

Wstęp do multimediiów

Laboratorium 7

Analiza obrazu – detekcja twarzy

1. Cel i zakres ćwiczenia

Celem ćwiczenia jest zapoznanie studentów z wybranymi algorytmami umożliwiającymi wykrywanie twarzy w obrazach. W ćwiczeniu zostaną wykorzystane algorytmy zaimplementowane w bibliotekach OpenCV [1] oraz Dlib [2]. Ćwiczenie zostanie zrealizowane w środowisku Google Colab [3], w którym dostępne są wszystkie wymagane biblioteki.

Możliwe jest również wykonanie ćwiczenia w lokalnym środowisku Python. Najwygodniejsze jest pobranie i zainstalowanie dystrybucji Pythona zawierającej już najpopularniejsze pakiety, np. Anaconda, która w wersji „Individual Edition” dostępna jest za darmo (<https://www.anaconda.com/products/individual>). Standardowo Anaconda zawiera wiele popularnych pakietów Pythona, ponadto zawiera wygodny program narzędziowy do przeglądania i instalowania nowych pakietów – conda. Po pobraniu i zainstalowaniu środowiska Anaconda konieczne jest doinstalowanie biblioteki OpenCV (polecenie: `conda install opencv` lub `pip install opencv-python` w konsoli Anaconda Prompt) oraz biblioteki Dlib (polecenie: `conda install dlib` w konsoli Anaconda Prompt).

2. Wprowadzenie

Jednym z pierwszych algorytmów wykrywania twarzy w obrazach był algorytm zaproponowany przez P. Viola i M. Jones [4]. Algorytm ten, mimo iż obecnie znanych jest wiele znacznie bardziej efektywnych rozwiązań, nadal jest wykorzystywany, gdyż umożliwia detekcję twarzy w czasie rzeczywistym przy niewielkich nakładach obliczeniowych, co jest istotne np. w przypadku implementacji w systemach wbudowanych. Działanie algorytmu polega na przeszukiwaniu obrazu i wykrywaniu regionów o określonych cechach reprezentowanych falkami Haara. Cechy te zawierają informacje o zmianie wartości kontrastu pomiędzy prostokątnymi grupami pikseli. Wykryte cechy są następnie analizowane przez kaskadę klasyfikatorów, które wybierają regiony zawierające cechy zgodne z modelem twarzy wyuczonym na zbiorze obrazów trenujących.

Wśród klasycznych (tzn. nie używających sieci neuronowych) algorytmów detekcji twarzy za jeden z najbardziej efektywnych uważany jest algorytm, w którym jako wektor cech używany jest histogram zorientowanych gradientów (*Histogram of Oriented Gradients, HOG*) [5], a jako klasyfikator maszyna wektorów nośnych (*Support Vector Machine, SVM*) [6]. Wektor cech (HOG) wyznaczany jest na podstawie gradientów wyznaczonych w kierunku poziomym i pionowym dla rozłącznych bloków składowej luminancji obrazu. Klasyfikator (SVM) jest trenowany na

reprezentatywnym zbiorze wektorów cech zawierającym zarówno przykłady pozytywne (tzn. zawierające twarze) jak i negatywne.

Współczesne algorytmy detekcji twarzy wykorzystują splotowe sieci neuronowe (*Convolutional Neural Networks, CNN*). W algorytmach tych stosowane są różnorodne architektury sieci neuronowych, do trenowania których wykorzystywane są bazy zawierające tysiące obrazów. W ćwiczeniu zostanie wykorzystana sieć Max-Margin Object Detection (MMOD) [8], której implementacja dostępna jest w bibliotece Dlib.

3. Przebieg ćwiczenia, zadania do realizacji

Kody źródłowe niezbędne do realizacji ćwiczenia (w formacie notatnika .ipynb do uruchomienia w Google Colab, oraz skryptu .py do wykonania w lokalnym środowisku Pythona) wraz z potrzebnymi modelami i przykładowymi obrazami testowymi udostępnione są na platformie Teams.

Po uruchomieniu notatnika popierany jest przykładowy obraz testowy z bazy Wider [9] oraz zbiory danych (modele) dla kaskady Haara i MMOD. Następnie w obrazie testowym przy użyciu trzech detektorów wykrywane są twarze.

1. Zaproponuj miary efektywności algorytmów detekcji twarzy, np.:
 - procent poprawnie wykrytych twarzy w obrazie (liczba twarzy poprawnie wykrytych w stosunku do liczby wszystkich twarzy),
 - procent/liczba niewykrytych twarzy w obrazie,
 - liczba/procent obiektów niepoprawnie rozpoznanych jako twarze (procent: liczba niepoprawnych detekcji w stosunku do liczby wszystkich detekcji).

Wyznacz wartości tych miar dla obrazu testowego, podaj również całkowitą liczbę twarzy na zdjęciu.

Wyniki zbierz w tabeli ułatwiającej ich porównanie.

Uwaga: zliczając twarze w obrazie uwzględnij również twarze częściowo przysłonięte, obrócone, rozmyte, widoczne z profilu i od tyłu, występujące w tle, na rysunkach, itp.

Uwaga: wraz z przykładowymi obrazami testowymi z bazy WIDER udostępniony jest również ich opis (tzw. *ground truth*) – można go wykorzystać do sprawdzenia i porównania jakie wzorcowe twarze zostały oznaczone na tych obrazach.

2. Pobierz dodatkowe obrazy testowe i wyznacz dla nich miary zaproponowane w pkt. 1. Sprawdź działanie detektorów na co najmniej trzech innych obrazach testowych (przykładowe obrazy testowe dostępne są na platformie Teams, można wykorzystać również własne obrazy), w tym przynajmniej jednego „trudnego” (dużo twarzy, twarze różnych rozmiarów, mocno przysłonięte, rozmyte, obrócone, itp. – dla prostych przypadków bez problemu można uzyskać detekcję poprawną w 100%).
3. Wyznacz miary zaproponowane w pkt. 1 łącznie dla wszystkich testowanych obrazów.
4. Wprowadź odpowiednie modyfikacje w kodzie źródłowym i zmierz czasy wykonania poszczególnych algorytmów. Wyniki zbierz w tabeli.

Uwaga: należy zmierzyć czas detekcji, bez wstępnych przygotowań (np. ładowanie modelu) i bez przetwarzania i wyświetlania wyników.

Uwaga: w środowisku Google Colab biblioteka Dlib wykorzystuje wspomaganie GPU; domyślna instalacja biblioteki Dlib w lokalnym środowisku Pythona nie wykorzystuje wspomaganie GPU – ma to ogromny wpływ na czas działania algorytmu bazującego na CNN i należy o tym pamiętać omawiając wyniki.

5. Uzyskane wyniki (obrazy z wynikami detekcji, wartości miar z pkt. 1, 2, 3 oraz czasy wykonania z pkt. 4) zamieść w sprawozdaniu. Porównaj i skomentuj uzyskane wyniki, opisz wnioski.

Uwaga: sprawozdanie może mieć formę notatnika .ipynb z widocznymi wynikami działania oraz dodatkowymi polami tekstowymi zawierającymi zebrane wyniki, komentarze, wnioski.

Literatura

1. <https://opencv.org/>
2. <http://dlib.net/>
3. <https://colab.research.google.com/>
4. Paul Viola , Michael Jones, "Rapid object detection using a boosted cascade of simple features", *Conference on Computer Vision and Pattern Recognition* (2001)
5. <https://learnopencv.com/histogram-of-oriented-gradients/>
6. https://docs.opencv.org/3.4/d1/d73/tutorial_introduction_to_svm.html
7. <https://learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/>
8. Davis E. King, Max-Margin Object Detection}, *Computer Vision and Pattern Recognition* [cs.CV], 2015, <http://arxiv.org/abs/1502.00046>
9. <http://shuoyang1213.me/WIDERFACE/>