie poległ przy próbie wykrywania loga na większych zdjęciach przy pomocy algorytmu Multiscale Detection. Z tego powodu zdecydowano się na zmianę podejścia i użycie algorytmu HOG.

3.2. Podejście drugie – HOG (Histogram of Oriented Gradients)

Ze względu na fakt iż pierwsze podejście nie okazało się dobrym wyborem, postanowiono użyć algorytmu HOG, który jest szeroko stosowany do detekcji obiektów na zdjęciach w połączeniu z modelem SVM. W pierwszej kolejności użyto gotowego algorytmu HOG z pakietu OpenCV w celu sprawdzenia czy wybrane podejście ma szanse powodzenia, następnie po serii owocnych prób zaimplementowano opisywany algorytm w całości.

3.2.1. Algorytm HOG

Algorytm HOG, ogólnie mówiąc, polega na użyciu jako deskryptor cech histogramów kierunków gradientów, uzyskanych w pewien specyficzny sposób z obrazu.

Kroki algorytmu:

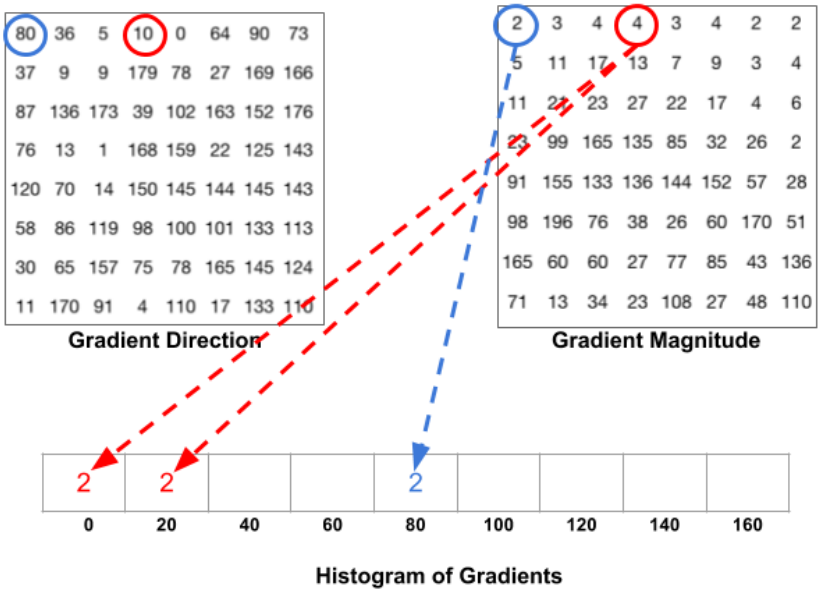
1. Znalezienie kierunku oraz wartości gradientów w obrazie dla każdego piksela.
   Najpierw używa się dwóch filtrów Sobela (poziomego i pionowego) do znalezienia dwóch pochodnych kierunkowych obrazu Gx i Gy.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

   Następnie przy pomocy poniższych wzorów dla każdego piksela oblicza się wartość gradientu G oraz kierunek θ:

$$G = \sqrt{G_x^2 + G_y^2} \qquad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

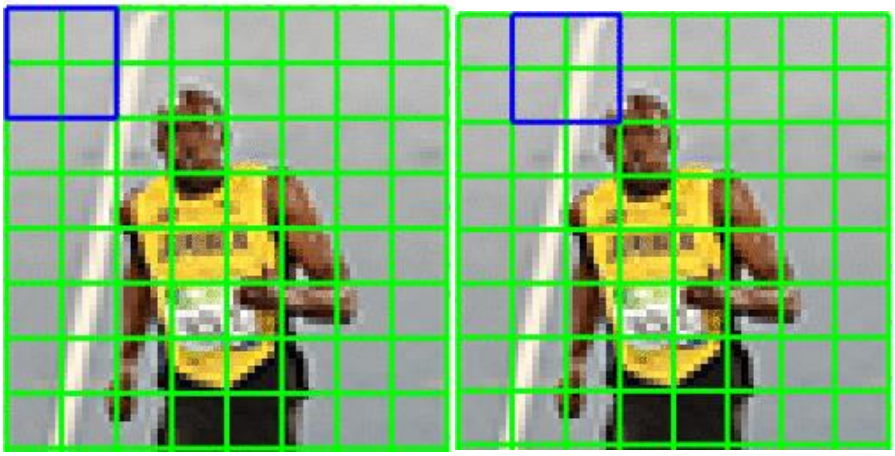
2. Obliczenie histogramów gradientów w komórkach.
   Dokonywany jest podział analizowanego obrazu o rozmiarze 64x64 na komórki o stałym rozmiarze. Wybrano podział 8 x 8. Następnie w każdej z komórek tworzony jest histogram wartości gradientów w zależności od ich kierunku dla n wybranych podziałów. Przyjęto 9 podziałów na kąty 0, 20, 40, 60, 80, 100, 120, 140 i 160 stopni. Przykład:



   Wartość gradientu w pierwszym pikselu jest równa 2, natomiast kierunek wynosi 80 stopni zatem wartość 2 w całości przydzielana jest do wartości kąta 80

w histogramie. Z kolei w pikselu czwartym wartość gradientu wynosi 4 a kierunek 10 dlatego wartość 4 jest rozdzielana po równo pomiędzy kąty 0 i 20 stopni w histogramie.

3. Normalizacja w blokach.
   Dokonywany jest podział obrazu na bloki, gdzie rozmiar bloków musi być wielokrotnością rozmiaru komórek. Maksymalny rozmiar bloku to wielkość obrazu. Przyjęto rozmiar bloku 16x16 px obejmujący 4 komórki. Normalizacja odbywa się w blokach dla wszystkich wartości histogramów uzyskanych w pkt 2 które w danym momencie obejmuje blok. Proces ten odbywa się w pętli, gdzie w każdym następnym kroku okno bloku przesuwane jest o zdefiniowaną wartość (przyjęto 8 px) i zwracane są znormalizowane wartości. Na koniec wszystkie zwrócone w ten sposób wartości są łączone i zwracane jako jednowymiarowy wektor. Przykład:



   W pierwszym obrazie blok obejmuje dwie pierwsze komórki z pierwszego rzędu oraz dwie pierwsze z drugiego. Histogramy z tych komórek są lokalnie normalizowane w bloku a ich wartości są dodawane do listy. W drugim obrazie okno bloku przesunęło się o jeden krok w prawo, ponownie dokonywana jest normalizacja i dodanie wyniku do listy. Po przejściu całego obrazu przez okno bloku lista otrzymanych wartości jest zwracana jako wynik i końcowy deskryptor cech.

3.3. SVM

Jako klasyfikatora użyto Maszyny Wektorów Nośnych (SVM) z pakietu OpenCV. Dane zostały podzielone na zbiór treningowo-walidacyjny oraz testowy w stosunku 9:1. Następnie model został wytrenowany na zbiorze treningowo-walidacyjny z walidacją k-fold z k=10. Na końcu model został jeszcze przetestowany na danych testowych. Algorytm HOG jako deskryptor cech w połączeniu z SVM okazał się strzałem w dziesiątkę. Skuteczność zarówno podczas walidacji jak i na zbiorze testowym wyniosła 100% przy wartości parametrów: C = 1 oraz γ = 1. Zarówno w przypadku algorytm z pakietu OpenCV jak i tego zaimplementowanego samodzielnie.

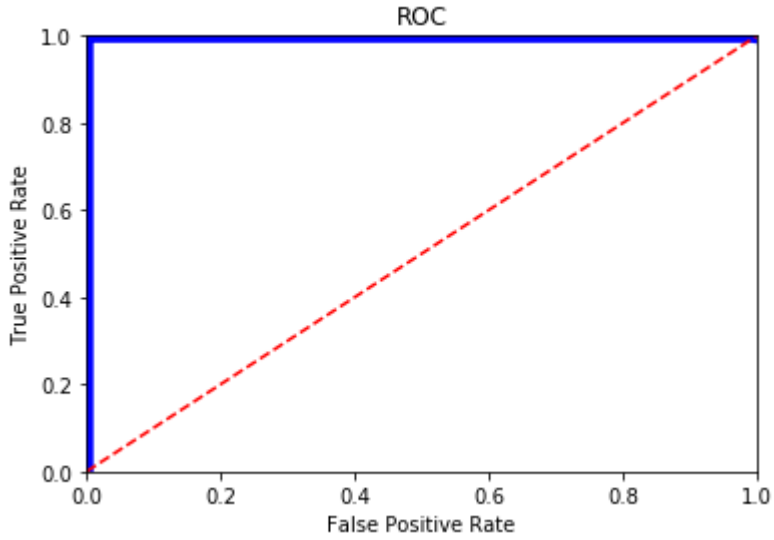
- Wyniki ewaluacji modelu na zbiorze treningowym:
  Accuraccy = 1.0
- Wyniki ewaluacji modelu na zbiorze walidacyjnym:
  Accuraccy = 1.0
- Wyniki ewaluacji modelu na zbiorze testowym:
  Recall = 1.0
  Precision = 1.0

F – score = 1.0

Confusion matrix:

	actual 1	actual 0
predicted 1	732	0
predicted 0	0	768

Krzywa ROC:



3.4. Multiscale Detection

Mając wytrenowany klasyfikator ostatnim elementem pozostało już zaimplementowanie wykrywania różnych rozmiarów loga na różnych rozmiarów zdjęciach. Postanowiono wykorzystać zmodyfikowaną metodę Multiscale Detection. Algorytm składa się z trzech zagnieżdżonych pętli:

- W pierwszej z nich zdjęcie wejściowe jest przeskalowywane do rozmiaru k * 64 piksele, gdzie k należy do przedziału <1, mniejszy wymiar obrazu wejściowego + x>. Wartość x została dobrana eksperymentalnie tak, żeby okno przechodzenia przez obraz było dostatecznie małe, aby wykrywać nawet niewielkich rozmiarów loga i jednocześnie nie wydłużać zbytnio obliczeń. Dla własnej implementacji x = 1. Dla implementacji z pakietu OpenCV x = 4.
- W drugiej i trzeciej pętli następuje przejście przez obraz oknem o wielkości 64x64 px z eksperymentalnie dobranym krokiem (8 px dla implementacji z pakietu OpenCV, 16px dla implementacji własnej) . Dla każdego wyciętego okna obliczane są cechy wg algorytmu HOG, a następnie na ich podstawie dokonywana jest klasyfikacja z użyciem wytrenowanego modelu SVM. Jeżeli wynik jest pozytywny to współrzędne rogów okna dodawane są do zainicjalizowanej na samym początku listy, która po przejściu wszystkich pętli jest zwracana.

Otrzymane w ten sposób współrzędne prostokątów, które się przecinają są ujednolicane do jednego prostokąta, tak żeby nie zamazywać obrazu po naniesieniu na niego ramek. Ostatecznie ramki są nanoszone na obraz.

3.5. Wyniki

Ostatecznie algorytm został przetestowany na sześciu wstępnie wybranych obrazach ('1.jpg', '2.jpg', '3.jpg', '4.jpg', '5.jpg', '6.jpg' w katalogu ‘evaluation’). Poniżej przedstawiono wyniki:

4. Wnioski

Podsumowując, cel jakim była implementacja programu rozpoznającego logo firmy Shell został zrealizowany z bardzo dobrymi wynikami. Skuteczność klasyfikacji na zbiorach treningowym, walidacyjnym i testowym wyniosła 100%. Implementacja algorytmu z pakietu OpenCV jest znacznie szybsza niż implementacja własna autora. Wynika to głównie z faktu iż język C++ w którym napisana jest w/w biblioteka jest językiem kompilowanym, znacząco szybszym od interpretowanego języka Python. Ponadto biblioteka ta jest nieustannie rozwijana i optymalizowana przez szerokie grono specjalistów co niewątpliwie również wpływa na jej wydajność.