Vysoká Škola Báňská – Technická Univerzita Ostrava
Fakulta Elektrotechniky a Informatiky
Katedra Informatiky

Analýza obrazu II

Detekce obsazenosti parkovacích míst

2020

Richard Zvonek, ZVO0016

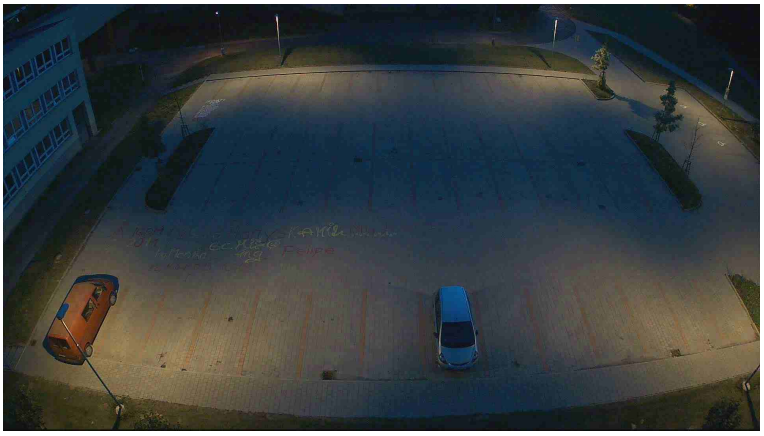
Obsah

1 Zadání

Na zadaných testovacích datech detekovat obsazenost parkovacích míst, vyzkoušet metody s trénováním, bez trénování a srovnat výsledky. Pro srovnání výsledků vypsat počet nesprávně určených (false positive, false negative) výsledků a F1 score.

Testovací sada obsahuje 24 fotografií parkovacích míst, obsahující různě zaplněné parkoviště v různé denní doby.

(a) Testovací obraz



(b) Noční testovací obraz

Obrázek 1: Ukázka testovacích obrazů

2 Detekce bez trénování

Pro detekci bez trénování jsem vyzkoušel několik různých metod rozpoznávání. Kromě nejúspěšnější a nejjednodušší metody pomocí Cannyho detektoru hran (viz 2.1) jsem vyzkoušel i variaci se stejným postupem s jinými detektory hran (Sobelův operátor, Laplace). Canny se osvědčil jako nejlépe fungující, proto jej dále rozvedu. Kromě těchto jednoduchých metod jsem rozpracoval i HAAR a HOG, ale HAAR jsem dále neimplementoval a HOG jsem nakonec nedokončil a zůstal v rozpracovaném stavu.

2.1 Rozpoznávání pomocí detektoru hran

Nejjednodušší, nejméně náročnou a nejrychleji naimplementovanou metodou vůbec se stalo jednoduché rozpoznávání za pomocí detektorů hran. Po testování různých detektorů jsem nakonec použil Cannyho

detektor, pro získání minimální a maximální hodnoty jsem nejprve volil empiricky zvolené metody, později jsem se uchýlil k automatickému výpočtu těchto hodnot pomocí OTSU prahování histogramem. Pro zvýšení úspěšnosti detektoru bylo potřeba vyladit předzpracování obrazu, nejvíce se mi osvědčila metoda rozmazání obrazu mediánem. Vyzkoušel jsem také úpravu jasu lineárně nebo pomocí histogramu, ale tyto metody nevedly ke zvýšení úspěšnosti detektoru.

Po získání obrazu z detektoru je potřeba použít nějakou metriku pro určení, zda je parkovací místo obsazeno nebo ne. V mém případě jsem vyzkoušel dva přístupy. V prvním přístupu jsem hledal horizontální a vertikální hrany pomocí Hough transformation [1], ve druhém přístupu jsem jednoduše spočítal počet bílých a černých pixelů, vypočítal jsem jejich podíl a podle něj jsem rozhodl, jestli je dané místo obsazeno nebo ne. Překvapivě se druhý způsob ukázal jako poměrně přesný, po částečně empirickém upravování hraničních hodnot[2] jsem dosáhnul relativně kompetetivního score (viz tabulka 2). Nakonec jsem se rozhodl použít pouze tento jednoduchý způsob a detekci horizontálních a vertikálních hran jsem zahodil.

3 Detekce s trénováním

Při práci s detektory jsem narazil na potřebu mít modulární zdrojový kód, takže jsem vědoval několik hodin usilovnému refactoringu, až jsem došel k poměrně péknému objektovému, obecnému kódu. Definice nové neuronové sítě je tak práce na pár minut a jen pár řádků kódu. Pro samotné neuronové sítě jsem použil framework dlib, která potřebuje veškerou nutnou funkcionalitu. Při zprovoznění knihovny jsem nejprve neměl žádné problémy, ale celkové trénování a rozpoznávání bylo zoufale pomalé. Tato vlastnost byla způsobena nejen mým ne úplně high end vybavením, ale primárně nepoužíváním CUDA. Po rekompilaci dlib s CUDA mi dlib přestal fungovat úplně z důvodu nesprávně nastavených ovladačů pro CUDA v arch linuxu. Následoval můj pokus o přeinstalaci ovladačů, který ovšem skončil nenaběhnutím systému. Pokračoval jsem tedy na čerstvě reinstalovaném linuxu, na kterém už bylo rozběhnutí dlib s CUDA otázkou chvilky.

Při pokusech s neuronovými sítěmi jsem narazil na některé nedostatky dodané tréninkové sady, které dále rozebírám v kapitole 3.1. Samotné použití neuronových sítí bylo nakonec celkem jednoduché, každou neuronovou síť jsem několikrát přetrénoval, každým tréninkem jsem dosáhl jiné úspěšnosti. Do celkových výsledků počítám nejlepší výsledek, každá nejlépe natrénovaná neuronová síť je pak uložena v přiloženém souboru.

3.1 Trénovací data

Pro trénování neuronových sítí byly dodány trénovací obrazy prázdných a plných parkovacích míst. Nevýhodou dodaných trénovacích dat je absence nočních snímků. Na tento nedostatek neuronové sítě trpěly, proto jsem se rozhodl implementovat umělé rozšíření datové sady. Pro výpočet rozšíření datové sady jsem využil vysoce sofistikovaného postupu, viz výpis 1. Po rozšíření trénovací sady se celkový počet trénovacích dat zosmínásobil, ukázka výsledku rozšíření je zobrazen na obrázcích 4.

Po aplikaci rozšíření se zvýší doba trénování neuronové sítě, zvýší se paměťová náročnost GPU při tréninku, ale celková přesnost se podstatně zvýší (v řádu jednotek procent).

3.2 Použité neuronové sítě

3.2.1 LeNet

LeNet síť jsem implementoval podle příkladu přímo od autorů knihovny dlib. Tato síť má v některých případech problémy s ostrými stíny, případně s nočním, silně zašumným obrazem. Dalším problémem je také nesprávné rozpoznání pouličního osvětlení jako zaplněného parkovacího místa, viz obrázek 2.

LeNet síť se dále vyznačovala rychlým tréninkem.

[1] `https://www.learnopencv.com/hough-transform-with-opencv-c-python/`

[2] Vyexportoval jsem poměry a analyzoval jsem distribuci hodnot

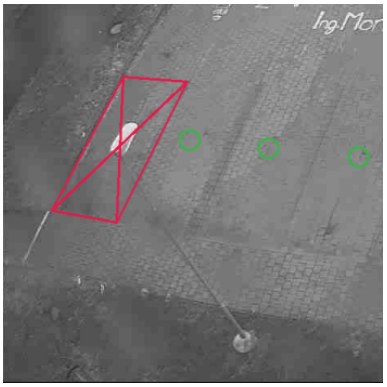
Listing 1 Výpočet rozšíření datové sady

```
void TrainInputSet::getExtendedImages(const cv::Mat &inputImg,
std::vector<cv::Mat> &extendedImages) {
    cv::Mat flippedx;
    cv::Mat flippedy;
    cv::Mat flippedxy;

    cv::flip(inputImg, flippedx, 0);
    cv::flip(inputImg, flippedy, 1);
    cv::flip(inputImg, flippedxy, -1);

    extendedImages.emplace_back(flippedx);
    extendedImages.emplace_back(flippedy);
    extendedImages.emplace_back(flippedxy);

    // Simulate night
    extendedImages.emplace_back(inputImg / 2);
    extendedImages.emplace_back(flippedx / 3);
    extendedImages.emplace_back(flippedy / 4);
    extendedImages.emplace_back(flippedxy / 5);
  }
```



Obrázek 2: Ukázka nesprávně rozpoznané lampy

3.2.2 AlexNet

AlexNet síť jsem implementoval podle příkladu uvedeného na cvičení. Oproti klasické síti AlexNet jsem na vstup nedal obraz o velikosti 227x227, ale použil jsem nezměněný obraz 80x80. K této úpravě jsem se uchýlil po obtížích s tréninkem sítě, kty i při malých dávkách jsem měl problémy s nedostatečnou velikostí paměti a případně s velmi dlouhým tréninkem. Po znížení velikosti vstupního obrazu jsem už s natrénováním neměl problém, přestože se tato síť ze všech vyzkoušených řad k těm s řádově nejvyšší dobou tréninku.

Oproti síti LeNet nemá tato síť problémy s nočními snímky, ale sdílí problém s nesprávným rozpoznáváním lampy (viz viz obrázek 2), případně s parkovacími místy, které jsou opticky zakryta vozidly z vedlejšího parkovacího místa, viz obrázek 3.

3.2.3 VGG19

Síť Vgg19 jsem implementoval podle příkladu uvedeného na cvičení. Oproti předchozím sítím se podstatně déle trénuje a nedosahuje lepších výsledků. Také je velmi hladová na systémové prostředky, i při rozlišení vstupu 32x32 jsem měl problémy s nedostatkem paměti. Předpokládám, že při hlubším zkoumání bych

Obrázek 3: Ukázka opticky překrytého parkovacího místa

byl určitě schopen tuto síť odladit, ale vzhledem k dlouhým trénovacím časům jsem ji dále netestoval. Síť VGG19 v mnou natrénovaném stavu má podobné nedostatky, jako síť LeNet.

3.2.4 ResNet

Síť ResNet jsem implementoval podle příkladu na stránkách dlib. Při prvním spuštění jsem byl překvapen, protože při trénovacím čase srovnatelným s LeNet dosahovala tato síť drasticky lepších výkonů. Tato síť je schopna spolehlivě rozpoznávat i na částečně překrytých místech a eliminuje i některé detekce lamp. Ne však všechny.

3.2.5 GoogLeNet

Síť GoogLeNet oproti ResNet překvapuje ještě lepším časem tréninku se srovnatelným výkonem při rozpoznávání. Problémem je opět lampa a v některých případech i částečné překrytí parkovacího místa.

4 Kombinované rozpoznávání

Pro zvýšení přesnosti jsem implementoval kombinace jednotlivých rozpoznávacích metod. Až na kombinaci všech metod ale nebyly kombinace o tolik lepší, než samotné nejlepší neuronové sítě, proto se o nich zmíním jen krátce.

4.1 Kombinace AlexNet, ResNet a GoogLeNet

Vzhledem k tomu, že v některých případech měly kombinované sítě nedostatky v různých oblastech, očekával jsem, že se tyto nedostatky vyeliminují. To se ovšem úplně nestalo, kombinace těchto tří sítí není lepší, než nejlepší z nich.

4.2 Kombinace všech rozpoznávacích metod

Jednou z nejlepších kombinací je kombinace všech metod rozpoznávání. Rozpoznání se přijme, pokud se na výsledku shodne 4 z 5 sítí. Tato kombinace dosahuje úplně nejlepších výsledků, ale také je časově nejnáročnější.

5 Závěr

Naimplementoval jsem různé metody pro rozpoznávání obsazenosti parkoviště, výsledné srovnání je přiloženo v tabulce 1 a 2. Zdrojové kódy jsou k dispozici na `https://github.com/SacrificeGhuleh/ANOII2020/tree/ANOII2020` (je třeba rekurzivní git clone z důvodu použití submodulu), pro build je potřeba cmake. Využívám některé funkce z C++17, je tedy potřeba mít kompatibilní kompilátor. Testoval jsem funkčnost na gcc v10.2.0. Na `https://github.com/SacrificeGhuleh/ANOII2020/releases/tag/ANOII2020` je také možné stáhnout výsledky jednotlivých metod rozpoznávání v obrazové formě a binární soubory natrénovaných sítí (soubor ANOII.zip).

Co se týče samotných výsledků, nejlepší výsledek dává rozpznávání pomocí kombinace všech metod. Tato úspěšnost je však vykoupena mnohem vyšším časem, který je potřeba pro rozpoznání. Mými osobními favority se tak stávají sítě ResNet a GoogLeNet, které dosahují skvělých výsledků s nízkými časy trénování i rozpoznávání.

Přílohy

Srovnání neuronových sítí

DNN	Velikost obrazu	Learning Rate	Min Learning Rate	Batch size	Max steps without progress	Max epochs	Doba tréninku[3]
LeNet	28×28	0.01	10^{-6}	128	1000	300	$\approx 45s$
AlexNet	80×80	0.01	0.001	256	1000	300	$800s \sim 1000s$
Vgg19	32×32	0.01	10^{-7}	64	500	300	$\approx 3000s$
ResNet	32×32	0.01	10^{-7}	64	128	300	$\approx 40s$
GoogLeNet	32×32	0.01	10^{-7}	64	128	300	$\approx 35s$

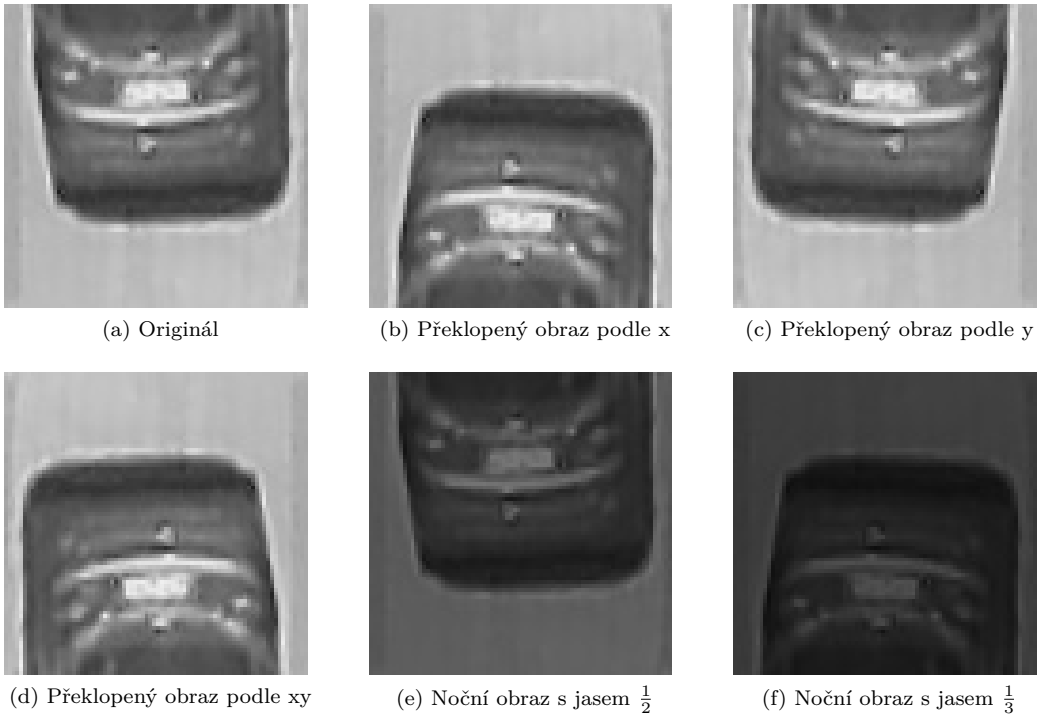
Tabulka 1: Nastavení neuronových sítí

Metoda	False positives	False negatives	Přesnost rozpoznání	F1 score
Canny	91	0	0.932	0.965
LeNet	25	0	0.981	0.990
AlexNet	12	0	0.991	0.996
Vgg19	26	0	0.981	0.990
ResNet	11	0	0.992	0.996
GoogLeNet	10	1	0.992	0.994
Kombinace: AlexNet ResNet GoogLeNet	10	1	0.993	0.996
Kombinace: všechny metody $\min\frac{4}{5}$	5	0	0.996	0.998

Tabulka 2: Výsledky různých typů rozpoznávání

[3]Trénování a rozpoznávání bylo testováno na Manjaro Linux x86_64 5.8.18, Intel i5-5200U@2.7GHz, NVIDIA GeForce GTX 850M, 4GB VRAM

Rozšířená trénovací sada



(a) Originál (b) Překlopený obraz podle x (c) Překlopený obraz podle y

(d) Překlopený obraz podle xy (e) Noční obraz s jasem $\frac{1}{2}$ (f) Noční obraz s jasem $\frac{1}{3}$

(g) Noční obraz s jasem $\frac{1}{4}$ (h) Noční obraz s jasem $\frac{1}{5}$

Obrázek 4: Ukázka rozšířené trénovací sady