Informatyka, studia dzienne, mgr II st. semestr I

Obliczenia inteligentne **2016/2017**

Prowadzący: dr inż. Arkadiusz Tomczyk środa, 10:15

Data oddania: _____ Ocena: _____

Weronika Hryniewska 214906

Zadanie IV: rozwiazanie problemu dla bazy obrazów znaków drogowych

1. Przygotowanie srodowiska

Cały napisany kod znajduje sie na moim Githubie: github.com/Hryniewska/OI

Znajdując się w katalogu ze środowiskiem rozwiązanie należy uruchomić w następujący sposób: *run.sh solution*2 > *result.txt*

2. Ewaluacja rozwiazania

2.1. Parametryzacja

Zapoznanie się z możliwościami rozwiązania tego problemu. Opis zadania sugeruje użycie sieci neuronowych. Jednak patrząc na trudności, jakie mieli inni w dobrej ich implementacji oraz wysokiej rozpoznawalności elementów na zdjęciach, może nie jest to jednak rozwiązanie, którego chciałabym się podjąć.

Moje rozwiązanie rozpoznaje znaki drogowe, dzięki odpowiedniemu przetworzeniu obrazu. Zostało to wykonane przy użyciu biblioteki OpenCV w jeżyku C++.

Aby jak najlepiej dobrać parametry przetwarzające obraz dla danego rozwiązania, został napisany skrypt w Bashu, który uruchamiał rozwiązanie z różnymi parametrami. Skrypt nosił nazwę parametrize.sh. Dobierał on parametr na podstawie danych z ../../environment/data/train/images/ i zapisy-

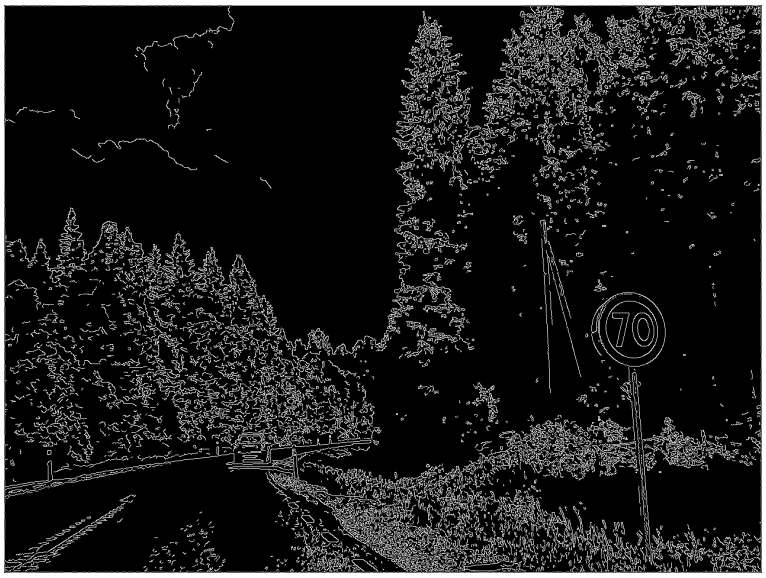
wał wyniki do w pliku tekstowego w katalogu ./data. W tym celu należało poprawić kod programu, aby wywoływał się z jeszcze jednym parametrem np. -param 60. Potem napisany skrypt dla każdego param z przedziału 10, 20, ..., 90 zrobił identyfikację znaków drogowych z parametrem -param $p i zapisał do pliku wynik_$$p$.txt.

Następnie należało przejrzeć wygenerowane pliki i wybrać ręcznie ten, który daje najlepszy wynik - 80. Zatem ten wynik został zapisany w ./data-/param.txt. Teraz skrypt run.sh wywoływał rozwiązanie z parametrem pobranym z pliku tj. ./bin/run –solution -param cat data/param.txt.

Wobec tego program dobierał parametr, który sprawiał, że najwięcej znaków drogowych była rozpoznawana poprawnie. Skrypt run.sh już tylko testował wyniki na zbiorze testowym, a nie optymalizował. Na zbiorze testowym z zasady nie optymalizuje się, do tego służy zbiór treningowy.

Na podstawie analizy miejsc, w których zostały wykryte przypuszczalne lokalizacje znaków, można stwierdzić, że im mniejszy był parametr, tym więcej znaków zostało wykrytych. Poniżej zaprezentowano przykłady przetworzonych obrazów w zależności od parametru.

Rysunek 1. Obraz otrzymany przy parametrze równym 20



Rysunek 2. Obraz otrzymany przy parametrze równym 80

2.2. Zmiana środowiska

W międzyczasie okazało się, że zaszła konieczność zmiany komputera, na którym pracowałam. Po postawieniu systemu operacyjnego Linux, okazało się, że niestety napisane wcześniej środowisko, a dokładnie biblioteki i makefile są niestety niekompatybilne do nowszej wersji systemu. Bardzo dużo czasu zajęło znalezienie drobnych różnic i poprawienie ich.

Okazało się również, że jedno zdjęcie musiało zostać usunięte z bazy testowej, ponieważ okazało się uszkodzone.

Wtedy także okazało się, że napisany wcześniej program testowy nie do końca pasuje do tego typu rozwiązania i nie wszystkie jego funkcjonalności działają poprawnie. Pomijał np. niektóre poprawnie rozpoznane znaki. Przypuszczalnie mogło to na przykład w pewnym stopniu wynikać z nie do końca właściwego działania programu przy innej wersji bibliotekach. Poprzednia metoda sprawdzająca operała się na opisanej poniżej metodzie oceny.

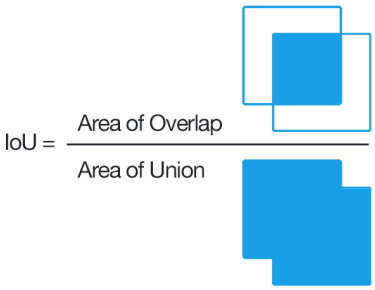
Poprawność była oceniana przy pomocy znaków + i -. + oznacza, że rozwiązanie nie wykracza poza próg tolerancji błędu ustalony tutaj na 20%. Ocena obliczana jest według następującego wzoru:

$$rating = \frac{|C|}{|T|} - \frac{|S - C|}{|T|} \tag{1}$$

gdzie T to pole rozwiązania testowego, S to pole rozwiązania otrzymanego przez program, a $C = T \cap S$ jest wspólną częścią rozwiązania wzorcowego i uzyskanego w programie.

Postanowiono nie poprawiać tamtego kodu, tylko napisać środowisko testujące z trochę inną metodą oceny. Jako metodę oceny wybrano Intersection

of Union, która jest opisana w zastosowaniu właśnie do oceny poprawności lokalizowania znaków drogowych w artykule podanym w przypisie[1].



Rysunek 3. Graficzne przedstawienie nowej metody oceny[1]

Wzór matematyczny nowej metody oceny:

$$IoU = \frac{T \cap S}{T \cup S} \tag{2}$$

gdzie T to pole rozwiązania testowego, a S to pole rozwiązania otrzymanego przez program

Powołując się na przytoczony wcześniej artykuł założono, że jeśli IoU przekroczy 0.5, wtedy wynik rozpoznania jest określany jako 'dobry'.

Rysunek 4. Poprawność lokalizacji rozpoznanego znaku[1]

2.3. Dokładniejsze dane

Sam procentowy wynik poprawności działania programu nie pozwala na głębsze przeanalizowanie problemu rozpoznawania znaków drogowych. Zatem postanowiono dopisać fragment do środowiska, który umożliwiłby eksportowanie do pliku bardzo szczegółowych danych uzyskiwanych podczas zmian parametru w argumencie funkcji cv::Canny() w OpenCV. Wygląd struktury pliku był następujący:

```
 wartosc parametru 10
+ nie bylo znaku 188
```

[1] www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection

```
+ rozpoznano znak 422
liczba znakow w moim rozwiazaniu 3195
liczba znakow do rozpoznania 3655
wszystkich sprawdzen 6142
do 50% 94
<50-60) 120
<60-70) 113
<70-80) 18
<80-90) 46
<90-100> 31
```

gdzie:
— wartosc parametru - parametr funkcji cv::Canny(), która służy do wykrywania krawędzi. jest to parametr określający iloczyn minimalnego progu oraz pewnego współczynnika
— + nie bylo znaku - na zdjęciu nie było znaku i go nie rozpoznano
— + rozpoznano znak - ile razy znak został dobrze zlokalizowany
— liczba znakow w moim rozwiazaniu - ile znaków rozpoznało moje rozwiązanie
— liczba znakow do rozpoznania - ile znaków było w pliku referencyjnym
— wszystkich sprawdzen - w przypadku znalezienia kilku znaków na jednym zdjęciu, należało sprawdzić, czy którykolwiek za znaków nie pokrywa się z którymkolwiek znakiem referencyjnym na danym zdjęciu
— wartości procentowe - w jakim stopniu znaki do wykrycia pokrywały się ze znakami wykrytymi

Ponadto wygenerowano również pliki, które reprezentowały IoU dla poszczególnych zlokalizowanych znaków, jeśli znaleziony znak miał jakąkolwiek część wspólną ze znakiem referencyjnym. Prezentowane pola powierzchni podano w jednostkach piksele do kwadratu:

```
 4092 / 6545 = 62%
6478 / 7140 = 90%
5925 / 9600 = 61%
3822 / 43914 = 8%
1936 / 2296 = 84%
```

2.4. Wnioski pośrednie

Wnioski wysnuto na podstawie pliku "wyniki_koncowe.txt", którym uwzględnione są wyniki rozpoznawania uzyskiwane dla różnych wartości parametru.

Zatem jeśli zaproponowane rozwiązanie zlokalizowało już położenie znaku, to zlokalizowało go bardzo dobrze. Wskazuje na to, stosunkowo niskia wartość "do 50%", która potrafi wynosić zaledwie 8,15% wszystkich zlokalizowanych znaków. Problem tego rozwiązania stanowi lokalizacja znaku na zdjęciu. Ze względu na bardzo różne wartości kontrastu, jasności, rozmycia zdjęcia, niesamowicie trudne wydaje się takie dobranie wszystkich parametrów występujących w rozwiązaniu tak, aby możliwie wszystkie jak najwięk-

sza liczba znaków została zlokalizowana.

Akltualnie najlepszym uzyskiwanym wynikiem procentowym było 17% skuteczności dla parametru o wartości 10.

2.5. Ewaluacja przetwarzania obrazu

W celu poprawy wyników rozpoznawania znaków drogowych przeanalizowano kilka podobnych rozwiązań[2], które polegały na rozpoznawaniu obiektów na obrazach. Zauważono, że przydatną operacją może być wyrównanie histogramu, które szczególnie w wypadku zdjęć o niskim kontraście, pomoże lepiej rozpoznawać kontury między np. zielenią a znakami drogowymi. Ponadto zaimplementowano również delikatne rozmycie, które w przypadku nie do końca równych kontrurów znaków, powinno pozwolić na ich lepszą identyfikację.

Dla pewności, że rozmycie rzeczywiście również poprawia otrzymywane wyniki, na chwilę usunieto funkcję blur() z programu. Otrzymane wyniki wskazywały jasno, że zarówno jedna, jak i druga operacja poprawiły rozpoznawanie znaków drogowych.

Ze względu na dużą czasochłonność, postanowiono na razie na analizę tylko dla parametru równego 60. Jeśli zostanie zauważona poprawa dla tego parametru, to trzeba będzie przeprowadzić te operacje również dla innych wartości tego parametru.

Wynik skuteczności rozwiązania dla parametru 60 z rozwiązaniem zawierającym wyrównanie histogramu i rozmycie to trochę ponad 17%. Natomiast dla rozwiązania z samym wyrównaniem histogramu to 15,7%,co stanowi wynik gorszy niż w przypadku braku obu z tych funkcji. Dla rozwiązania z samym rozmyciem wynik jest bardzo nieznacznie lepszy niż bez niego.

Akltualnie najlepszym uzyskiwanym wynikiem procentowym było 18,2% skuteczności dla parametru o wartości 10.

2.6. Dalsza ewaluacja przetwarzania obrazu

Po rozmowie z koleżanką, dowiedziałam się, że czasami gorsze wyniki uzyskuje się na zdjęciach o dużej rozdzielczości, ze względu na zbędne szczegóły, które na nich występują. Z tego powodu postanowiłam zmniejszyć długość i szerokość obrazu o 50%. Wskutek tego wzrośnie też znacznie szybkość obliczeń, która ograniczona jest głównie ze względu na liczbę i wielkość przetwarzanych obrazów. Jednak pomysł ten nie przyniósł dobrych rezultatów. Wyniki lokalizowania znaków (mimo zmiany niektórych parametrów) były prawie równe zero.

[2] http://docs.opencv.org/2.4/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html

Po dokładniejszym przejrzeniu zdjęć testowych, stwierdziłam, że są one stosunkowo ciemne - szczególnie drzewa. Zatem zostało wstawione progowanie[3], a usunięte rozciągnięcie histogramu. Przyniosło ono pozytywny rezultat.

2.7. Optymalizacja wielu parametrów

Kolejnym krokiem w celu polepszenia działania programu do przetwarzania obrazu jest optymalizacja kilku parametrów. Aby poprzednie rozwiązania nie przestały działać, parametry zostały dodane jako opcjonalne.

Skrypt do optymalizacji został napisany w Pythonie przy użyciu biblioteki Scipy, a dokładnie skorzystano ze scipy.optimize. Uruchamia on moje rozwiązanie ze wstępnymi parametrami początkowymi: "–param 40" oraz "–param2 30". Następnie uruchamia zbudowany "run" z folderu environment/bin i parsuje plik "wyniki_koncowe.txt szukając linijki, gdzie jest napisane: "poprawnosc w %:". Zczytuje liczbę, która pojawiła się w tej linijce i zwraca ją do wybranego z biblioteki Scipy algorytmu (ze znakiem minus, bo zależy nam na maksymalizacji, a to jest algorytm minimalizujący). W zależności od tego, czy nastąpi poprawa dokładności, dobierane są nowe parametry i ponownie wykonywane są wszystkie kroki, która mają na celu sprawdzenie nowej dokładności. Algorytm zatrzymuje się, gdy zostaną spełnione warunki stopu - w zależności od osiągniętej liczby iteracji, albo gdy w kilku kolejnych iteracjach nie ma zauważalnej poprawy dokładności.

Stosując ten algorytm można by zoptymalizować nawet 20 parametrów, jeśliby posiadało się szybki i wydajny sprzęt lub zostawiło komputer z uruchomionym skryptem na długi czas.

Jednak w przypadku przetwarzania obrazu liczy się nie tylko idealny dobór parametrów, ale też ich kolejość, czy też to, która z wielu podobnych funkcji zostanie wybrana.

Ze względu na dość duży zakres wartości parametrów, optymalizacja nie została przeprowadzona do końca. Ale dzięki drukowaniu zmienianych parametrów na tablicę i uzyskiwanej przy tym dokładności, wybrano parametry dające lepsze rezultaty.

3. Wnioski

Podsumowując, niestety zdjęcia często mają bardzo kiepską jakość - są niewyraźne, zawierają dużo szumu i histogram jest bardzo przesunięty w kierunku czerni. Czasami ja sama, gdy patrzę na zdjęcie, to potrzebuję dłuższej chwili, aby zobaczyć znak. Najlepiej byłoby zmienić bazę na lepszą, ponieważ to jest aż dziwne, żeby były tak kiepskiej jakości - nawet te robione komórką. Ponadto często zaznaczone są w annotacjach znaki, które są na tyle daleko, że nie ma praktycznie szans, żeby rozpoznać nawet ludzkim okiem, co to jest

[3] www.docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#threshold

za znak.

Sądzę, że graniczną wartością, którą można by osiągnąć, stosując moje rozwiązanie po pewnych modyfikacjach, jest wartość około 30%-45%. Jednak ze względu na bardzo dużą długotrwałość każdej z tych operacji, trzeba by na pewno zrównoleglić obliczenia. Można by na przykład sprawdzać jeszcze kolor znaku o danym kształcie. Jednak to poprawiłoby detekcję znaków tylko nieznacznie. Znacznie trudniejsze jest bowiem samo zlokalizowanie znaku, który często jest nierównomiernie oświetlony lub w jakiś sposób zlewa się z otoczeniem.

Gdybym wtedy kiedy zaczynałam robić to rozwiązanie, miała taką wiedzę jak teraz, to wydaje mi się, że lepiej wybrałabym narzędzie do zrobienia tego zadania. A mianowicie nauczyłabym się korzystać z biblioteki do Pythona o nazwie Tensorflow[4]. Jednak w dalszym ciągu nie mam pewności, że to przyniosłoby pożądane rezultaty, czyli czy dokładność rozpoznawania znaków na zdjęciach byłaby zdecydowanie wyższa. Bowiem znaków drogowych jest naprawdę wiele, a w mojej bazie zdjęć jest niestety wiele takich, które nie zawierają żadnego znaku.

[4] www.tensorflow.org