ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra výkonové elektroniky a strojů

BAKALÁŘSKÁ/DIPLOMOVÁ PRÁCE

Systém pro automatickou modifikaci videa v reálném čase s prvky AI(umělé inteligence)

Jan Beneš **2021**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan BENEŠ**
Osobní číslo:	**E18B0113P**
Studijní program:	**B2644 Aplikovaná elektrotechnika**
Studijní obor:	**Aplikovaná elektrotechnika**
Téma práce:	**Systém pro automatickou modifikaci videa v reálném čase s prvky AI (umělé inteligence)**
Zadávající katedra:	**Katedra výkonové elektroniky a strojů**

Zásady pro vypracování

1. Prostudujte principy a algoritmy neuronových sítí a hlubokého učení se zaměřením na zpracování obrazu.
2. Prostudujte principy zpracování obrazu a vyberte vhodné knihovny.
3. Navrhněte učící se systém určený k filtrování objektů v obraze v reálném čase (rozmazání obličejů, rozpoznání a překrytí nevhodných objektů ve videu).
4. Implementujte řešení na zvolené vývojové platformě.
5. Implementujte řešení na zvoleném mikrokontroleru s podporou neuronových sítí a kamery.

Rozsah bakalářské práce: **30 – 40 stran**
Rozsah grafických prací: **podle doporučení vedoucího**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Chollet, F. Deep learning v jazyku Python: knihovny Keras, Tensorflow. Přeložil Rudolf PECINOVSKÝ. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.
2. Cuesta, H. Analýza dat v praxi. Přeložil Jiří HUF. Brno: Computer Press, 2015. ISBN 978-80-2514361-2.
3. Dendaluce, M., Valera, J., Gómez, V., Irigoyen, E., Larzabal, E. (2014). Microcontroller Implementation of a Multi Objective Genetic Algorithm for Real-Time Intelligent Control. 10.1007/978-3-319-01854-6_8.
4. Understanding and Visualizing Neural Networks in Python. Analytics Vidhya – Learn Machine learning, artificial intelligence, business analytics, data science, big data, data visualizations tools and techniques. Analytics Vidhya [online]. Copyright 2013 [cit. 30.04.2020]. Dostupné z: https://www.analyticsvidhya.com/blog/2019/05/understanding-visualizing-neural-networks/.
5. Kim, T. S., Bae, J., Sunwoo, M. H. Fast Convolution Algorithm for Convolutional Neural Networks, 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Hsinchu, Taiwan, 2019, pp. 258-261.

Vedoucí bakalářské práce: **Ing. Petr Kropík, Ph.D.**
Katedra elektrotechniky a počítačového modelování

Datum zadání bakalářské práce: **9. října 2020**
Termín odevzdání bakalářské práce: **27. května 2021**

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Prof. Ing. Václav Kůs, CSc.
vedoucí katedry

Abstrakt

Předkládaná bakalářská práce se zabývá detekcí lidských obličejů v reálném čase. Zaměřuje se na reálné použití v reálném čase. Rozpoznávací software běží na mikropočítači Raspberry PI 3 model B, ke kterému je připojena Raspberry kamera V2.1. Práce je rozdělena na tři části. V první části této práci jsou objasněny používané termíny a teorie. Tyto teoretické poznatky jsou dále potřeba v části praktické a analytické. Praktická část se zabývá výběrem a použitím správné detekční metody, dále se jedná o výběr vhodné vývojové platformy, mikro(kontroléru/počítače),potřebných periferií a vhodnou volbou a úpravu datasetu. Analytická část se zabývá samotným kódem a problematikou detekce.

Klíčová slova

Konvoluční neuronová síť, haar features, cascade, zpracování obrazu, umělá inteligence, detekce obrazu, poznání obrazu.

Abstract

The presented bachelor thesis deals with the detection of human faces in real time. It focuses on real-time use. The recognition software runs on a microcomputer Raspberry PI 3 Model B, to which a Raspberry V2.1 camera is connected. The work is divided into three parts. The first part of this work clarifies the terms and theories used. This theoretical knowledge is further needed in the practical and analytical part. The practical part deals with the selection and use of the correct detection method, selection of a suitable development platform, micro (controller / computer), the necessary peripherals and the appropriate choice and modification of the dataset. The analytical part deals with the code itself and the issue of detection.

Key words

Convolutional neural network, haar features, cascade, image processing, artificial intelligence, image detection, image recognition

Prohlášení

Prohlašuji, že jsem tuto diplomovou/bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské/diplomové práce, je legální.

V Písku dne 26.5.2021 Jan Beneš

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce Ing. Petru Kropíkovi, Ph.D. za drahocenný čas, konzultace, poskytnutý hardware a vzdálený přístup počítači a clusteru.

Obsah

ÚVOD **9**

SEZNAM ZKRATEK A POJMŮ **1**

1 UMĚLÁ INTELIGENCE **2**

2 DRUHY ALGORITMŮ PRO DETEKCI OBRAZU **4**

2.1 KONVOLUČNÍ NEURONOVÉ SÍTĚ 4
2.2 HAAR CLASSIFIER 9

3 VÝBĚR VÝVOJOVÉHO PROSTŘEDÍ A ÚPRAVA DATASETU **14**

4 MASK R-CNN **18**

5 HAAR LIKE FEATURES **28**

6 INTEGRACE DO RASPBERRY PI 3 B **33**

7 ZÁVĚR **37**

SEZNAM LITERATURY A INFORMAČNÍCH ZDROJŮ **38**

SEZNAM OBRÁZKŮ **41**

SEZNAM TABULEK **42**

PŘÍLOHA A. **43**

PŘÍLOHA B. **43**

PŘÍLOHA C. **43**

Úvod

Hlavním důvodem, proč jsem si vybral právě počítačové vidění je ten, že jsem se chtěl věnovat oboru vědy, který má zářnou budoucnost a je neustále ve vývoji. Rád bych přispěl a byl ve středu dění při vzniku nové éry umělé inteligence. Původně jsem žertoval, že mi vadí na streamovacích platformách nahodilá nahota, a že bych měl nejraději filtr, kterým bych vše překryl. Když jsem si na tuto připomínku vzpomněl později, tak mi vrtalo hlavou jak by to vůbec šlo udělat, tak jsem našel počítačové vidění, ale neměl jsem naprosté tušení, jak co funguje ani o čem se komunita na internetu baví. Brzo jsem se začal více o toto téma zajímat a zjistil jsem, že můj předchozí žert není možné uskutečnit. Od té doby co jsem se ale začal o tuto tématiku zajímat, mi v hlavě utkvělo několik nápadu jak využít počítačové vidění a svoje nápady se pokusit dostat na trh. Někde se ale musí začít, a proto jsem si vybral jednu z nejzákladnějších úkonů pro počítačové vidění, a to detekci obličejů. Jako každý programátor, který začínal se svým prvním „Hello World“ i já jsem musel někde začít. Jelikož jsem začínal téměř od nuly, bez žádné znalosti o AI a lehkou znalostí o programování, bylo pro mne těžké hledat i samotné zdroje učení a i když dnes na internetu najdete spoustu užitečných informací, tak za opravdové znalosti se platí zlatem. Proto další důvod proč tuto práci píšu je seznámit čtenáře alespoň se základy počítačového vidění, aby věděl odkud alespoň začít. Já sám vím, že bych za něco takového byl dříve vděčný.

Dnes je počítačové vidění všude okolo nás ať už se jedná o zabezpečovací systémy či například o autonomní systémy jakou jsou v elektromobilech. Počítačové vidění je už starší záležitost, která se začala drát napovrch na přelomu tisíciletí. Ročně se přichází s novými algoritmy a upgrady starších algoritmů a já se nemohu dočkat dne, kdy AI bude schopna rozpoznávat obraz stejně jako člověk ne-li lépe.

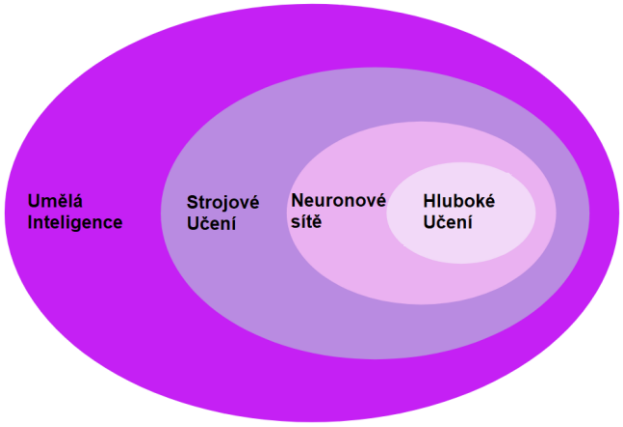
Cílem této práce je seznámit čtenáře se základy počítačového vidění. Práce se zabývá jak teorií a rozdělením AI tak samotnou implementací a učením sítě na reálných datech. Jedná se o real time rozpoznání lidských obličejů a následnou úpravu obrazu, já si vybral cenzuru. Vše je pak zapouzdřeno v mikropočítači Raspberry Pi, který je možné přenášet, projekt by šel rozšířit například o možnosti změny rozlišení, nebo o stojan/pouzdro na kameru a mikropočítač.

Seznam zkratek a pojmů

Pycharm ……. Python vývojové prostředí
CNN…… Konvoluční neuronové sítě
RCNN……. Regionální konvoluční neuronové sítě
Haar-like features…… Haar funkce
BND box …… Ohraničující úhelník
ReLu…… Usměrňovací lineární jednotka
False positive……... Označení špatného obrazce
False negative…….. Neoznačení správného obrazce
Overfit……. Přeučení modelu
Underfit……... Model není dostatečně neučen
Deep Learning……. Hluboké učení
Grayscale……. Černo-bílá o různé intenzitě(1 vrstvá)
Label…… Označení
MNIST……. Data set obsahující ručně psaná čísla
Kernel……... 3x3 nebo 5x5 matice
Feature map…….. Mapa funkcí
Padding……. Vycpávky matice
Stride…… Kroky kernelu
Pooling operation……. Sdružování
Regression function….. Regresní ztrátová funkce
Binary classification… Binární klasifikace
RMSprop…….. RootMeanSquare probability
OpenCV……..... Knihovna pro manipulaci s obrazem
Tensorflow…… Platforma pro strojové učení
Class……. Jeden z mnoha objektů například člověk, obličej či jablko
GAN……. Generative Adversarial Network
Cropped images…… Ořízlé obrázky
mAP…….. mean Average Precision – střední hodnota přesnosti

1 umělá inteligence

Nejdříve se musí definovat, co se vlastně myslí, když se řekne pojem AI. Co je AI, strojové a hluboké učení a jaký mají mezi sebou vztah?



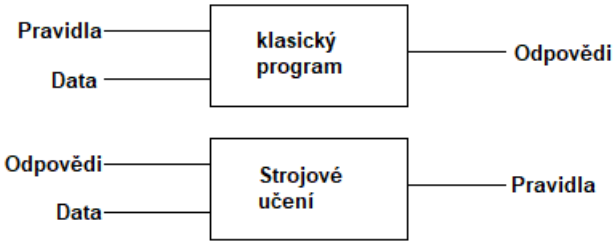
Obrázek 1: Podmnožiny umělé inteligence

Umělá inteligence

AI je obecná oblast zahrnující nejen strojové a hluboké učení, ale také další množství přístupů, které nezahrnují žádné učení. Například první AI se řídily předepsanými pravidly, nešlo tedy o strojové učení. Dlouhou dobu se používala tzv. symbolická AI, která byla definována rozsáhlými explicitními pravidly. Pro řešení složitějších nejasných problémů, jako je rozpoznání obrazu se přestala používat symbolická AI a nahradilo jí strojové učení.

Strojové učení

Strojové učení se zabývá otázkou : byl by počítač schopen samostatně si vytvořit pravidla, podle kterých by se řídil, aniž by mu tyto pravidla byly poskytnuty? Strojově učený systém je spíše natrénován, než naprogramován, jsou mu předány data a odpovědi a on sám si pro budoucí příklady vytvoří pravidla.



Obrázek 2:Strojové učení proti klasickému přístupu

Hluboké učení

Hluboké učení je podmnožinou strojového učení. Je to nový přístup k učení se

z reprezentace dat, který klade důraz na učení se pomocí vrstev. Výraz hloubka tedy odpovídá myšlence, že učení se dělá postupně ve vrstvách. Počet vrstev v modelu se tedy nazývá hloubka modelu. Moderní hluboké učení se mnohdy skládá z desítek až stovek vrstev. V hlubokém učení jsou tyto vrstvené reprezentace, téměř vždy, naučeny prostřednictvím modelů nazývaných neuronové sítě, strukturované ve vrstvách na sobě.

2 Druhy algoritmů pro detekci obrazu

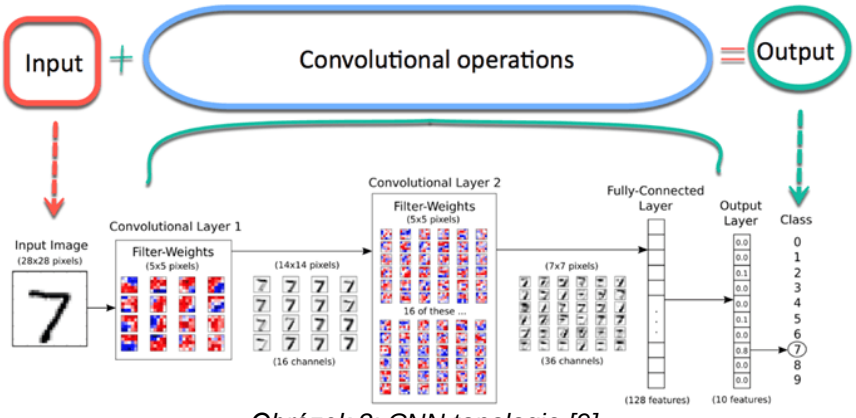
Existuje několik druhů pro detekci objektů v obraze, kde některé jsou si podobné fungují například na principu neuronových sítí, a jiné fungují například na matematickém popisu objektu.

- Fast R-CNN.
- Faster R-CNN.
- Histogram of Oriented Gradients (HOG)
- Region-based Convolutional Neural Networks (R-CNN)
- Region-based Fully Convolutional Network (R-FCN)
- Single Shot Detector (SSD)
- Spatial Pyramid Pooling (SPP-net)
- YOLO (You Only Look Once)
- OpenCV – Haar classifier

Já jsem si vybral dva přístupy, R-CNN a Haar classifier. R-CNN jsem si vybral z důvodu, že jsem se chtěl naučit konvoluční neuronové sítě a je to skvělý začátek do počítačového vidění plus, CNN má budoucnost. Po natrénování CNN jsem zjistil, že není vhodný na živou úpravu dat, pokud není podporován výkonným CPU. Na Haar přístup jsem přešel abych mohl integrovat model do Raspberry Pi.

2.1 Konvoluční neuronové sítě

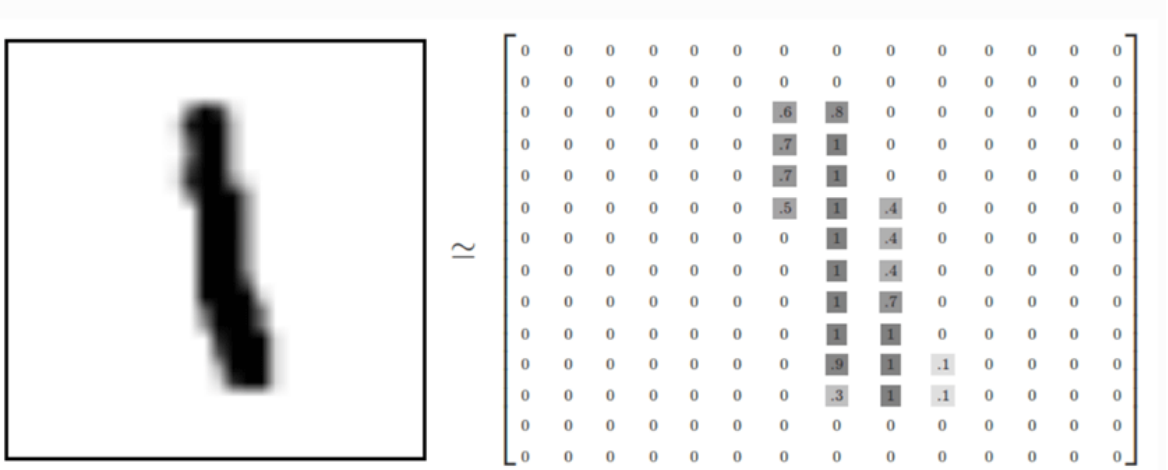
Konvoluční neuronová síť není obtížná na pochopení, obrázek je zpracován v konvoluční fázi a poté je mu přiřazen label.



Obrázek 3: CNN topologie [9]

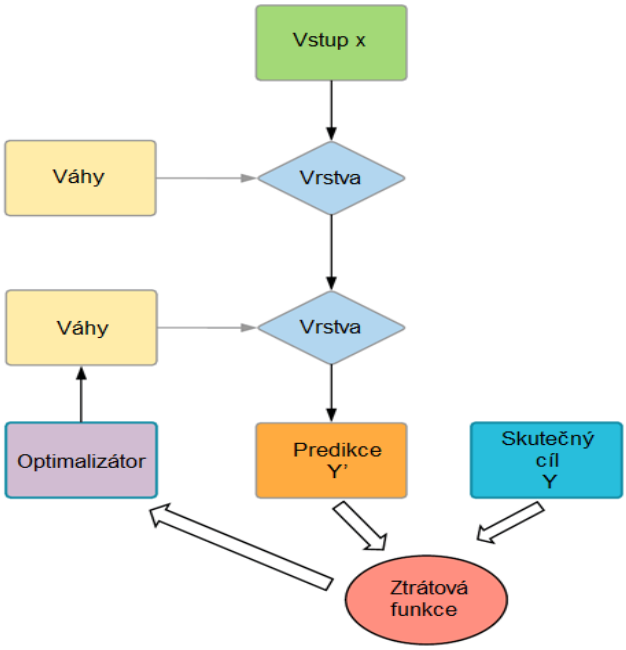
obrázek se skládá řádků a sloupců a vytváří dvojrozměrné pole, matici. Grayscale obrázek má pouze jednu vrstvu(hloubku) a barevná mají tři vrstvy, za každou složku RGB. Tyto vrstvy jsou pak na sebe naskládané. Každý pixel pole má přiřazenou hodnotu od 0 do 255,

tato hodnota určuje intenzitu barvy, např. hodnota 0 může signalizovat bílou a hodnota 255 tmavší barvu. Pro jednoduchost se dají tyto hodnoty převést na rozmezí 0 až 1, 0 bílá a 1 černá.



Obrázek 4: MNIST číslo 1 [10]

Vrstvy



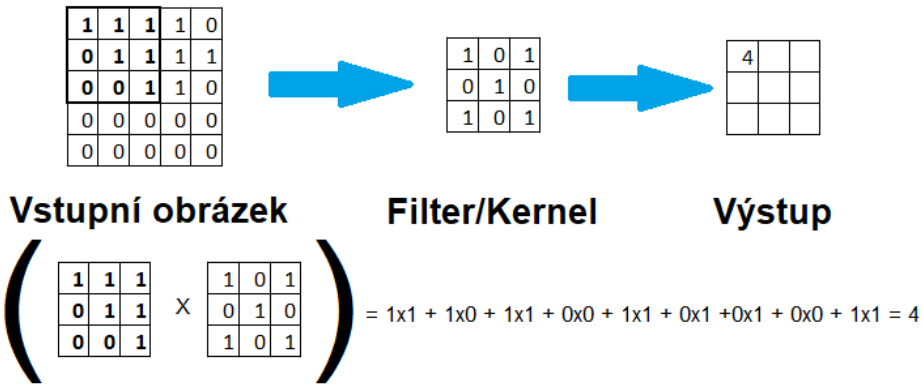
Obrázek 5: Vrstvy CNN

Parametry vrstev jsou váhy, učení tedy znamená najít správné hodnoty vah. Nalezení správné hodnoty není jednoduché, síť může obsahovat miliony parametrů. Chceme-li tedy správně nastavit váhy vrstev, potřebujeme sledovat výstup, k tomu slouží ztrátová funkce. Tato funkce vrací ztrátové skóre, což je rozdíl mezi predikcí a skutečností. Základním trikem hlubokého učení, je použít toto zpětnovazební skóre a upravit hodnotu vah, tak aby se skóre snížilo. Toto nastavení vah, má za úkol optimalizátor. Z počátku jsou vahám přiděleny náhodné hodnoty a až po určitém počtu iterací se ztrátové skóre sníží. Při dostatečně nízkém skóre se síť bere za natrénovanou.

Konvoluce

Nejdůležitější částí modelu je konvoluční vrstva. Tato vrstva slouží k redukci velikosti obrázku, z čehož vyplývá zrychlení výpočtu, tudíž celého modelu. Účel konvoluce je

extrahovat vlastnosti objektu z obrázku. Konvoluce je druh elementárního násobení. Počítač projíždí obrázek filtrem o velikosti 3x3 nebo 5x5, taktéž nazýván kernel, poté je provedena konvoluce. Výstup konvoluce je nazýván feature map. Je třeba poukázat na to, že velikost mapy funkcí se po konvoluci zmenší.



Obrázek 6: Konvoluce

Kernel

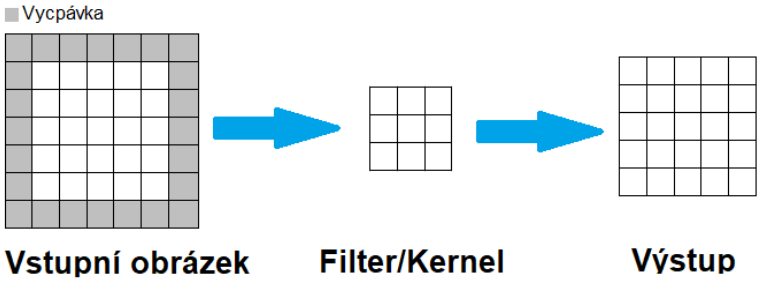
Existuje nespočetné množství filtrů, každý z filtrů plní svojí vlastní jedinečnou funkci. Kernelu se taky říká váha.

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Obrázek 7: Druhy kernelů [11]

Padding

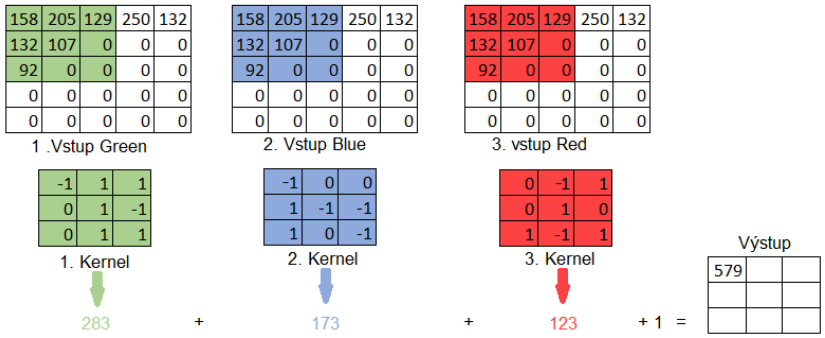
V Podkapitole konvoluce byla zmínka, že po konvoluci se výstupní matice zmenší, abychom tomuto předešli používá se vycpávka(padding). Kernel se posouvá s určitým krokem(stride), je to o kolik se posune při průchodu maticí, většinou se používá krok rovno jedné. Krok i velikost kernelu ovlivňuje výslednou velikost výstupu, čím větší velikost a krok, o to více se výstup zmenší, abychom zachovali parametry 1:1(vstup : výstup), přidáme vycpávky na vstup.



Obrázek 8: Padding

Hloubka

Hloubka definuje počet filtrů, které se použijí během konvoluce, v podkapitole o konvoluci byl použit jeden filtr. Po každé konvoluci je výstupu přiřazen Bias, což je hodnota rovna 1.



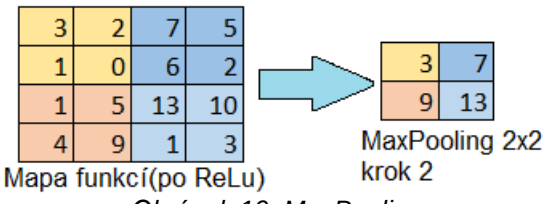
Obrázek 9: RGB kernel

ReLu

Na konci konvoluční operace je výstup podroben aktivační funkci, aby se zajistila nelinearitu. Nejběžnější aktivační funkce je funkce ReLu. Všechny negativní pixely budou převedeny na nulu.

Sdružování

Účel sdružování je jednoduchý, jedná se o snižování dimenze matice. Sdružování je důležité pro snížení výpočetní doby a potřebného výkonu. Při snížení dimenze má síť menší počet vah a zabraňuje se přeučení. Standartní postup, jak sdružovat je MaxPooling, kde např. vstupní matice(mapa funkcí) 4x4 je sdružovaná 2x2 s krokem 2. Dalším druhem sdružovaní je MeanPooling, střední sdružování, kde se vezme střední hodnota.



Obrázek 10: MaxPooling

Plné propojení vrstev

Znamená, že všechny vrstvy(neurony) jsou připojené na výstup.

Ztrátová funkce

Tato funkce porovnává, jak přesně náš model odhaduje realitu od predikce. Vrací ztrátové skóre, čím vyšší tím horší predikce.

Druhy ztrátových funkcí:

- Regresní ztrátová funkce:

  Regresní modely se zabývají predikcí spojité hodnoty. Například, pokud známe místo, kde se nachází obytný dům, jeho počet místností a obytnou plochu, jsme schopni předpovědět cenu. Dalo by se říct, že předvídá množství nežli label.

- Binární klasifikační funkce:

  Binární funkce je predikční algoritmus, kde výstup může jeden ze dvou možných, tedy 0 nebo 1. Výstupem této funkce, není čistě 0 nebo 1, ale dalo by se říct, že se jedná o velikost jistoty, hodnota se totiž může pohybovat mezi 0 až 1. Při hodnotě <0.5 je vyhodnocena 0 a naopak.

- Vícetřídní binární funkce

  Jedná se pouze o rozšířenou verzi klasické binární funkce. Vícetřídní má pouze vícero tříd nežli jen dvě jako klasická binární.

Optimalizátor

Podle ztrátového skóre upravuje váhy vrstev.

Druhy optimalizátorů:

- Gradientní sestup:

  Optimalizační algoritmus, který se snaží najít lokální minimum funkce, tedy nastavení vah tak, aby docházelo k minimálnímu ztrátovému skóre. Myšlenkou jsou opakované kroky v opačném směru gradientu funkce v aktuálním bodě, protože tím zajistíme směr nejstrmějšího klesání.

- Gradientní optimalizovaný sestup:

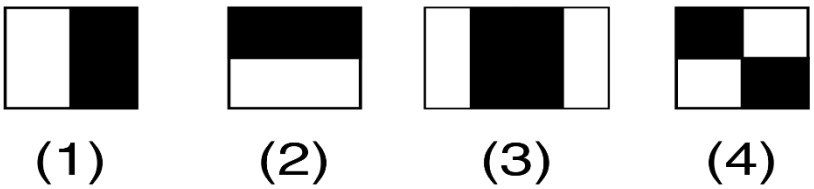
  Jelikož klasický gradientní sestup může vést k velikým oscilacím, začali se používat optimalizované gradientní sestupy. Nejznámějším je RMSprop, který má nastavitelnou rychlost učení(learning rate), zjednodušeně řečeno, RMSprop zrychluje čas na ose x a tím pokládá celou funkci a snižuje kmitání.

2.2 Haar classifier

Haar klasifikátor je OpenCV algoritmus. Dělá klasifikaci mezi obrázkem, co obsahuje objekt a obrázkem co objekt neobsahuje. Postup je velmi podobný CNN v tom, že vstupní obrázek je postupně skenován filtrem o určité velikosti, který sleduje, jestli zrovna tato oblast obsahuje hledané features či ne.

Haar features

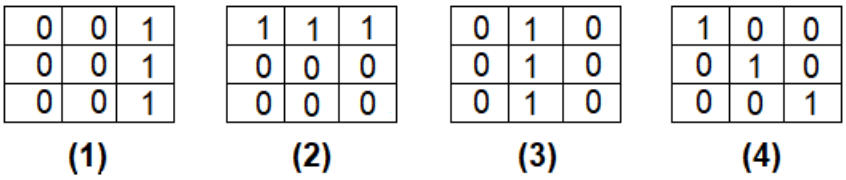
Haar feature je jako Kernel v CNN až na to že, v CNN jsou hodnoty Kernelu trénované, kdyžto haar features jsou manuálně nastaveny. Neznamená to, že vlastnosti nastavujeme skutečně ručně. Jelikož features jsou to samé, co kernel, také se občas tak označují. Najdou se také pod pojmem rectangle features, podle počtu sektorů se k nim přidává číselná přípona.



Obrázek 11: Druhy features [12]

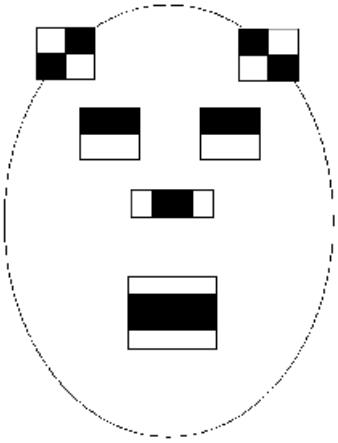
(1) Edge feature – Two-rectangle feature
(2) Edge feature – Two-rectangle feature
(3) Line feature – Three-rectangle feature
(4) Four-rectangle feature

Číselně vyjádřeny by mohli vypadat například takto.



Obrázek 12: Číselně vyjádřen feature

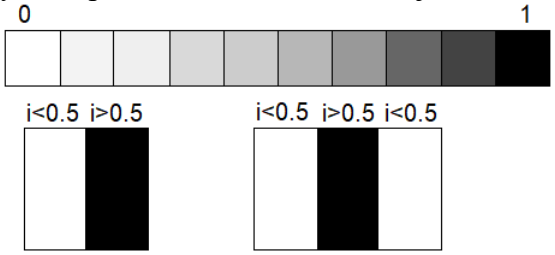
Aplikované features na lidském obličeji, by mohli vypadat takto



Obrázek 13: Zjednodušený model obličeje

Trénování features

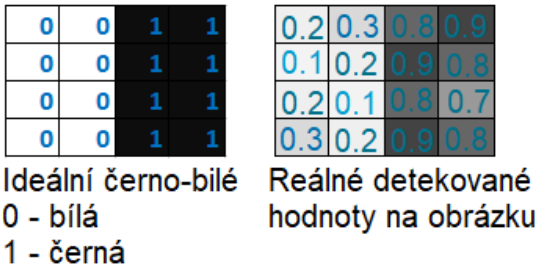
Pro trénování je potřeba mít dva druhy datasetu, první dataset(positive) by měl obsahovat pouze náš hledaný objekt, zatímco druhý dataset(negative) obsahuje vše kromě našeho hledaného objektu. Positive dataset je nutné před trénováním převést do grayscale. Grayscale přiřadí každému pixelu intenzitu tmavosti, 0 bílá 1 černá. Poté algoritmus přiřadí každé oblasti features, například nos (neskládá se pouze z jedné feature ale z mnoha). Negativní dataset slouží k tomu, aby se nepřiřazovali features i k objektům, které nechceme.



Obrázek 14: Intenzita pixelů

Detekce features

Algoritmus porovnává, jak moc blízko jsme od ideální vs klasické feature. Je potřeba spočítat střední hodnotu světlých pixelů a poté střední hodnotu tmavých pixelů. Jejich rozdíl nám poví, jestli jsme našli feature, ideálně by tento rozdíl měl být roven 1.



Obrázek 15: Ideální vs skutečná feature

Z obrázku 14, lze spočítat přesnost feature následujícím způsobem.

1. Spočítat střední hodnotu světlých pixelů.
2. Spočítat střední hodnotu tmavých pixelů.
3. Spočítat jejich rozdíl.

Pro ideální:

$$\Delta = tmavé - světlé = \frac{1}{n}\sum_{tmavé}^{n} I(x) - \frac{1}{n}\sum_{světlé}^{n} I(x)$$

$$\Delta = tmavé - světlé = 1 - 0 = 1$$

pro reálné:

$$\Delta = tmavé - světlé = \frac{1}{n}\sum_{tmavé}^{n} I(x) - \frac{1}{n}\sum_{světlé}^{n} I(x)$$

$$= \frac{4 * 0.8 + 3 * 0.9 + 0.7}{8} - \frac{4 * 0.2 + 2 * 0.3 + 2 * 0.1}{8}$$

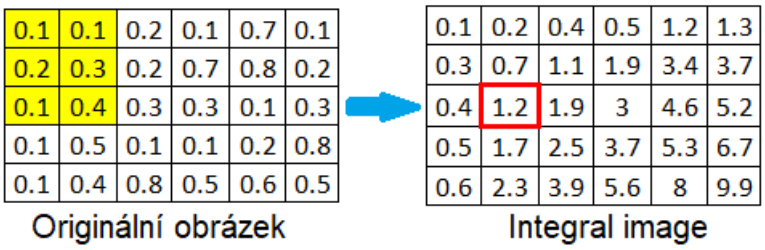
$$= 0.825 - 0.2 = 0.625$$

Hledání střední hodnoty v každé oblasti by zabíralo velké množství času i výpočetního výkonu, proto se zavádí nová reprezentace dat, Integral image.

Integral image

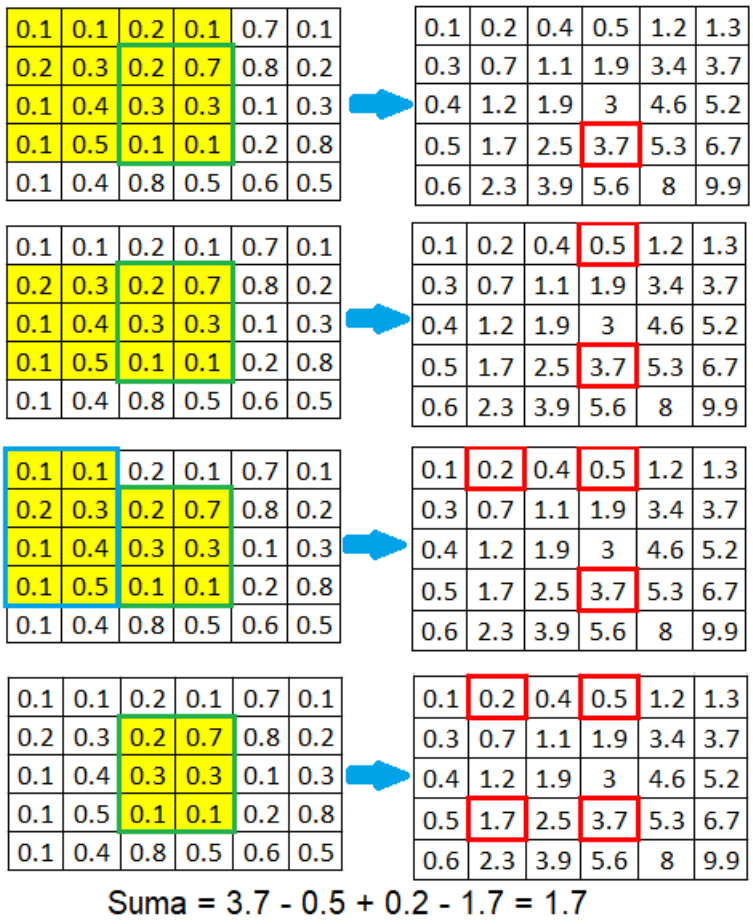
Problém nastává, že musíme spočítat průměr jedné oblasti několikrát, dostáváme se na kvadratickou funkci času, kdy čím větší plocha, tím více roste čas s kvadrátem. Integral image zařídí konstantní běžící čas. Proč se jedná o tolik matematických operací? Protože potřebujeme použít haar features s různými velikostmi na všech možných místech ≈ 200k features ke kalkulaci. Převod originálního obrázku na integral image probíhá tak, že se

vezmou všechny kombinace obdélníků a sečte se jejich hodnota, která se zapíše do dolního pravého rohu integral image.



Obrázek 16: Integral image

Důvod, proč je integral image rychlejší je, že není třeba vždy přepočítávat sumu, originální obrázek je substituován integral image, a poté nám už jen stačí rohové body, jak je ukázáno na obrázku 17.



Obrázek 17: : Postup detekce integral image

Zeleně je vyznačená oblast na původním obrázku.

1. Prvním krokem je vzít si sumu žlutě označeného obdélníka, 3.7 .

2. Je potřeba odečíst sumu nepotřebné části obdélníka, 0.5 .

3. Je třeba přičíst sumu 0.1+0.1=0.2 abychom dostali sumu modrého obdélníka 1.7 .

4. Nakonec modrého obdélníka odečteme a dostaneme sumu zeleného obdélníka 1.7, kdybychom počítali ručně, tak nám suma vyjde 1.7 též.

AdaBoost

Haar features se tedy snaží najít například nos, ústa či oči a další. Originálně ale nebyli features ničím omezeny a bylo jich přibližně ≈180k, později byly sníženy na ≈6000. Většina z těchto features nebudou fungovat správně nebo budou irelevantní pro rozpoznání hledaného objektu, protože budou příliš náhodné na to, aby něco našli. Proto je potřeba udělat výběr z velké množiny features, vyberou se pouze ti, které lépe fungují a eliminují se irelevantní. Abychom dosáhli tohoto cíle, používá se technika AdaBoost, ve které každá z těchto 180k features byla použita na obrázek separovaně a vytvořily se takzvaní weak learners. Někteří weak learners oddělovali positive obrázky od negative obrázků dobře, někteří ne. Weak learners jsou navrženi tak, aby nesprávně určovali pouze minimální počet obrázků. Jejich výsledky mohou být lepší než jen hrubý odhad. Nakonec těchto ≈180k sloučí do ≈6000 a vytvoří takzvané strong learners.

Attentional Cascade

Pro zrychlení detekce byl navržen přístup Attention Cascade. Myšlenka za tímto je taková, že ne všechny features musíme kontrolovat najednou v každém okénku. Pokud první feature není v okénku, tak není třeba zkoušet další a hledaný objekt v té oblasti není.

Postup:

- Features jsou aplikovány postupně v etapách. První etapa bude obsahovat jednoduché features v porovnání s více komplexníma v dalších etapách, které už hledají každý detail objektu. Pokud první etapa selže, zahodí okno na kterém je a přejde na další. Tímhle způsobem se ušetří spoustu výpočetního výkonu a zahozená okna se nebudou znova projíždět.
- Druhá etapa začne pouze pokud okno prošlo první etapou, tento proces pokračuje dále, jestli okno projde druhou etapou, je posláno dál, jinak je zahozeno a tak dále.

Běžný algoritmus obsahuje okolo ≈38 etap a ≈6000 features. V prvních pěti etapách může být například 1,10,25,25 a 50 features a dále se zvyšují. Počáteční etapa odstranila okna, která neobsahovala žádné rysy objektu, čímž se sníží počet false negative a pozdější etapy se mohou zaměřit na snížení false positive.

3 Výběr vývojového prostředí a úprava Datasetu

Než se začne vůbec s návrhem a trénování modelu, je potřeba si zjistit a zajistit co vše je potřeba pro realizaci hlubokého učení.

- Vybrat vhodný programovací jazyk
- CNN vs Haar
- Dataset
- Vhodné anotační prostředí
- Vhodné knihovny pro zpracování obrazu
- Potřebný hardware

Vývojové prostředí

Jedním z nejdůležitějších kroků je vybrat si správné vývojové prostředí, natrénovat model jde téměř ve všech světově známých programovacích jazycích. Ne ale všechny jazyky podporují snadnou implementaci hlubokého učení.
Mezi nejznámější patří:

- Python
- Java
- C++

Se všemi zmíněními jazyky jsem obeznámen, ale python je jazyk, který je asi nejvíce vstřícný k uživateli a nemá tak těžkou učící se křivku. Zároveň Python je také nejpoužívanějším jazykem pro vývoj hlubokého učení a to především z důvodu dostupnosti kvalitních knihoven. Spousty těchto knihoven lze nalézt i v C++ a Javě, například Tensorflow, který je i na platformě Java a C++. Tensorflow je dokonce napsán v jazyce C++ ale vývojáři si vybrali jako hlavní platformu Python. Já si zvolil Python z důvodu nekomplexnosti a přehlednému IDE, Pycharm.

Dataset

Před výběrem frameworku je potřeba sehnat dataset objektu na který, se náš model bude učit. Nejdříve se musí určit jaký objekt neboli classu chceme detekovat. Pro tento projekt

jsem vybral lidské obličeje, což bývá jedna ze základních class, co se detekuje. Ne každému frameworku stačí stejný dataset jako jinému. Jak CNN tak Haar potřebují speciálně upravené datasety pro správnou funkci. Pokud chceme, přímo najít a přesně určit souřadnice objektu a nejenom, že se tam objekt vyskytuje, je potřeba dataset správně anotovat a přiřadit vhodný label. Další důležitá vlastnost datasetu je jeho kvalita a kvantita, například pro detekci obličeje, pokud bychom chtěli, aby byl model schopný rozpoznat jakýkoliv obličej musíme mít různorodá data. Potřebovali bychom všechny druhy národností a ras, muže a ženy, děti, dospělé i staré, zdravé i znetvořené. Je proto potřeba mít dataset, který obsahuje co největší škálu obličejů. Existuje několik technik, jak vylepšit náš dataset, nejpoužívanější jsou ale tyto tři na obrázku 17.

Obrázek 18: Druhy získávání dat [13]

Získávání dat navíc není vždy možné a může být vskutku drahé. Syntetické data vytvořené například GAN sítí se zdají slibné ale komplikované a mohou se lišit od očekávání. Syntetická data jsou vhodná například pokud chceme síť naučit rozpoznávat synteticky stvořené objekty. Na druhou stranu, augmentace dat je velmi jednoduchá a má vysoké účinky.

Augmentace

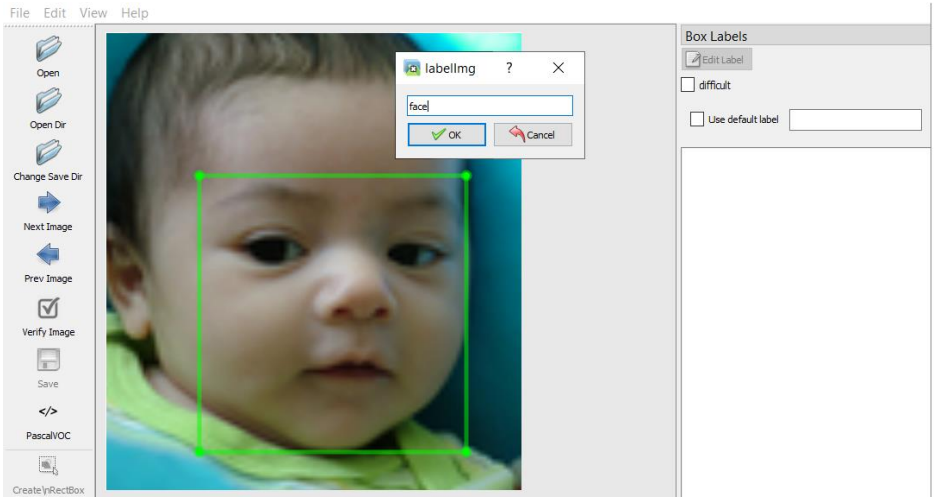
Je aplikovaná téměř na všechny datasety a je jí dosáhnuto prostými transformacemi obrázku. Problém nastává, při výběru správné techniky augmentace pro dostupný dataset.
Nalezení správné metody zabírá velké množství času a zkušeností. I po několika pokusech nemusí odborník na hluboké učení najít správnou možnost. Efektivní strategie augmentace se u každého datasetu liší a některé techniky augmentace mohou být pro model škodlivé.
Například, při práci s datasetem MNIST aplikování rotační augmentace by vedlo ke zhoršení modelu, protože otočit ‘9‘ o 180 stupňů by nám vytvořilo ‘6‘, ale label by zůstal stejný. Na druhou stranu, aplikovat rotační augmentaci na snímky pořízené satelitem je prospěšné, protože například auto, ať ho otočíte jak chcete, bude seshora pořád auto.

Anotace

Anotace je důležitou součástí hlubokého učení, přece jen pokud nespecifikujeme co se má naše síť naučit nemůžeme očekávat správné výsledky. Například, pokud bychom chtěli mít model, co rozpozná a najde přesné umístění obličeje musíme přesně specifikovat, kde se obličej nachází. Pokud bychom tak neučinily model by se učil podle všeho co je na obrázku, pokud by většina datasetu měla v pozadí automobily mohl by model kategorizovat automobily i obličeje do jedné třídy. Existuje i druhá možnost pokud nechceme anotovat. Princip spočívá v tom, že obrázky budou obsahovat pouze náš hledaný objekt takzvané cropped images.

LabelImg

Labelimg je grafický anotační nástroj, který je vhodný pro anotaci jedné či více class a ohraničení hledaného objektu BND boxem.



Obrázek 19: LabelImg

Postup:

- Označuje se okolo celého objektu. Je lepší, když BND box obsahuje i trochu okolí, než aby byl objekt oříznutý. Takže je potřeba mít BND box co nejtěsněji okolo objektu, ale zároveň nesmí BND box přesahovat dovnitř objektu.

- Pro překryté objekty, je lepší je označit celé. Pokud není objekt vidět z důvodu překrytí jiného, označte cílový objekt jako kdyby vidět byl. Model pak lépe porozumí hranám objektu.

- Objekt, který je částečně mimo obrázek je lepší označit též, i částečný objekt je objekt.

- Pokud zadáváte úlohu označení externě nějaké firmě či spolupracujete na označování s někým jiným, vždy buďte ve svých pokynech explicitní, špatně označený dataset je nepoužitelný.

4 Mask R-CNN

Mask region-based CNN je jeden z modelů, které se používají pro rozpoznání (recognition) objektu. Mask R-CNN projekt lze naleznout na GitHubu [1], obsahuje všechny potřebné knihovny pro tvorbu Keras modelů. Tento přístup může být velmi obtížný, jelikož požaduje specifickou přípravu a rozdělení datasetu. Mask R-CNN má možnost rychlého trénování, kdy převezmeme hotové modely a pouze přetrénujeme na vlastních datech.

Tabulka 1: Použité a potřebné knihovny

Knihovna	**import**		
os	listdir		
xml.etree	ElementTree		
numpy	zeros	asarray	expand_dims
matplotlib	pyplot		
matplotlib.patches	Rectangle		
mrcnn.config	Config		
mrcnn.model	MaskRCNN	mold_image	
mrcnn.utils	Dataset		
cv2	cv2		

Z knihovny `os` používám funkci `listdir`, která slouží k manipulaci složek.

Python poskytuje ElementTreeAPI, který může být použit k parsování a načítání XML souborů, využívat budu hlavně funkci `find()` a `findall()` .

Knihovno `numpy` je používaná pro práci s maticemi.

`mrcnn` knihovna obsahuje samotné funkce Mask R-CNN, např. pro úpravu a načtení datasetu.

`Matplotlib` je matematická knihovna, která obsahuje i možnost dělání grafů.

OpenCV(cv2) knihovna je knihovna, která slouží k úpravě obrazu.

Dataset

Mask R-CNN je navrhnut tak, aby se naučil předvídat jak masku hledaného objektu tak BND box okolo hledaného objektu. V mém případě jsem ale nepoužil možnost masky, ale pouze možnost BND boxu. Z BND boxu jsem vytvořil pseudo-masky, které pokrývají celý BND box, neobtékají jako na obrázku 19. Jak lze vidět na obrázku 19, maska překrývá a obtéká celý objekt, kdyžto BND box pouze ohraničuje objekt.

Obrázek 20: Mask vs BND box [14]

Pokud bychom chtěli aby náš model byl schopný vytvářet masky, museli bychom náš dataset anotovat společně s maskami. Je potřeba několik kroků proto, abychom dataset připravily.

Postup:

1) Stáhnutí datasetu
2) Anotace datasetu
3) Parsování anotačních souborů
4) Vytváření objektu/classy face
5) Otestování datasetu

Stáhnutí datasetu

Dataset, který jsem použil jsem získal ze stránky Kaggle, v [2]. přikládám odkaz na dataset. Dataset obsahuje 70 tisíc obrázků lidských obličejů různé etnicity, pohlaví, národnosti, stáří a věku. Obrázky mají velikost 128x128 pixelů.

Anotace datasetu

Pro anotaci jsem využil program LabelImg zmíněný v kapitole 5.3. Anotační informace jsem uložil do XML souboru, pro každý obrázek byl vytvořen jeden.

```
<annotation>
	<folder>thumbnails128x128</folder>
	<filename>00000.png</filename>
	<path>C:\Users\42072\Desktop\bp_benes_jan\Face_detection\Face_dataset\thu
mbnails128x128\00000.png</path>
	<source>
		<database>Unknown</database>
	</source>
	<size>
```

```
            <width>128</width>
            <height>128</height>
            <depth>3</depth>
      </size>
      <segmented>0</segmented>
      <object>
            <name>face</name>
            <pose>Unspecified</pose>
            <truncated>0</truncated>
            <difficult>0</difficult>
            <bndbox>
                  <xmin>17</xmin>
                  <ymin>15</ymin>
                  <xmax>98</xmax>
                  <ymax>109</ymax>
            </bndbox>
      </object>
</annotation>
```

Lze vidět, že soubor obsahuje "size" element, který obsahuje velikost obrázku. Dále element "object", který popisuje umístění BND boxu okolo classy face, kterou lze nalézt pod "name".

Rozdělení datasetu

Aby model byl schopen spočítat mAP je potřeba kromě trénovacího datasetu i validační/testovací dataset, v mém případě jsem si rozdělil 1600 obrázků na učení a 401 na validaci/test.

```
def load_dataset(self, dataset_dir, is_train=True):
  self.add_class("dataset", 1, "face")
  images_dir = dataset_dir + '/Testing_batch/'
  annotations_dir = dataset_dir + '/annots/'
  # find all images
  for filename in listdir(images_dir):
    image_id = filename[:-4]
    if is_train and int(image_id) >= 1600:
      continue
    if not is_train and int(image_id) < 1600:
      continue
    img_path = images_dir + filename
    ann_path = annotations_dir + image_id + '.xml'
    self.add_image('dataset', image_id=image_id, path=img_path,
annotation=ann_path)
```

Funkce `load_dataset()` má dvě funkce:

1. přiřadí do trénovacího datasetu prvních 1600 obrázků i s jeho anotačním XML souborem
2. Přiřadí do validačního datasetu 401 obrázků i s jeho anotačním XML souborem

Parsování anotačních souborů

Dále je potřeba získat souřadnice BND boxů abychom mohli vytvořit masku, která bude

překrývat náš objekt.

```
def extract_boxes(self, filename):
  tree = ElementTree.parse(filename)
  root = tree.getroot()
  boxes = list()
  for box in root.findall('.//bndbox'):
    xmin = int(box.find('xmin').text)
    ymin = int(box.find('ymin').text)
    xmax = int(box.find('xmax').text)
    ymax = int(box.find('ymax').text)
    coors = [xmin, ymin, xmax, ymax]
    boxes.append(coors)
  width = int(root.find('.//size/width').text)
  height = int(root.find('.//size/height').text)
  return boxes, width, height
```

Funkce `extract_boxes()` parsuje umístění BND boxů v obrázků, souřadnice jednotlivých boxů jsou ukládány do listu, protože jeden obrázek může obsahovat jeden či více obličejů. Funkce vrací seznam boxů a velikost obrázku.

Vytváření masky

Jelikož maska je matice nul a jedniček, kde nula symbolizuje kde hledaný objekt není a jednička, kde se objekt nachází. Stáčí nám vytvořit matici plnou nul o velikosti našeho obrázku a pomocí souřadnic BND boxu nahradit určitá místa jedničkami. Jelikož náš model se učí pouze na jednu classu “face”, tak víme, že všechny box popisují tuto classu a nemusíme je tedy procházet podle druhů.

```
# load the masks for an image
def load_mask(self, image_id):
  info = self.image_info[image_id]
  path = info['annotation']
  boxes, w, h = self.extract_boxes(path)
  masks = zeros([h, w, len(boxes)], dtype='uint8')
  class_ids = list()
  for i in range(len(boxes)):
    box = boxes[i]
    row_s, row_e = box[1], box[3]
    col_s, col_e = box[0], box[2]
    masks[row_s:row_e, col_s:col_e, i] = 1
    class_ids.append(self.class_names.index('face'))
  return masks, asarray(class_ids, dtype='int32')
```

Funkce `load_mask()`, vrací matici nul a jedniček pro každou classu a jeho BND box. Nakonec musíme vracet s jakým obrázkem jsme vlastně teď pracovali a k tomu slouží funkce `image_reference()`.

```
def image_reference(self, image_id):
  info = self.image_info[image_id]
  return info['path']
```

Všechny tyto funkce jsou součástí classy `FaceDataset()`.

```
class FaceDataset(Dataset):
  def load_dataset(self, dataset_dir, is_train=True):…
  def extract_boxes(self, filename):…
  def load_mask(self, image_id):…
  def image_reference(self, image_id):…
```

Otestování datasetu

Je důležité otestovat, jestli se všechny obrázky správně nahrály, teď vytvoříme trénovací a testovací dataset a otestujeme zdali datasety obsahují správný počet obrázků.

```
train_set = FaceDataset()
train_set.load_dataset('path_to_dataset', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))
test_set = FaceDataset()
test_set.load_dataset('path_to_dataset', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))
```

Pokud správně dopadlo, měli bychom vidět:

```
Train: 1600
Test: 401
```

Nastavení parametru trénování

Mask R-CNN počítá s tím, že máte přístup k velmi výkonnému clusteru složeného z GPU, jelikož ne všichni mají možnost trénovat na výkonných sestavách, je třeba provést config, který upraví parametry trénování.

```
class FaceConfig(Config):
  NAME = "Face_cfg"
  NUM_CLASSES = 1 + 1
  STEPS_PER_EPOCH = 100
  IMAGES_PER_GPU = 1
  IMAGE_MIN_DIM = 128
  IMAGE_MAX_DIM = 128
  LEARNING_RATE = 0.0005
```

Trénování

Před samotným trénováním je potřeba načíst váhy modelu, pokud nechceme učit model kompletně od znova, toto není doporučeno, jelikož váhy jsou předem nastaveny pro rozpoznání obrázku.

```
config = FaceConfig()
config.display()
model = MaskRCNN(mode='training', model_dir='./', config=config)
model.load_weights('mask_rcnn_coco.h5', by_name=True,
          exclude=["mrcnn_class_logits", "mrcnn_bbox_fc", "mrcnn_bbox",
"mrcnn_mask"])
model.train(train_set, test_set, learning_rate=config.LEARNING_RATE, epochs=30,
layers='heads')
```

Jak lze vidět, využíváme váhy 'mask_rcnn_coco.h5' , jelikož ale model nebyl určen přesně

k našemu účelu, je třeba vynechat množství výstupních vrstev. Poté stačí zavolat .train a čekat dokud nebude model natrénován, je ale dobré kontrolovat ztrátu každé epochy, jelikož model by mohl mít tendenci se přeučit pokud by například byl dostatečně přesný na 19 epoše a učil by se dál, mohl by nastat problém, kdy model je skvělý na trénovacích datech, ale mizerný na testovacích. Po každé epoše se model uloží ve formátu `mask_rcnn_face_cfg_0029.h5` . Při různém ladění jsem došel k závěru, že pro lepší ztrátové skóre bych musel použít větší množství obrázků v datasetu, pro 29 epochu jsem byl schopen ztrátu snížit až na 0.1. Výpis epoch vypadá takto.

```
1/50 [..............................] - ETA: 13:47 - loss: 5.7114
```

Obrázek 21: Mask_R-CNN epocha

kromě loss výpis obsahuje spousty dalších parametrů, ale ty nejsou pro nás zatím důležité.

mAP, přesnost modelu

Výkon modelu pro úlohu rozpoznávání objektů se často hodnotí pomocí střední absolutní přesnosti neboli mAP.

Předpovídáme BND boxy, abychom mohli určit, zda je předpověď BND boxu dobrá nebo ne, na základě toho, jak dobře se předvídané a skutečné BND boxy překrývají. To lze vypočítat vydělením oblasti překrytí celkovou plochou obou ohraničujících rámečků nebo průsečíkem(překryté oblasti) vyděleným sjednocením, označovaným jako „intersection over union“ nebo IoU. Předpověď perfektního BND boxu bude mít IoU 1.

Je standardní předpokládat pozitivní predikci BND boxu, pokud je IoU větší než 0,5, např. překrývají se o 50% nebo více.

Přesnost označuje procento správně předpovězených ohraničujících rámečků (IoU> 0,5) ze všech předpovězených BND boxů.

Průměrná přesnost (AP) napříč všemi obrázky v datové sadě se nazývá „mean average“ přesnost nebo mAP.

Knihovna mask-rcnn poskytuje `mrcnn.utils.compute_ap` pro výpočet AP a dalších metrik pro dané obrázky. Toto skóre AP lze sbírat napříč datasetem a průměr vypočítat, aby poskytlo představu o tom, jak dobrý je model při detekci objektů dané classy.

Nejprve musíme místo `FaceConfig(Config):` definovat nový objekt `PredictionConfig(Config):,` který se použije pro vytváření předpovědí. Můžeme rozšířit naši dříve definovanou `FaceConfig(Config):,` abychom znovu použili parametry. Místo toho definujeme novou classu se stejnými hodnotami, aby byl kód kompaktní. Konfigurace

musí změnit některé výchozí hodnoty , které se liší od toho, jak jsou nastaveny pro trénování modelu (bez ohledu na to, zda používáme GPU nebo CPU).

```
class PredictionConfig(Config):
  NAME = "face_cfg"
  NUM_CLASSES = 1 + 1
  GPU_COUNT = 1
  IMAGES_PER_GPU = 1
```

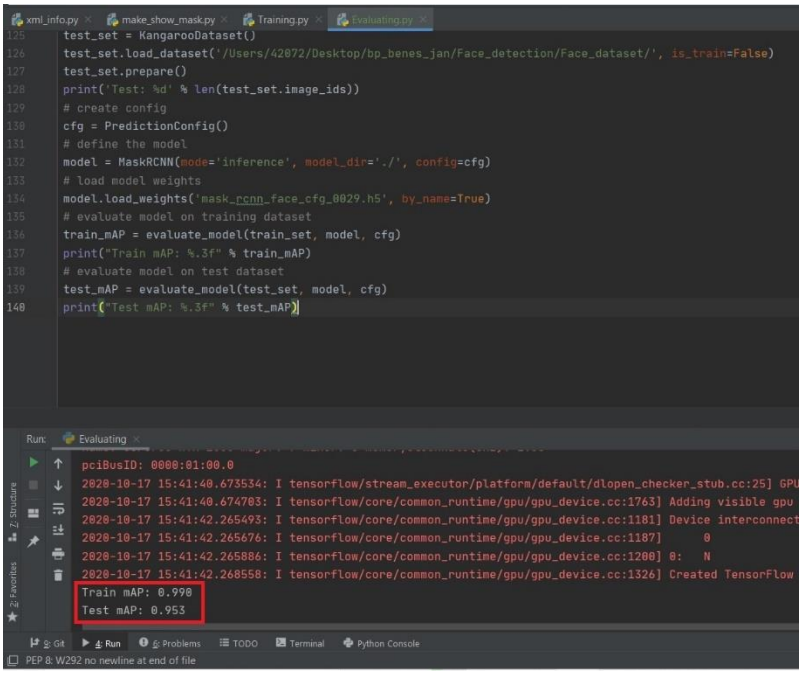
Dále je potřeba nadefinovat funkci, která spočítá mAP, je třeba načíst BND box a masku obrázku, poté je třeba upravit obrázek na stejnou velikost jako která byla použita při trénování. Nakonec je zavolána funkce `compute_ap()`, která vrátí AP daného obrázku.

```
def evaluate_model(dataset, model, cfg):
  APs = list()
  for image_id in dataset.image_ids:
    image, image_meta, gt_class_id, gt_bbox, gt_mask =  load_image_gt(dataset,
cfg, image_id, use_mini_mask=False)
    scaled_image = mold_image(image, cfg)
    sample = expand_dims(scaled_image, 0)
    yhat = model.detect(sample, verbose=0)
    r = yhat[0]
    AP, _, _, _ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["rois"],
r["class_ids"], r["scores"], r['masks'])
    APs.append(AP)
  mAP = mean(APs)
  return mAP
```

Nakonec je třeba použít správný config načíst model a použít vlastní natrénované váhy, poté zavoláme funkci `evaluate_model()` pro trénovací a validační dataset.

```
cfg = PredictionConfig()
model = MaskRCNN(mode='inference', model_dir='./', config=cfg)
model.load_weights('mask_rcnn_face_cfg_0029.h5', by_name=True)
train_mAP = evaluate_model(train_set, model, cfg)
print("Train mAP: %.3f" % train_mAP)
test_mAP = evaluate_model(test_set, model, cfg)
print("Test mAP: %.3f" % test_mAP)
```

Jak lze vidět z obrázku 21, mAP je pro trénovací data 0.99 a pro evaluační data 0.953, lze tedy říct, že model je schopný rozeznávat obličeje na 95,3%.



Obrázek 22: CNN_mAP

Detekce

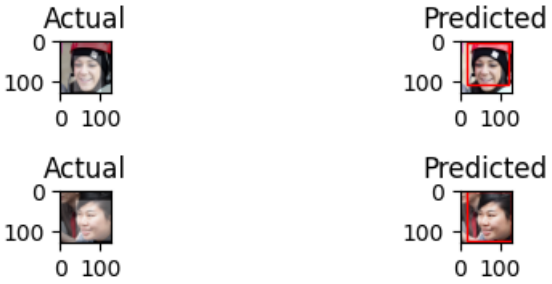
Pro vizuální ukázku přesnosti a detekce modelů slouží funkce plot_actual_vs_predicted(). Která vezme předpovězené souřadnice BND boxu obrázku a následně je překryje s původním obrázkem do grafu. Vizuálně vykreslí reálné vs předpovězené.

```
def plot_actual_vs_predicted(dataset, model, cfg, n_images=5)…
```

Poté stačí zavolat funkci `plot_actual_vs_predicted()`, jako parametry vložíme o anotovaný trénovací dataset a anotovaný testovací dataset.

```
plot_actual_vs_predicted(train_set, model, cfg)
plot_actual_vs_predicted(test_set, model, cfg)
```

Výsledek je vidět na obrázku 23.



Obrázek 23: CNN_actual_vs_predicted

Jak lze vidět, model se natrénoval na rozpoznávání „obličejů“, problém nastal, že jsem očekával model, který bude detekovat pouze obličeje, jenže díky mé až moc obecné anotaci,

kde jsem označoval nejen obličej ale celou hlavu se model naučil rozpoznávat lidské hlavy. Jelikož mAP vyšlo obstojné, myslel jsem si, že je přesnost skvělá, problém vznikl na mojí straně, jelikož jsem modelu poskytl nepřesná data. Model jsem později využil jako anotační/cropp nástroj na Haar přístup.

Detekce v reálném čase

Pro detekci v reálném čase potřebujeme dostávat obrázky/snímky z kamery v reálném čase. K tomu využijeme knihovnu cv2. Config zůstane stejný jako PredictionConfig(Config): .Dále si definuji funkci `classify_image():`

```
def classify_image(image,model,cfg):
  scaled_image = mold_image(image, cfg)
  sample = expand_dims(scaled_image, 0)
  # make prediction
  yhat = model.detect(sample, verbose=0)[0]
  return yhat['rois']
```

Funkce `classify_image():` bere jako vstupní parametry, vstupní obrázek – image, model, a config – cfg. Funkce vrací souřadnice BND boxu. Další funkce je `img_bnd_highlight():`.

```
def image_bnd_highlight(image,coordinates):
  for box in coordinates:
    y1, x1, y2, x2 = box
    new_img = cv2.rectangle(image,(x1,y1),(x2,y2),(255,255,255),5)
    return new_img
```

Funkce `img_bnd_highlight():` bere jako vstupní parametry, vstupní obrázek – image a souřadnice, které poskytne funkce `classify_image():`

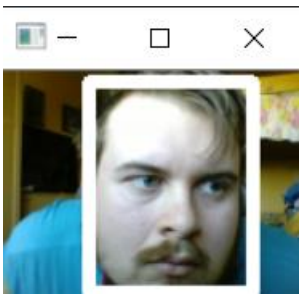
Inicializace modelu pak vypadá stejně jako v kapitole 6.3. Teď už stačí jen vše dát dohromady.

```
cap = cv.VideoCapture(0)

while True:
  ret, frame = cap.read()
  if not ret:
    print("Can't receive frame (stream end?). Exiting ...")
    break
  r = 150.0 / frame.shape[1]
  dim = (150, int(frame.shape[0] * r))
  resized = cv.resize(frame, dim, interpolation=cv.INTER_AREA)
  coords = classify_image(resized,definitive_model,cfg)
  image = image_bnd_highlight(resized,coords)
  cv.imshow('frame', image)
  if cv.waitKey(1) == ord('q'):
    break
```

```
cap.release()
cv.destroyAllWindows()
```

Díky openCV zapneme stream kamery, nastavuji čtení na 2 snímky za vteřinu. Ve while loop ukládám z kamery obrázek do proměnné frame a testuji, jestli se snímek správně načetl, pokud ne, ukončím program. Dále upravuji dimenzi/velikost obrázku pro zrychlení výpočtu. Poté volám funkce `classify_image():` a funkci `img_bnd_highlight():` a nakonec zobrazím výsledek.



Obrázek 24: CNN_real_time_prediction

5 Haar like features

Detekce objektů pomocí kaskádových klasifikátorů založených na Haar features je efektivní metoda detekce objektů navržená Paulem Violou a Michaelem Jonesem v článku „Rapid Object Detection using a Boosted Cascade of Simple Features“ v roce 2001. Jedná se o přístup založený na strojovém učení, kde funkce kaskády je trénována z mnoha pozitivních a negativních obrazů. Poté se používá k detekci objektů na jiných obrázcích.

Použité a potřebné knihovny

Pro haar přístup v pythonu nám stačí pouze OpenCV knihovna, všechny ostatní potřebné funkce obsahuje python ve svém základu.

Dataset

Dataset pro haar like features je rozdílný oproti CNN přístupu, zde potřebujeme nejenom positive dataset, tedy dataset, který obsahuje náš hledaný objekt, ale i negativní dataset, který obsahuje vše kromě hledaného objektu. Pro pozitivní dataset byl použit stejný jako v CNN zdroj [2]. Pro negativní dataset byl použit dataset z kaggle, zdroj [3].
Postup pro úpravu positive datasetu je podobný jako u CNN.

1) Stáhnutí datasetu
2) Převedení do grayscale
3) Anotace/cropp datasetu
4) Vytvoření vektorového souboru

Stáhnutí datasetu

Dataset je stejný jako ten co byl použit u Mask_R-CNN.

Převedení do grayscale

Jelikož OpenCV haar classifier potřebuje positive dataset v grayscale, je třeba ho tedy převést. Jelikož OpenCV je knihovna pro práci s obrazem, je možné upravovat hodnota každého pixelu zvlášť, takže je možné převést obrázek do grayscale.

```
import os,cv2
path = r'p_images'
```

```
dstpath = r'./haar_cascade/dataset/positive'

try:
  makedirs(dstpath)
except:
  print ("Directory already exist, images will be written in same folder")

files = os.listdir(path)

for image in files:
  img = cv2.imread(os.path.join(path,image))
  gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  cv2.imwrite(os.path.join(dstpath,image),gray)
```

Jak lze vidět, konverze je celkem jednoduchá. Ze složky `p_images` vezmeme všechny obrázky a po převedení do grayscale je uložíme v `pg_folder`.

Anotace/cropping datasetu

Jelikož, jsem už jeden dataset anotoval, a jsou to ty samé datasety, tak se mi nechtělo ručně a pracně anotovat celý dataset znovu. Proto jsem využil starý CNN model, aby mi tento nový dataset ořízl, neboli zastřihnul.

```
import cv2 as cv
from Live_Detection_CNN_Functions import definitive_model
from Live_Detection_CNN_Functions import cfg
from Live_Detection_CNN_Functions import classify_image
from Live_Detection_CNN_Functions import image_bnd_highlight
import os

scanned_dir = sorted(os.listdir('./haar_cascade/dataset/positive'))
i = 0
for item in scanned_dir:
  i = i + 1
  path = './haar_cascade/dataset/positive/' + item
  image = cv.imread(path)
  coords = classify_image(image, definitive_model, cfg)
  cropped = image[coords[0][0]:coords[0][2], coords[0][1]:coords[0][3]]
  cv.imwrite('./test/'+ item,cropped)
  if i > 10:
    break
```

Takto vypadá kód pro testování prvních 10 obrázků. Princip spočívá v tom, že použiji model pro detekci obličejů a najdu souřadnice BND boxu a přepíšu originální obrázek tím, co je v BND boxu.

Obrázek 25: originál_vs_cropped [15]

Vytvoření vektorového souboru

Haar classifier potřebuje pro trénování .vec soubor s údaji, kde se nachází obličej na obrázku, tím že, jsem obrázky neanotoval, ale pouze ořezal ušetřím si starosti a práci s vytvářením .txt souboru, který obsahuje souřadnice BND boxu. Abych mohl vytvořit .txt soubor a nezjišťoval, jaký má každý obrázek rozměry, všechny obrázky převedu na velikost 80x85.

```
import cv2 as cv

scanned_dir = sorted(os.listdir('./haar_cascade/dataset/positive'))
i = 0


for item in scanned_dir:
  path = './haar_cascade/dataset/positive/' + item
  image = cv.imread(path)
  imgResize = cv.resize(image,(80,85))
  cv.imwrite('%04i.png' % i, imgResize)
  i = i + 1
```

Po všech těchto úpravách máme obrázky, kde je pouze grayscale obličej o velikosti 80x85. Můžeme tedy vytvořit info.txt, tedy textový soubor, který obsahuje všechny obrázky a souřadnice BND boxů.

```
import os

if __name__ == '__main__':
  file = open('info.txt', 'w+')
  scanned_dir = sorted(os.listdir('./p'))
  for item in scanned_dir:
    line = 'positive/' \
        + item + \
        ' 1 ' + '0' + \
        ' ' + '0' + \
        ' ' + '80' + \
        ' ' + '85' + \
        '\n'
    file.write(line)
```

info.txt má potom takovýto tvar, kde první číslo říká, kolik se tam nachází objektů a další čísla jsou souřadnice xmin, ymin, xmax, ymax.

positive/00000.png 1 0 0 80 85

positive/00001.png 1 0 0 80 85

positive/00002.png 1 0 0 80 85

……

Dále je třeba vytvořit bg.txt soubor, který obsahuje pouze názvy negativního datasetu, jelikož haar classifier si vezme negativní obrázek a vybere si pouze jeho část, proto není nutné anotovat/croppovat negativní dataset, na velikosti nezáleží. Ještě před vytvoření bg.txt si dataset přejmenuji, aby byl přehlednější.

```
import os

if __name__ == '__main__':
  path = './negative'
  i = 0
  for file in os.listdir(path):
    os.rename(os.path.join(path, file), os.path.join(path,str(i)+'.png'))
    i = i + 1;
```

A teď můžeme vytvořit bg.txt soubor, který obsahuje všechny negativní obrázky a má takovýto tvar:

0.png

1.png

2.png

……..

```
import os

if __name__ == '__main__':
  file = open('bg.txt', 'w+')
  scanned_dir = sorted(os.listdir('./negative'))
  for item in scanned_dir:
    file.write(str(item) + '\n')
```

Teprve teď se může nechat vytvořit vektorový soubor. Soubor se vytvoří pomocí příkazu:

```
opencv_createsamples -info info.txt -num 2001 -w 80 -h 85 -vec positives.vec
```

positives.vec obsahuje všechny informace o datasetu:

-num 2001: počet positivních obrázků

-w/h: šířka/výška obrázku

Trénování Haar classifier

Pro funkci detektoru potřebujeme .xml soubor, který obsahuje všechny relevantní features hledaného objektu. Trénink začne pomocí příkazu:

```
opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 2001 -
numNeg 3029 -numStages 30 -w 80 -h 85
```

Kde:

-numPos je počet positivních obrázků, které se využijí s positives.vec

-numNeg je počet negativních obrázků, které se využijí s bg.txt

-numStages je počet epoch

Je třeba podotknout, že potřebný čas na trénování se s každou další epochou narůstá, pokud bychom měli například 4 epochy a první epocha trvala 10 minut, tak druhá by trvala 20 minut a třetí 40 minut, čtvrtá by trvala 50 minut , je to z toho důvodu, že nová epocha vždy prochází všechny předchozí a pouze k nim přidává, samozřejmě záleží na výpočetním výkonu. Trénovací log pak vypadá přibližně takto, v příloze A.

Testování modelu

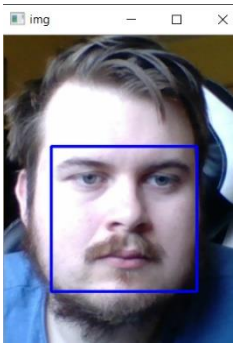
Otestovat model lze velmi snadno, stačí nám nahrát náš .xml soubor a zavolat detekci.

```
import cv2

object_cascade = cv2.CascadeClassifier('face.xml')
cap = cv2.VideoCapture(0)

while:
  ret, img = cap.read()
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  object = object_cascade.detectMultiScale(gray, 50, 50)
  for (x, y, w, h) in object:
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 255, 0), 2)
  cv2.imshow('img', img)
  k = cv2.waitKey(30) & 0xff
  if k == 27:
    break
cap.release()
cv2.destroyAllWindows()
```

Na obrázku 26 lze vidět výsledek.



Obrázek 26: haar_detekce

6 Integrace do Raspberry Pi 3 b

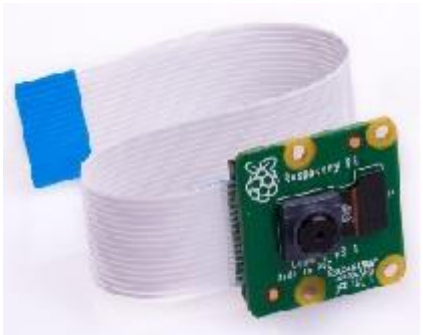
Důvod, proč jsem vybral Raspberry, je ten, že je to ve své podstatě miniatura počítače. Jelikož CNN model měl rychlost 4 snímky za vteřinu, což je pro živý přenos nedostatečné, zvolil jsem postup haar classifierem. Na Raspberry funguje a je podporovaná plná verze pythonu, na kterém je knihovna OpenCV.

Raspberry Pi

Hlavním prvkem pro tvorbu kompaktního prototypu pro detekci obrazu je Raspberry Pi model 3 b & Raspberry Pi Camera V2.1 sensor. Na obrázku 26 lze vidět Raspberry Pi model 3 b, který byl vyroben firmou Raspberry Pi foundation. Tento model 3 b je třetí generací Raspberry mikropočítačů. Na první pohled se od sebe verze moc neliší, ale třetí generace je 10x výkonnější než první generace. Navíc model 3 obsahuje Wi-Fi bezdrátový modul a Bluetooth, což se skvěle hodí pro vzdálené připojení.

Obrázek 27: Raspberry_Pi_b3 [16]

Mezitím na obrázku 27, je vidět použitá kamera která je vybavena vysoce kvalitním 8megapixelovým obrazovým snímačem Sony IMX219. Je schopna zachytit statické obrázky o rozměrech 3280 x 2464 pixelů a podporuje videa 1080p30, 720p60 a 640x480p60 / 90. Díky své malé velikosti a lehkosti (váží něco přes 3 g) je vhodná pro přenosné aplikace. Lze jej připojit k Raspberry Pi pomocí krátkého plochého kabelu.



Obrázek 28: Raspberry_camera_V2.1 [17]

Raspberry funguje na operačním systému Rasbian, což je bezpoplatkový operační systém založen na Debianu optimalizovaný pro Raspberry Pi hardware. Jak už jsem se zmínil, obsahuje nejenom python IDE ale i základní programy a nástroje pro běh Raspberry Pi. Aby bylo možné provozovat správný chod detektoru, je potřeba nainstalovat následující knihovny, NumPy, OpenCV a picamera.

Nastavení

Předtím, než se nahraje jakýkoliv kód/model na Raspberry je potřeba nainstalovat operační systém. Raspberry obsahuje slot na SD kartu, kam se ukládá jak samotný systém, tak i všechny programy a soubory. Prvně potřebujeme SD kartu formátovat, aby se mohl nahrát Rasbian. Raspberry doporučuje použít jejich oficiální jejich oficiální formáter a burner Raspberry Pi Imager. Imager obsahuje poslední verzi systému Rasbian a další verze systému. Pokud bychom chtěli vlastní verzi systému, nebo starší verzi, která už není v nabídce museli bychom formátovat ručně. Po nahrání Rasbianu na SD kartu, kartu vložíme do určeného slotu a zapneme Raspberry, po krátkém bootu a nastavení se pak stačí přihlásit. Po vyzvání pro přihlášení je uživatel `pi` a heslo je `raspberry.`

Po přihlášení do sytému je důležité updatovat systém a nainstalovat potřebné dependencies a zároveň všechny potřebné knihovny, pokud bychom používali jinou kameru, než oficiální, bylo by třeba nainstalovat speciální ovladač a zjistit, jestli je vůbec podporovaná.

1. Update os

```
sudo apt-get update
sudo apt-get upgrade
```

2. Instalace dependencies

```
sudo apt-get install build-essential cmake pkg-config
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt-get install libgtk2.0-dev libgtk-3-dev
sudo apt-get install libatlas-base-dev gfortran
```

3. Instalace Python 3 a Pip3

```
sudo apt-get install python3-dev
sudo apt-get install python3-pip
```

4. Instalace OpenCv

```
pip3 install opencv-python
```

5. Extra dependencies pro OpenCv

```
sudo apt-get install libqtgui4
sudo apt-get install libqt4-test
```

Pro otestování funkčnosti, stačí otevřít Thonny Python IDE a spustit jednoduchý program.

```
import cv2

cap = cv2.VideoCapture(0)

while True:
      _, img = cap.read()
      cv2.imshow('img', img)
      k = cv2.waitKey(30) & 0xff
   if k == 27:
    break

cap.release()
cv2.DestroyAllWindows()
```

Pokud vše dopadlo dobře, je vidět živý přenos kamery a stisknutím ESC program ukončíte.

Detekce

Po nahrání originálního kódu, který jsem testoval na stolním počítači jsem zjistil, že Raspberry má stále i přes snížené nároky haar přístupu pomalé výpočty a zvládá přibližně osm snímku za vteřinu, to pro mne bylo ale málo, tak jsem zapřemýšlel, jak bych mohl snížit výpočetní nároky, nebo naopak zvýšit výkon Raspberry. První co mne napadlo, je možnost, že nemám dost výkonné napájení, což se ukázalo, že nemám, ale i po výměně napájení problém přetrvával a nebylo vidět žádné zlepšení. Druhý nápad byl ten, že Raspberry má 4 jádra, a já jsem využíval pouze jedno, po dlouhé rešerši jsem došel k závěru, že pokud nechci svému garantovi dlužit nové Raspberry, tak je lepší neexperimentovat se školním majetkem. Poslední možností bylo zkusit upravit kód samotný, hned mne napadlo, že pokud bych snížil plochu výpočtu, tak bych i drasticky snížil dobu výpočtu. Tak jsem postupně začal snižovat rozlišení kamery, než jsem došel k uspokojujícímu výsledku, problém nastal, že kvalita obrazu byla otřesná. Teď už jsem ale věděl, že snižovat velikost vstupu je klíčem k úspěchu, potřeboval jsem tedy snížit velikost vstupních dat, ale na zároveň na výstupu zanechat originální velikost. Tak jsem začal na internetu hledat, jakým způsobem by se dal zmenšit obrázek a následně zvětšit co s nejmenší ztrátou, odpověď zní Aspect Ratio.

Aspect ratio, tedy formát obrazu říká, jaký je ideální poměr stran a jaká může být minimální komprese obrázku předtím, než bude nevratná. Po pár experimentech jsem dospěl k závěru, že vratné snížení/zvětšení je přibližně 25%. Proto jsem napsal funkci, která vezme originální obrázek, zmenší ho o 25% následně udělá detekci a před posláním na výstup ho zvětší zpět

i s oblastí, kde byl nalezen náš hledaný objekt, proto je důležité Aspect ratio, protože jinak bych musel pracně počítat, o kolik se zvětšil BND box a kam ho mám po detekci přesunout.

```
def rescale_frame(frame, percent=75):
  width = int(frame.shape[1] * percent / 100)
  height = int(frame.shape[0] * percent / 100)
  dim = (width, height)
  return cv2.resize(frame, dim, interpolation=cv2.INTER_NEAREST)

def aspect_ratio(img_orig, img_resized):
  …
  return vertical_scale, horizontal_scale
```

Funkce `rescale_frame` nám zmenší obrázek o 25% a funkce `aspect_ratio` nám vrátí hodnotu, o kolik se původní obrázek zmenšil, `vertical_scale` a `horizontal_scale` nám říkají, o kolik bude potřeba finální obrázek zvětšit, abychom se vrátily k původní velikosti.

Cenzura

Jelikož model je už schopen úspěšně najít obličeje bez toho, aniž by se zadrhával, tak je těžší část splněna. Teď už si s detekovaným objektem můžeme dělat co chceme, vystřihávat obličeje z fotek, přidávat na obličeje filtry a mnohé další. Já si vybral cenzuru, nejde o nic jiného, než že oblast, kterou obklopuje BND box přetvoříme k nerozeznání. Můj přístup je takový, že každý pixel si náhodně vybere svého souseda a převezme jeho hodnotu, nastavováním rozsahu lze docílit jemné, střední či hrubé cenzury, obrázek 28.

Obrázek 29: Cenzura

Samozřejmě model není dokonalý, jelikož je trénován na přední stranu obličeje, nefunguje na profil, tudíž jsem do programu přidal nastavitelnou ochranu, pokud by došlo k tomu, že obličej není detekován, ale předtím byl, tak cenzurovaná oblast zůstane namístě dokud se neobjeví nová detekovaná oblast, nebo dokud nedojde nastavený čas na ochraně.

7 Závěr

Zhodnocení

Pomocí naučených teoretických poznatků a rešerše byl vytvořen model na detekci lidských obličejů, který je založen na principu Haar features. Model byl trénován na mém osobním přenosném počítači s využitím CPU, byl použit dataset obsahující 2001 pozitivních obrázků a 1000 negativních obrázků. Následně nalezený obličej byl podstoupen cenzuře. Celý model byl poté integrován do mikropočítače Raspberry Pi 3 b, ke kterému byla připojena Raspberry Pi camera v2.1.Výsledná rychlost rozpoznání vychází na +30 FPS.

Možný další rozvoj

Model by bylo možné zrychlit, pokud by byl k dispozici lepší mikrokontroler/mikropočítač. Dále by bylo možné vytvořit stabilní úschovnu plus přidat baterii k Raspberry, aby bylo vskutku přenosné a nezávislé na přístupu k napájení. Poslední možné zlepšení co mne napadá je rozšíření datasetu nejenom na přední část obličeje, ale také na profil a další možné úhly.

Směr práce

Celá bakalářská práce byla o seznámení se základními principy detekování a úpravy obrazu. Po nabití základních znalostí a zkušeností bych rád pokračoval v rozvoji svých znalostí a pracím tomuto projektu podobným. Například kontrola ospalosti řidiče automobilu nebo jestli osoba nosí/nenosí předepsané pokrývky těla, například jestli nosí ve veřejných prostorách respirátor, nebo jestli nosí správný druh plavek na koupališti.

Seznam literatury a informačních zdrojů

Seznam literatury a použitých vizuálů

[1] *Understanding and Visualizing Neural Networks in Python. Analytics Vidhya - Learn Machine learning, artificial intelligence, business analytics, data science, big data, data visualizations tools and techniques. Analytics Vidhya* [online]. Copyright ©2013 [cit. 30.04.2020]. Dostupné z: https://www.analyticsvidhya.com/blog/2019/05/understanding-visualizing-neural-networks/

[2] Holczer, Balasz. *Artificial intelligence Masterclass* [Online]. [cit. 25.05.2021]. https://globalsoftwaresupport.teachable.com/p/artificial-intelligence-masterclass

[3] Girija Shankar Behera. Medium blog about AI. [Online]. [cit. 25.05.2021]. https://gshbehera.medium.com/

[4] CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.

[5] Gary Bradski and Adrian Kaehler. 2013. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library* (2nd. ed.). O'Reilly Media, Inc.

[6] Gareth Halfacree and Eben Upton. 2012. *Raspberry Pi User Guide* (1st. ed.). Wiley Publishing.

[7] Frank Millstein. 2018. *Convolutional Neural Networks In Python: Beginner's Guide To Convolutional Neural Networks In Python*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA.

[8] Robert Laganiere. 2014. *OpenCV Computer Vision Application Programming Cookbook* (2nd. ed.). Packt Publishing.

[9] CNN_topologie [Online]. [cit. 25.05.2021].
https://medium.com/@ankit.bhadoriya/handwritten-digits-recognition-recepi-using-tensorflow-7b67c7a7532a

[10] MNIST_číslo_1 [Online]. [cit. 25.05.2021].
https://www.programmersought.com/article/729198419/

[11] Druhy_kernelů [Online]. [cit. 25.05.2021].
https://en.wikipedia.org/wiki/Kernel_(image_processing)

[12] Druhy_Haar_features [Online]. [cit. 25.05.2021].
https://commons.wikimedia.org/wiki/File:VJ_featureTypes.svg

[13] Druhy_získávání_dat [Online]. [cit. 25.05.2021].
https://uxdesign.cc/synthetic-faces-facial-recognition-and-the-lesson-of-hikikomori-233c2197f513

[14] Mask_vs_BND_box [Online]. [cit. 25.05.2021]. [1]

[15] Originál_vs_cropped [Online]. [cit. 25.05.2021]. [2]

[16] Raspberry_Pi_model_3b [Online]. [cit. 25.05.2021].
https://rpishop.cz/raspberry-pi-3b/283-raspberry-pi-3-model-b-64-bit-5060214370028.html

[17] Raspberry_Pi_camera_v2.1 [Online]. [cit. 25.05.2021].
https://rpishop.cz/kategorie/329-raspberry-pi-kamera-modul-v2.html

Zdroje

[1] MASK_R-CNN: https://github.com/matterport/Mask_RCNN
[2] Dataset obličejů: https://www.kaggle.com/greatgamedota/ffhq-face-data-set
[3] Negativní dataset: https://www.kaggle.com/muhammadkhalid/negative-images

Seznam obrázků

OBRÁZEK 1: PODMNOŽINY UMĚLÉ INTELIGENCE 2
OBRÁZEK 2:STROJOVÉ UČENÍ PROTI KLASICKÉMU PŘÍSTUPU 2
OBRÁZEK 3: CNN TOPOLOGIE [9] 4
OBRÁZEK 4: MNIST ČÍSLO 1 [10] 5
OBRÁZEK 5: VRSTVY CNN 5
OBRÁZEK 6: KONVOLUCE 6
OBRÁZEK 7: DRUHY KERNELŮ [11] 6
OBRÁZEK 8: PADDING 7
OBRÁZEK 9: RGB KERNEL 7
OBRÁZEK 10: MAXPOOLING 7
OBRÁZEK 11: DRUHY FEATURES [12] 9
OBRÁZEK 12: ČÍSELNĚ VYJÁDŘEN FEATURE 10
OBRÁZEK 13: ZJEDNODUŠENÝ MODEL OBLIČEJE 10
OBRÁZEK 14: INTENZITA PIXELŮ 10
OBRÁZEK 15: IDEÁLNÍ VS SKUTEČNÁ FEATURE 11
OBRÁZEK 16: INTEGRAL IMAGE 12
OBRÁZEK 17: : POSTUP DETEKCE INTEGRAL IMAGE 12
OBRÁZEK 18: DRUHY ZÍSKÁVÁNÍ DAT [13] 15
OBRÁZEK 19: LABELIMG 16
OBRÁZEK 20: MASK VS BND BOX [14] 19
OBRÁZEK 21: MASK_R-CNN EPOCHA 23
OBRÁZEK 22: CNN_MAP 25
OBRÁZEK 23: CNN_ACTUAL_VS_PREDICTED 25
OBRÁZEK 24: CNN_REAL_TIME_PREDICTION 27
OBRÁZEK 25: ORIGINÁL_VS_CROPPED [15] 30
OBRÁZEK 26: HAAR_DETEKCE 32
OBRÁZEK 27: RASPBERRY_PI_B3 [16] 33
OBRÁZEK 28: RASPBERRY_CAMERA_V2.1 [17] 33
OBRÁZEK 29: CENZURA 36

Seznam Tabulek

TABULKA 1: POUŽITÉ A POTŘEBNÉ KNIHOVNY 18

Příloha A.

Trénovací log.

```
POS current samples: 0
……
POS current samples: 2001
NEG current samples: 0
……
NEG current samples: 2000
NEG count : acceptanceRatio  2000 : 0.587199

Precalculation time: 9.572

+----+---------+---------+
| N |  HR   |  FA   |
+----+---------+---------+

|  1|    1|    1|
+----+---------+---------+

|  2|    1|    1|
+----+---------+---------+

|  3|    1|    1|
+----+---------+---------+

|  4|    1|    1|
+----+---------+---------+

|  5|  0.9975|  0.6525|
+----+---------+---------+

|  6| 0.99625|   0.56|
+----+---------+---------+

END>

Training until now has taken 0 days 0 hours 0 minutes 57 seconds.
```

Příloha B.

Celistvý kód pro CNN přístup naleznete na GitHubu:
https://gitlab.fel.zcu.cz/zaverecne_prace/bp_benes_jan

Příloha C.

Celistvý kód pro Haar přístup naleznete na GitHubu:
https://gitlab.fel.zcu.cz/zaverecne_prace/bp_benes_jan