Zaawansowne algorytmy wizyjne

materiały do ćwiczeń laboratoryjnych

Piotr Pawlik, Tomasz Kryjak

PUBLISHED BY AGH

First printing, March 2018

Contents

1 — Detekcja sylwetek ludzkich

1.1 Cel zajęć

- zapoznanie z zagadnieniem detekcji sylwetek ludzkich,
- zapoznanie z deskryptorem HOG (ang. *Histogram of Oriented Gradients*),
- zapoznanie z klasyfikatorem SVM (ang. *Support Vector Machine*),
- podstawy uczenia maszynowego (metodologia).

1.2 Deskryptor HOG

W implementacji detektora HOG (ang. *Histogram of Oriented Gradients*) może pomóc artykuł autorów metody Navneet'a Dalal'a i Bill'a Triggs'a pt. "Histograms of Oriented Gradients for Human Detection" (udostępniony na stronie kursu). Rozdziały 1 i 2 można przeglądnąć. Rozdział 3 **dobrze jest dokładniej przeanalizować**. Rozdział 4 będzie potrzebny później przy etapie uczenia klasyfikatora, na razie można go tylko przeglądnąć. Wyniki w rozdziale 5 to w tym momencie ciekawostka. **Najważniejszy jest rozdział 6** – dokładny opis implementacji algorytmu HOG.

W uproszczeniu - algorytm mozna podzielić na trzy etapy:

- wyliczenie gradientów
- wyznaczenie histogramów gradientów
- normalizacja histogramów

Na wstępie proszę zwrócić uwagę na parametry algorytmu:

- przestrzeń barw – RGB,
- brak korekcji gamma (ani innego przetwarzania wstępnego),
- postać operatora gradientowego,
- liczbę przedziałów histogramu, analizowane orientacje,
- rozmiar komórki (ang. *cell*,)
- normę. Uwaga zamiast L2-Hys, będziemy używać po prostu L2 (no chyba, że ktoś bardzo chce),
- rozmiar bloku (ang. *block*), zachodzenie na komórki (ang. *stride*),
- rozmiar okna detekcji.

Kolejne kroki implementacji deskryptora HOG:

1. Ze strony kursu pobierz zbiór wycinków obrazu z sylwetkami ludzi (próbki pozytywne) oraz bez ludzi (próbki negatywne).
2. Wczytaj dowolny obraz pozytywny – dla niego opracujemy implementację HOG. Można wspomniane wyżej trzy etapy zaimplementować w osobnych funkcjach.
3. Zaczynamy od obliczania gradientu (rozdział 6.2 w artykule). Dla nas najważniejsze są postać maski $[-1\ 0\ 1]$ oraz ostatni akapit. Trzeba zrealizować detekcję krawędzi pionowych i poziomych oraz policzyć amplitudę w każdym kanale RGB osobno. Do obliczenia gradientów z w/w maską mozna uzyć funkcji:

   ```
   dx = scipy.ndimage.filters.convolve1d(np.int32(im), np.array([-1, 0,
       1]),1)
   dy = scipy.ndimage.filters.convolve1d(np.int32(im), np.array([-1, 0,
       1]),0)
   ```

   Po obliczeniu gradientów dla trzech składowych należy wybrać z nich te, których wartość jest największa. Można to robić pętlą, ale będzie to nieefektywne. Proponuję wykorzystać możliwość Pythona, w której z tablicy wybierane są wartości spełniające warunek. Przykładowo:

   ```
   grad=gradB.copy()       # poczatkowo wynikowa macierz to gradient
       skladowej B
   m1 = gradB-gradG # m1 - tablica pomocnicza do wyznaczania maksimum
       miedzy skladowymi B i G
   grad[m1<0]=gradG[m1<0] # w macierzy wynikowej gradienty skladowej B sa
       podmieniane na wieksze od nich gradienty skladowej G
   ```

   Następnie analogicznie znajdujemy większe gradnienty z akrualnej macierzy wynikowej i gradientów składowej R. Powiedzmy, że użyta tu tablica pomocnicza bedzie się nazywała m2.
   Ten sam mechanizm należy wykorzystać w wyznaczaniu macierzy orientacji gradietów (przy czym korzystamy z wyznaczonych uprzednio tablic pomocniczych m1 i m2).
4. Drugi etap - wyznaczanie histogramów w blokach (rozdział 6.3) to najtrudniejszy element implementacji (technicznie, bo koncepcyjnie dość prosty). Na początku trzeba zdefiniować rozmiar komórki (8) oraz obliczyć rozdzielczość w układzie współrzędnych związanym z komórkami (np. `YY_cell= np.int32(YY/cellSize)`). Następnie tworzymy kontener na histogramy – jeden histogram dla każdej z komórek. Objaśnienie do kodu znajdującego się poniżej - w pętli po komórkach realizujemy następujące operacje:
   - Pobieramy odpowiednie fragmenty z macierzy z amplitudą i fazą.
   - Iterujemy po otoczeniu.
   - Po pierwsze trzeba obsłużyć przypadek ujemnych kątów (poprzez dodanie 180, jeśli rozpatrujemy wartości w stopniach).
   - Po drugie trzeba określić do jakiego przedziału histogramu należy rozważany kąt. W tym celu należy przedział [0;180] podzielić na 9 części i dla każdej z nich wyznaczyć element "środkowy". Następnie trzeba wyznaczyć najbliższy mniejszy środek przedziału dla rozważanego kąta – da się to zrobić w jednej linijce kodu (trzeba trochę pokombinować). Uwaga. Proszę pamiętać o specjalnym traktowaniu przejścia 180 – 0 (zawijanie).
   - Wyznaczony przedział (kąt) należy zamienić na numer przedziału w histogramie (adres w tablicy).
   - Następnie trzeba obliczyć odległość od rozważanego kąta do środka przedziału (mniejszego). Uwaga. Tu również należy pamiętać o specjalnym traktowaniu wartości zbliżonych do 180.
   - Finalnie inkrementujemy dwa przedziały histogramu (sąsiadów rozważanego kąta). Uwaga. Dodawana jest wartość amplitudy rozdzielonej proporcjonalnie pomiędzy

dwa przedziały (interpolacja liniowa).

```
# Obliczenia histogramow
cellSize = 8  # rozmiar komorki
YY_cell= np.int32(YY/cellSize)
XX_cell= np.int32(XX/cellSize)

# Kontener na histogramy - zakladamy, ze jest 9 przedzialow
hist = np.zeros([YY_cell,XX_cell,9],np.float32)

# Iteracja po komorkach na obrazie
for jj in range(0,YY_cell):
 for ii in range(0,XX_cell):
  # Wyciecie komorki
  M = SXY[jj*cellSize:(jj+1)*cellSize,ii*cellSize:(ii+1)*cellSize]
  T = DIR[jj*cellSize:(jj+1)*cellSize,ii*cellSize:(ii+1)*cellSize]
  M = M.flatten()
  T = T.flatten()

      # Obliczenie histogramu
      for k in range (0,cellSize*cellSize):
  m = M[k];
        t = T[k];

      # Usuniecie ujemnych kata (zalozenie katy w stopniach)
        if (t < 0):
         t = t + 180

       # Wyliczenie przezdialu
        t0 = np.floor( (t-10)/20 )*20 +10; # Przedzial ma rozmiar 20,
    srodek to 20

       # Przypadek szczegolny tj. t0 ujemny
        if (t0 < 0):
          t0 = 170

       # Wyznaczenie indeksow przedzialu
        i0 = int((t0-10)/20);
        i1 = i0+1;

       # Zawijanie
        if i1 == 9:
          i1=0;

       # Obliczenie odleglosci do srodka przedzialu
        d = min(abs(t-t0), 180 - abs(t-t0) )/20

       # Aktualizacja histogramu
        hist[jj,ii,i0] = hist[jj,ii,i0] + m*(1-d)
        hist[jj,ii,i1] = hist[jj,ii,i1] + m*(d)
```

Dla przyspieszenia można operacje w pętli k spróbować zrealizować macierzowo, wówczas w pętli k pozostanie tylko:

```
diff = abs(T-t0)
in range (0,cellSize*cellSize):
min(diff[k], 180 - diff[k] )/20
1[jj,ii,i0[k]] += M[k]*(1-d)
1[jj,ii,i1[k]] += M[k]*(d)
```

5. Ostatni etap to normalizacja (rozdział 6.4 – *Block Normalization schemes*). Jest on stosunkowo prosty. W pętli po współrzędnych komórek (rozmiar pomniejszony o 1) należy

pobrać 4 histogramy. Następnie należy je zestawić w jeden wektor (`np.concatenate`), obliczyć normę (funkcja `np.linalg.norm`) i dokonać normalizacji z zastosowaniem L2. Znormalizowany wektor stanowi element finalnego wektora cech.

```
# Normalizacja w blokach
e = math.pow(0.00001,2)
F = []
for jj in range(0,YY_cell-1):
        for ii in range(0,XX_cell-1):
          H0 =  hist[jj,ii,:]
                H1 =  hist[jj,ii+1,:]
                H2 =  hist[jj+1,ii,:]
                H3 =  hist[jj+1,ii+1,:]
                H = np.concatenate((H0, H1, H2, H3))
                n = np.linalg.norm(H)
                Hn = H/np.sqrt(math.pow(n,2)+e)
                F = np.concatenate((F,Hn))
```

Na końcu powinniśmy otrzymać wektor 3780 liczb. Pierwsze 10 wartości dla obrazka *per00060.ppm*:
0.272706441466276
0.151429648114036
0.0738968298729655
0.112083078676680
0.151337880813270
0.0733459422230665
0.0581915714908520
0.199857893696497
0.127544372316149
0.142256183667681
0.0789572641899305

Mogą też Państwo wykorzystać poniższy kod do wświetlenia gradientów histogramów (przed ich normalizają).

```
def HOGpicture(w, bs): # w - histogramy gradientow obrazu, bs - rozmiar
komorki (u nas 8)
    bim1 = np.zeros((bs, bs))
    bim1[np.round(bs//2):np.round(bs//2)+1,:] = 1;
    bim = np.zeros(bim1.shape+(9,));
    bim[:,:,0] = bim1;
    for i in range(0,9):  #2:9,
      bim[:,:,i] = scipy.misc.imrotate(bim1, -i*20,'nearest')/255

    Y,X,Z = w.shape
    w[w < 0] = 0;
    im = np.zeros((bs*Y, bs*X));
    for i in range(Y):
      iisl = (i)*bs
      iisu = (i+1)*bs
      for j in range(X):
        jjsl = j*bs
        jjsu=(j+1)*bs
        for k in range(9):
          im[iisl:iisu,jjsl:jjsu] += bim[:,:,k] * w[i,j,k];
    return im
```

Finalnie proszę stworzony kod (wygodny na etapie implementacji) „obudować” w funkcję, która będzie pobierać obrazek (macierz), a zwróci wyznaczony wektor cech (znormalizowane histogramy gradientów).

1.3 Klasyfikator SVM

Mając daną funkcję do obliczania deskryptora HOG zaimplementuj uczenie klasyfikatora SVM.

1. Zaczynamy od przygotowania zbioru danych. Ze strony kursu ściągnij folder `pedestrians`. Zawiera on próbki pozytywne (z pieszymi) i negatywne (bez pieszych).
2. Dla obrazów z obu zbiorów wyliczmy deskryptor HOG (dla pierwszych 100).

```
HOG_data = np.zeros([2*100,3781],np.float32)

      for i in range(1,100):
         IP = cv2.imread('pos/per%05d.ppm' % i)
         IN = cv2.imread('neg/neg%05d.png' % i)
         F = hog(IP)
         HOG_data[i,0] = 1;
         HOG_data[i,1:] = F;
         F = hog(IN)
         HOG_data[i+100,0] = 0;
         HOG_data[i+100,1:] = F;
```

   Dane zawierają oprócz wektor indeks klasy (1 – pieszy, 0 – nie pieszy).
3. Zaczynamy od uczenia. Rozdzielamy danej wejściowe na wektor etykiet (pierwsza kolumna) i dane.

```
labels = HOG_data[:,0];
data = HOG_data[:,1:]
```

   Do pliku dołączamy `from sklearn import svm`. Tworzymy obiekt SVM: `clf = svm.SVC(kernel='linear', C = 1.0)`. Przeprowadzamy uczenie: `clf.fit(data,labels)`. Testujemy: `lp = clf.predict(data)`.
4. Analizujemy wyniki uczenia. Na zbiorze uczącym przeprowadzamy klasyfikację. Liczymy współczynniki TP, TN, FP, FN. Wyniki dobrego uczenia powinny być zbliżone do 100 %.
5. Przeprowadzamy walidację rozwiązania. Zasadniczo powtarzamy powyższy eksperyment. To jest moment na wprowadzanie ew. poprawek:
   - zmiana parametru C,
   - zmiana jądra itp.

   Oczywiście po wprowadzaniu zmian każdorazowo należy sprawdzić ich efekt na zbiorach: testowym i walidacyjnym.
6. Ostatecznie, jak już jesteśmy pewni, że dana wersja klasyfikatora osiąga najlepsze wyniki, to przeprowadzamy test na zbiorze testowy. Jego wyniki uznajemy za “ostateczny” dla klasyfikatora. Tak przynajmniej jest poprawnie metodologicznie.

W tym momencie mamy już wszystkie narzędzia do realizacji kompletnego systemu detekcji: deskryptor HOG i nauczony klasyfikator SVM. Możemy zatem zaimplementować detekcję na obrazie rzeczywistym.

1. Na stronie kursu zamieszczone są cztery obrazy testowe (`testImage1-4.png`). Ale równie dobrze można użyć dowolnych “z Internetu”.
2. Na początek warto sprawdzić, czy wysokość postaci na zdjęci mniej więcej odpowiada rozmiarowi naszego okna detekcji. Można to zrobić ręcznie (“zmierzyć wysokość w

pikselach") lub automatycznie (nanieść okno detekcji na obraz). Druga metoda jest o tyle lepsza, że i tak będziemy potrzebować funkcji rysowania prostokąta na obrazie (do wizualizacji detekcji). Dla przypomnienia *cv2.recangle*.

3. Następnie musimy dostosować rozmiar obrazka (funkcja `cv2.resize`) do wysokości postaci. Warto sobie przeglądać jak wyglądają sylwetki w zbiorze uczącym – pozwoli nam to zorientować się "jak mógł się nauczyć klasyfikator".
4. Implementujemy pętlę po obrazie. W każdej iteracji pobieramy wycinek o rozmiarze 64×128, obliczamy dla niego deskryptor HOG, a następnie dokonujemy klasyfikacji. Jeśli jest ona pozytywna to nanosimy prostokąt na obraz (oczywiście pomocniczy, żeby nie zakłócić detekcji w kolejnych lokalizacjach, kopię robimy za pomocą metody copy). **Uwaga**. Krok (czyli to o ile przesuwamy okno w pionie i poziomie) nie powinien wynosić 1, bo możemy nie doczekać się na wynik. Najlepiej dobrać go empirycznie, ale wartości 8 lub 16 wydają się odpowiednie.
5. Spróbuj, czy wykrywanie działa na załączonych obrazkach, ew. na własnych.
6. Przetwarzanie w wielu skalach **(nieobowiązkowe)**.
   Poprzednio dobieraliśmy odpowiednią skalę empirycznie/doświadczalnie ("na oko"). Teraz przetwarzamy obraz w wielu skalach. Przykładowo zaczynamy od rozdzielczości 1 i każdorazowo zmniejszamy o 10%. Sposób budowy piramidy należy dobrać empirycznie. W każdej skali realizujemy detekcję. **Uwaga**. Należy zdawać sobie sprawę, że można budować piramidę także "w górę" tj. zwiększać rozdzielczość obrazu wejściowego. Pozwala to na detekcję małych sylwetek (aczkolwiek skuteczność takiego podejścia nie jest bardzo duża). **Uwaga.** Do testów proszę sobie dobrać obraz o względnie małych rozmiarach na którym występują dwie różniące się rozmiarami sylwetki. Przykładowo `testImage4.png`. Proszę jednak nie oczekiwać zupełnie bezbłędnego działania systemu. Przede wszystkim dlatego, że zbiór próbek negatywnych nie był "zbyt wymagający".
7. Na koniec należy jakoś rozwiązać kwestię wizualizacji i ew. integracji detekcji. Najlepiej zapamiętać współrzędne okien, gdzie wykryto sylwetki, przeskalować je do rozdzielczości podstawowej i wyrysować.

Bibliography